

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 Kaggle班
2024/07/01

振り返りタイム

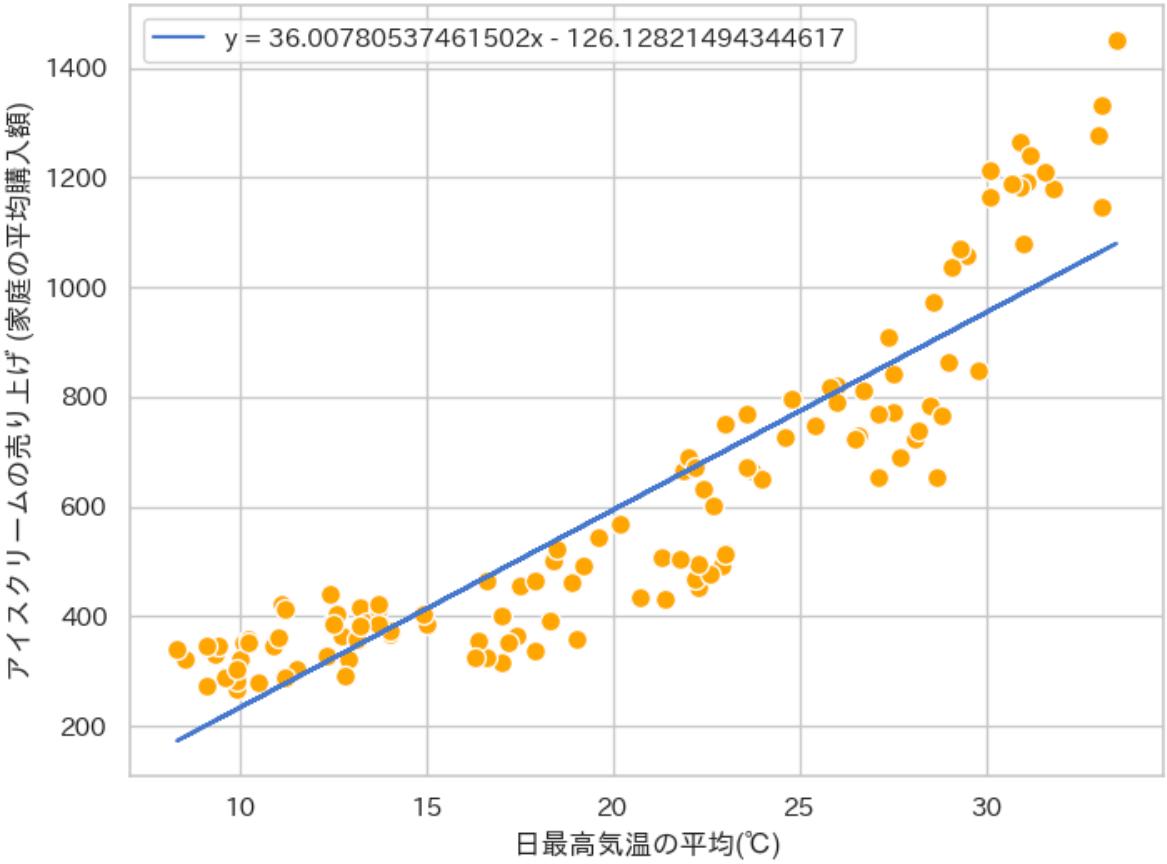
第一回 「学習」

第二回 「勾配降下法」

第三回 「自動微分」

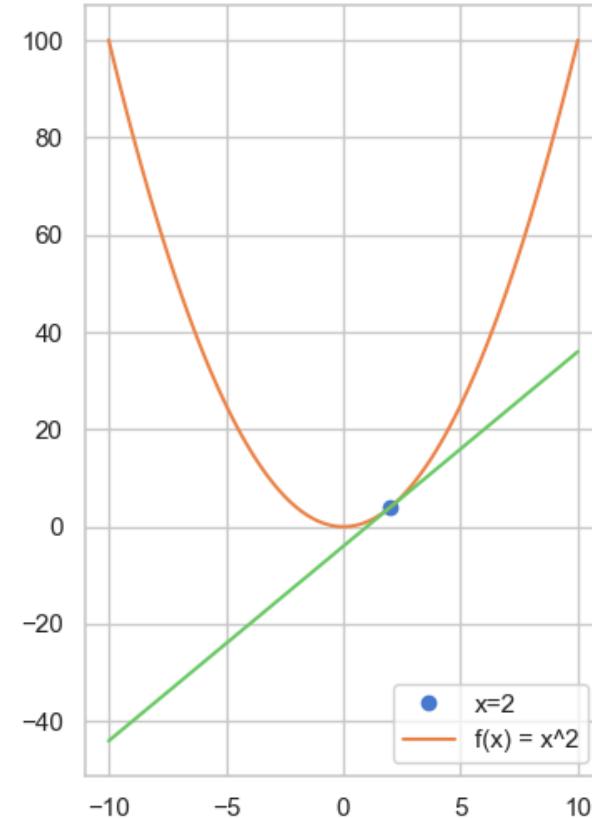
「学習」

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



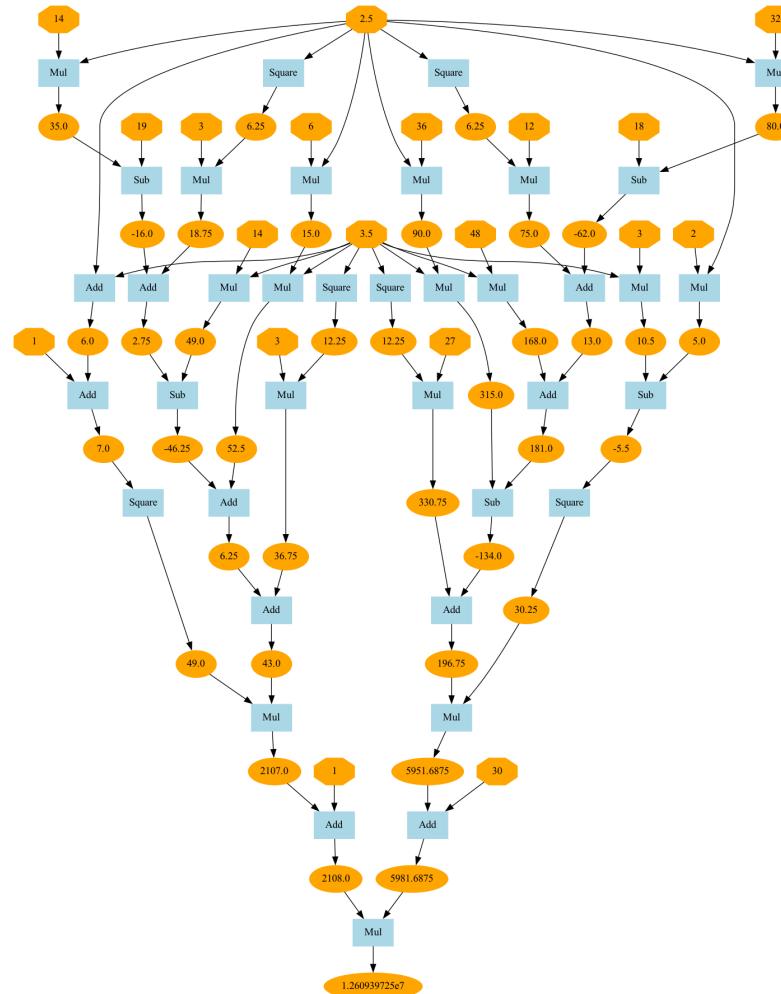
「勾配降下法」

3. 複雑な関数である損失関数を最小にするために、
「勾配降下法」を使ってパラメータを探索した



「自動微分」

4. 自動微分を使うことで、手で微分をしなくて勾配を計算できるようになり、勾配降下法を使えるようになった



振り返りタイム

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 [\leftarrow 今ココ！]

第三回までのまとめ

.....

われわ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 $f(x) = ax + b$ のかたちを仮定してきた（線形回帰）

⇒ われわれの手法はこの仮定に依存しているか？ 🤔



依存していない

線形回帰からの飛躍

我々の手法（自動微分と勾配降下法による学習）で満たすべき条件だったのは…

$L(a, b)$ が a, b について
微分可能である
のみ！



⇒ この条件を満たす関数なら どんなものでも 学習できる！

今日のお話は...

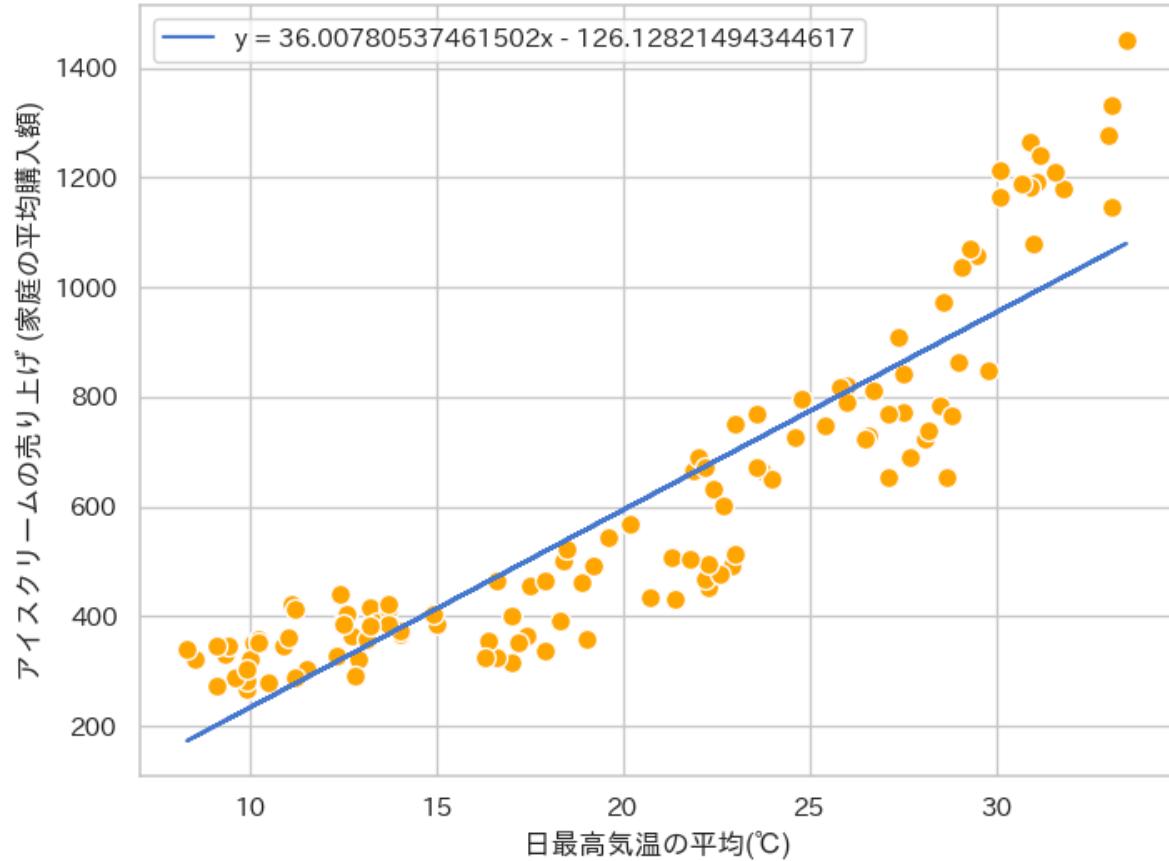
.....

f を変えよう

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

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?

$f(x) = ax^2 + bx + c$ でも大丈夫

$f(x) = \sin(ax + b)$ でも大丈夫

$f(x) = e^{ax+b}$ でも大丈夫

⇒ 直線以外を表現することはできるが

- 二次曲線
- sinカーブ
- 指数カーブ(?)

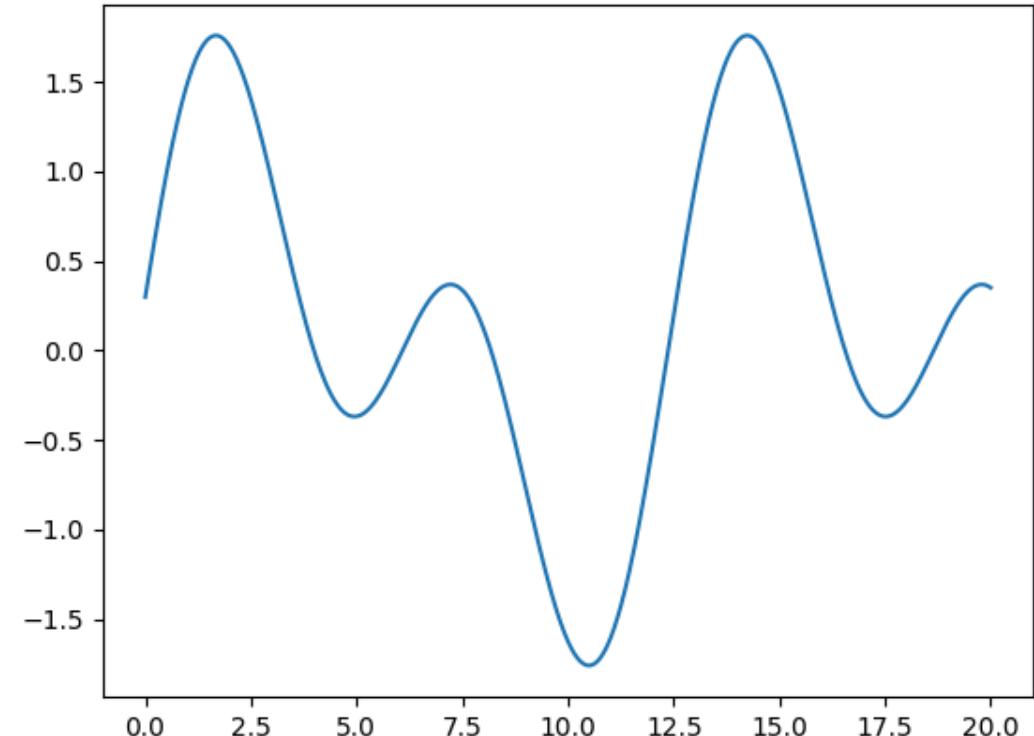
しか表現できない

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



みたいなものは表現できない

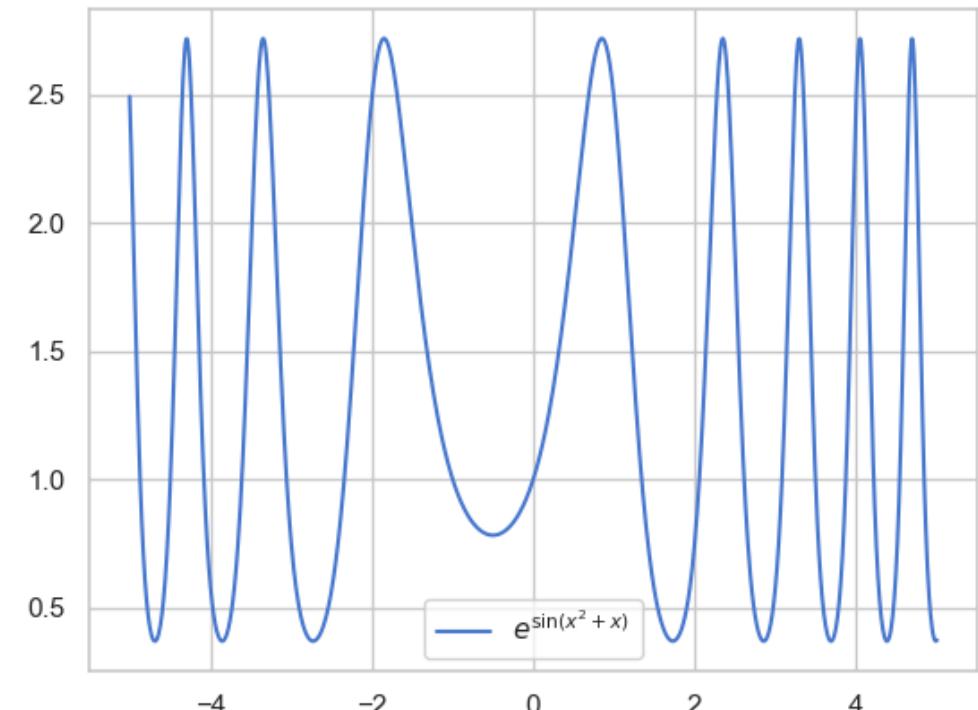


複雑さを生み出す方法

✓ アイデア1: 関数を合成する

$\exp, \sin, x^2 + x$ はそれぞれ単純な関数

👉 一方、合成した
 $h(x) = \exp(\sin(x^2 + x))$ は、



複雑さを生み出す方法

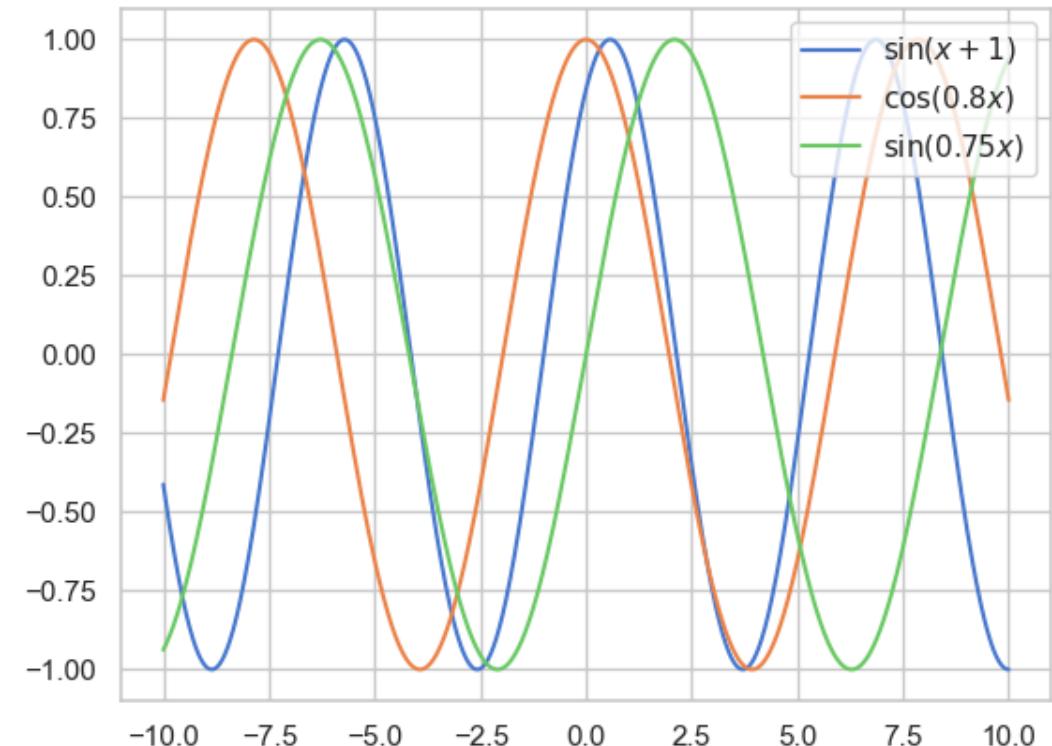
- ✓ アイデア2: 和をとる

複雑さを生み出す方法

三角関数を 3つ用意

- $f_1(x) = \sin(0.5x)$
- $f_2(x) = \cos(0.8x)$
- $f_3(x) = \sin(0.75x)$

それぞれは単純

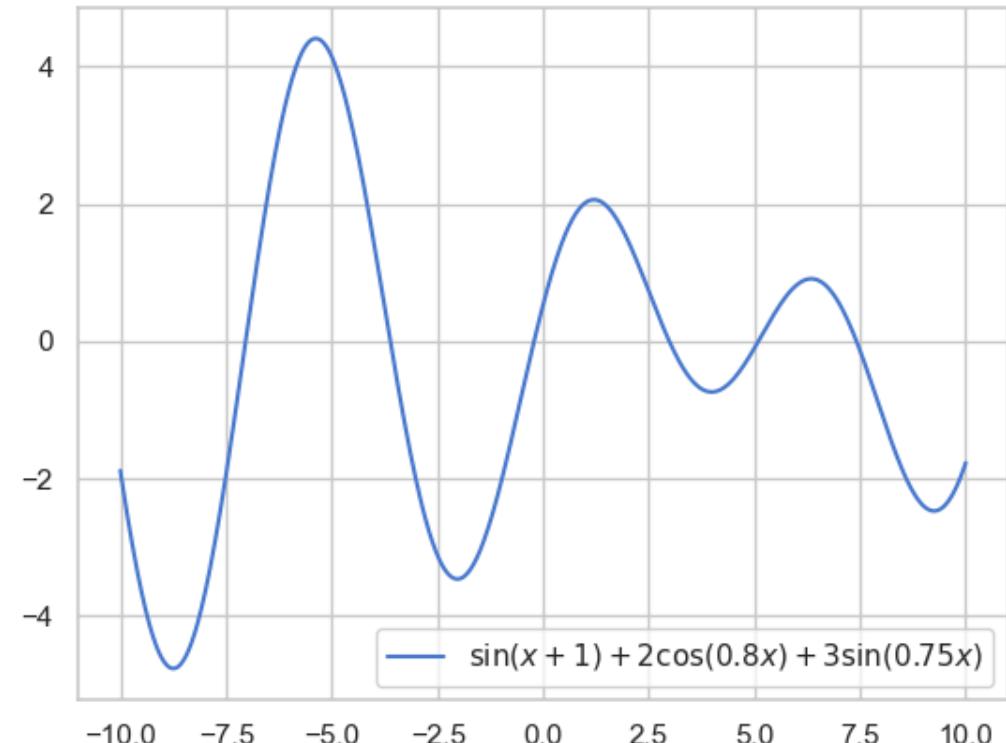


複雑さを生み出す方法

一方、重み付き和をとると

$$f(x) = 3f_1(x) - 2f_2(x) + f_3(x)$$

👉 そこそこ複雑



ぐにやつとした関数の表現のしかた

✓ 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

$$\mathbf{a} = (a_1, a_2, a_3, a_4, a_5),$$

$$\mathbf{b} = (b_1, b_2, b_3, b_4, b_5),$$

$$\mathbf{c} = (c_1, c_2, c_3, c_4, c_5)$$

をもつ

$$f(x; \mathbf{a}, \mathbf{b}, \mathbf{c}) = \sum_{i=1}^5 a_i \sin(b_i x + c_i)$$

を考える



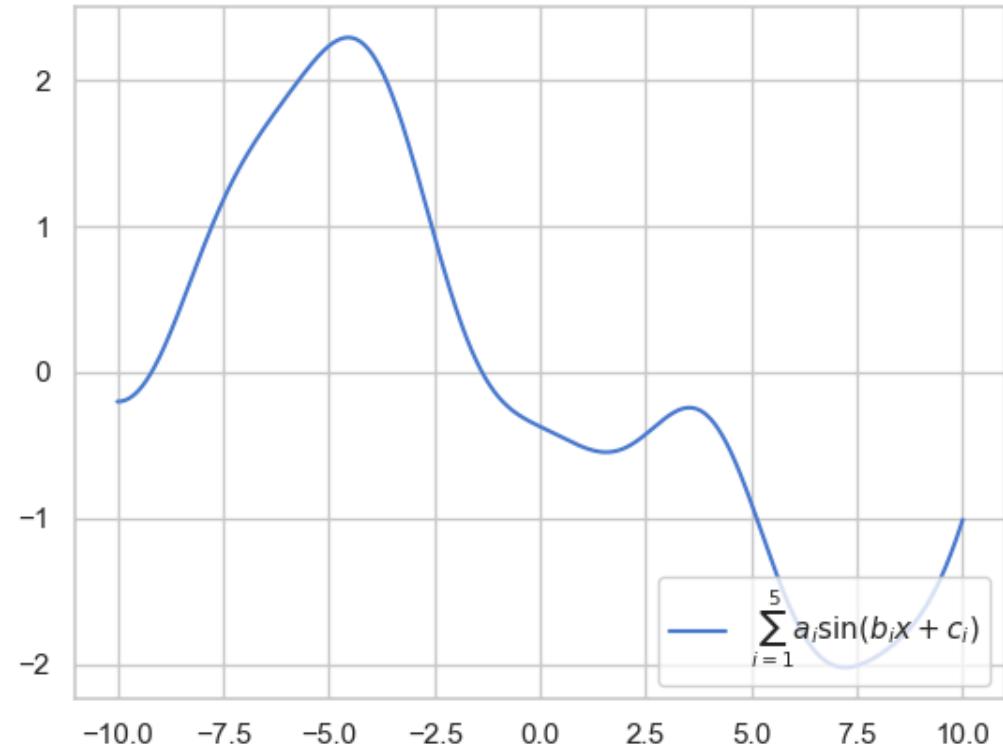
パラメータを変えることによつ て幅広い表現が得られる確認

$$\mathbf{a} = (0.83, 0.27, 0.84, 0.28, 0.14)^T$$

$$\mathbf{b} = (0.71, 0.47, 0.56, 0.39, 0.94)^T$$

$$\mathbf{c} = (0.08, 0.92, 0.16, 0.44, 0.21)^T$$

のとき



正確な値は $\mathbf{a} = [-6.4801085083, 6.1051318800, 7.9832160997, -8.3429947226, 5.3951834931]$, $\mathbf{b} = [0.0749141624, -0.0644060425, -0.2030537747, -0.2263950685, -0.0981644237]$, $\mathbf{c} = [-2.5574982288, 11.9469148236, -9.5401909056, -9.7479688677, 8.6363494625]$ です

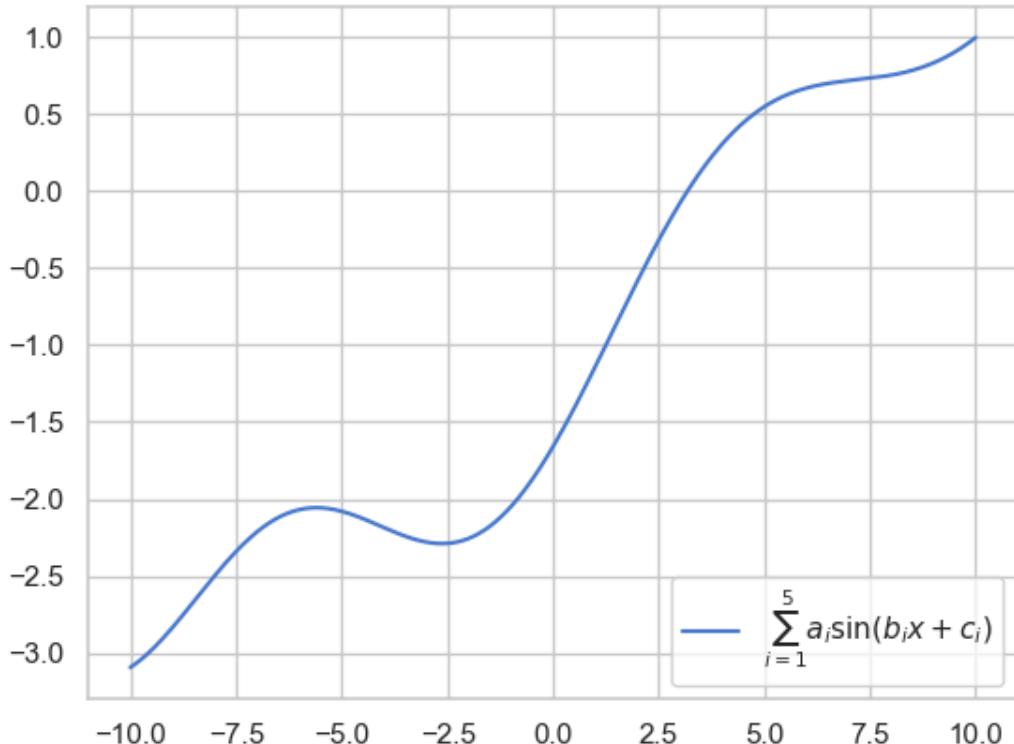
パラメータを変えることによつ て幅広い表現が得られる確認

$$\mathbf{a} = (0.39, -0.29, -0.67, -0.96, 0.92)^T$$

$$\mathbf{b} = (-0.35, 0.84, 0.22, -0.25, -0.04)^T$$

$$\mathbf{c} = (-0.61, -2.06, 3.97, 0.40, -3.85)^T$$

のとき



正確な値は $\mathbf{a} = [0.3902687779, -0.2931365774, -0.6719522358, -0.9616291983, 0.9248299981]$, $\mathbf{b} = [-0.3522200005, 0.8488925592, 0.2250941117, -0.2574283604, -0.0413569962]$, $\mathbf{c} = [-0.6140479386, -2.0554661015, 3.9653490740, 0.3968633932, -3.8504082926]$ です

「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、

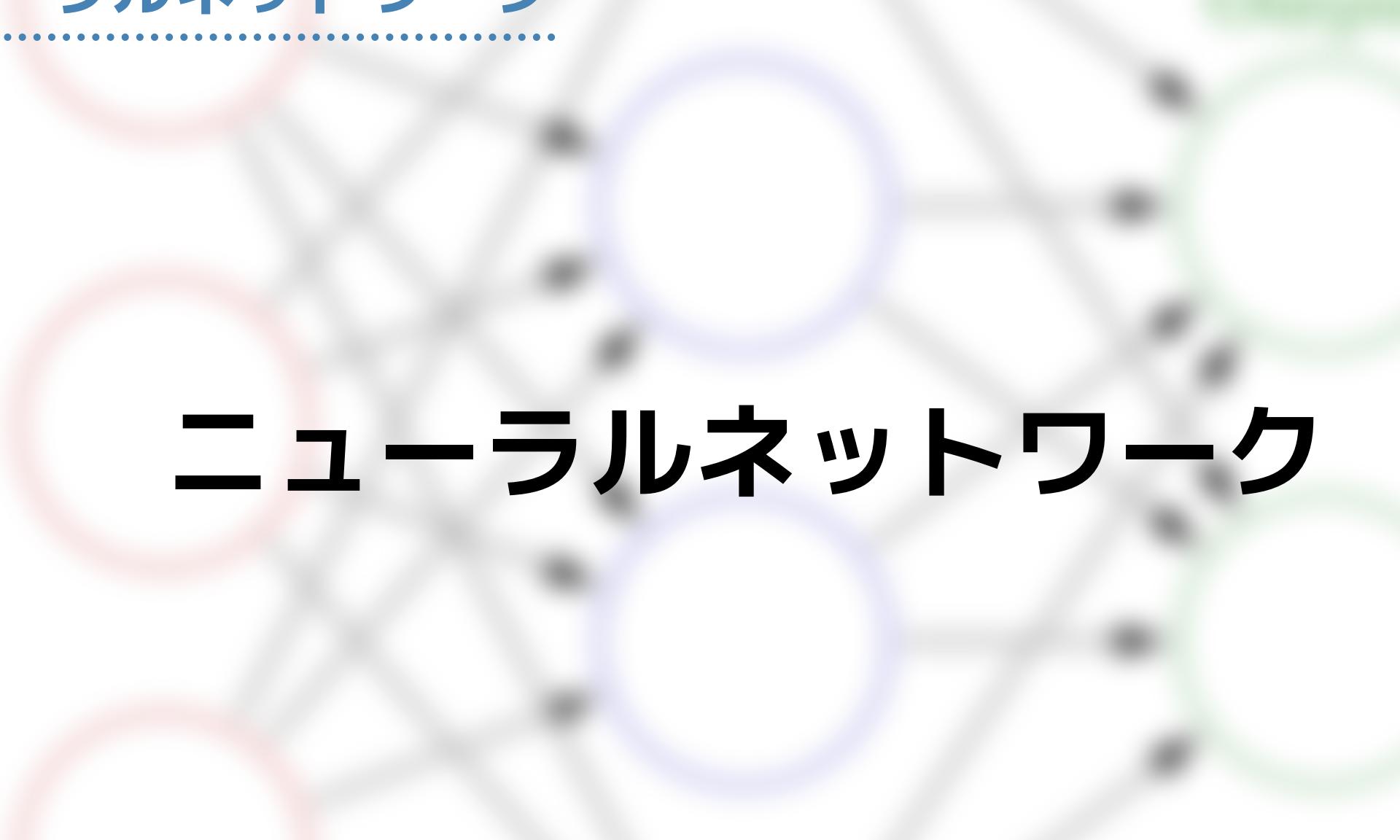
変化させるのが可能なところはパラメータにして、学習で求める」

「基になる関数」はどう選ぶべきか？

.....

「基になる関数」も
学習で求めよう

ニューラルネットワーク



ニューラルネットワーク

ニューラルネットワーク



[事実1]
最近流行りの機械学習モデル
はたいていニューラルネット
ワークをつかっている



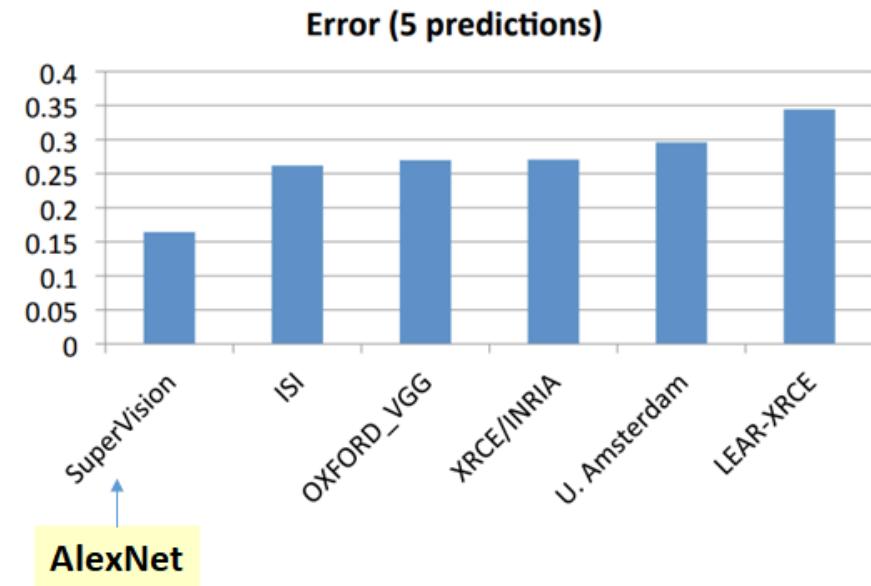
中央の画像は <https://diamond.jp/articles/-/241828> より. Ponanza と佐藤天彦名人の対局

ニューラルネットワーク

[事実2]

ある程度以上複雑なタスクでは、ニューラルネットワークが最も優れた性能を示すことが多い

Ranking of the best results from each team



画像は <https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b93598314160> から

今日の内容

.....

1. ニューラルネットワークの基本的な概念の整理
2. 全結合層の理解

基本的な概念

.....

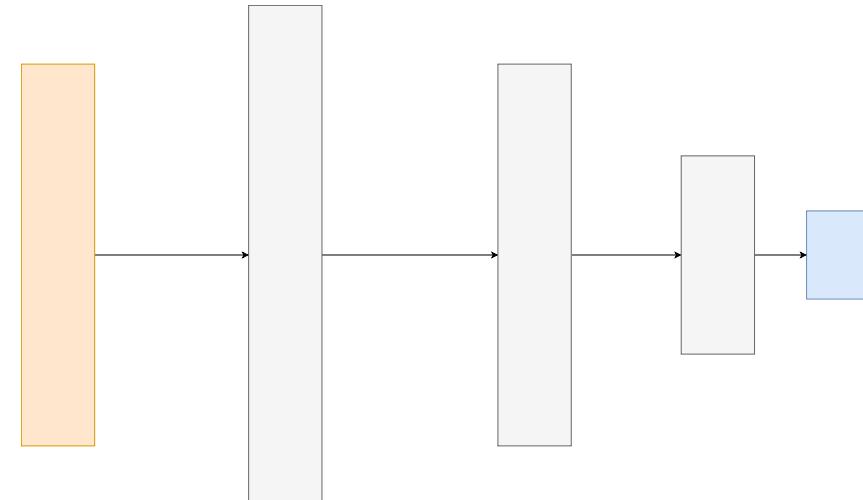
基本単位: レイヤー

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

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

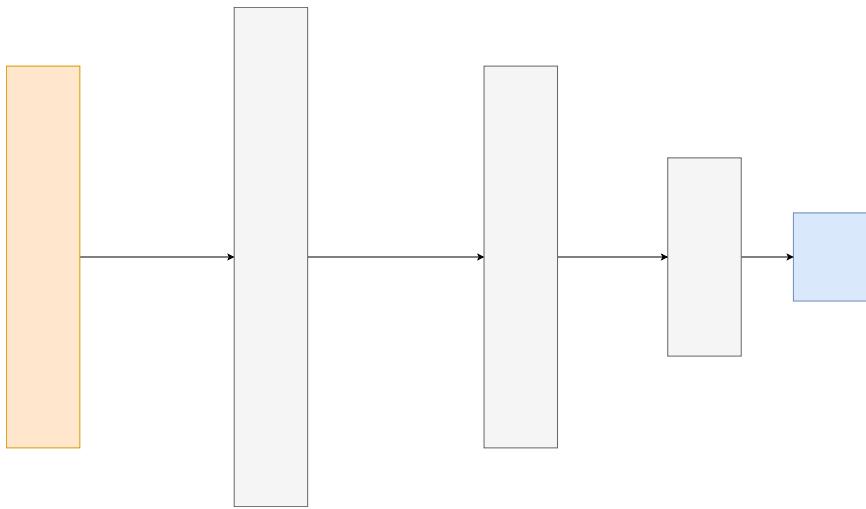


ニューラルネットワークの構造

- 入力層
入力を受け取る部分
- 出力層
出力を出力する部分
- 中間層(隠れ層, hidden layer)
それ以外



データの流れは、
 $x \rightarrow \text{入力層} \rightarrow \text{中間層} \dots \rightarrow \text{出力層} = y$



いろいろなレイヤーがある

PyTorch本体でデフォルトで定義されているものだけで 160 個以上? [1]

[1] `torch.nn.Module` のサブクラスの数を数えました。正確な数でないかもしれません。

全結合層 (Linear, Dense層)

もっとも普遍的・基本のレイヤー

全結合層 (Linear, Dense層)

パラメータ $W \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ と

各レイヤーが固有にもつ活性化関数 σ を用いて

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

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
2. 複雑な関数を表現するアイデア...

1. 合成
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

パラメータ $W \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ と

各レイヤーが固有にもつ活性化関数 σ を用いて

入力として $\mathbf{x} \in \mathbb{R}^n$ を受け取り、 $\sigma(W\mathbf{x} + \mathbf{b})$ 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...

1. 合成
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

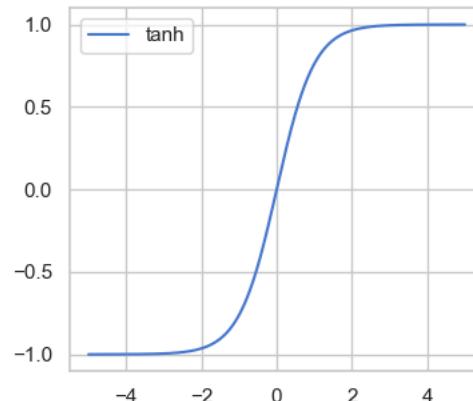
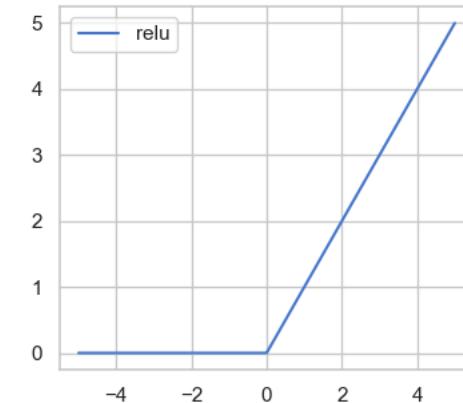
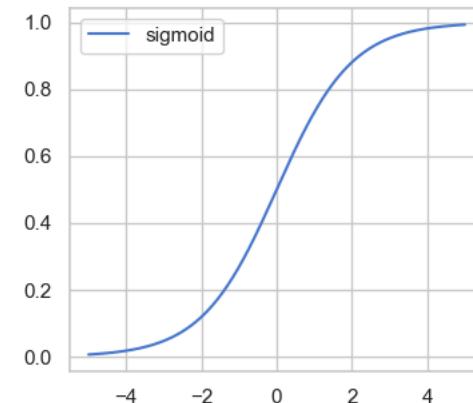
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

- ReLU関数

$$\text{ReLU}(x) = \max(0, x)$$

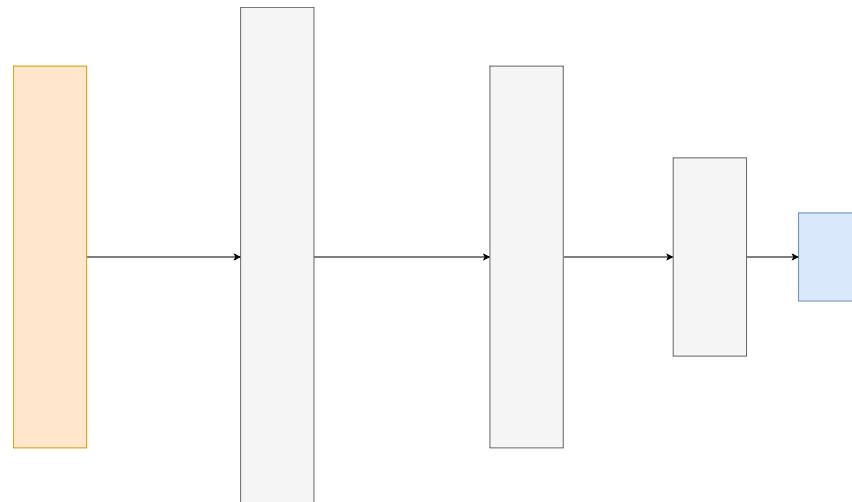
- tanh関数

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



アイデア1. 合成

合成を繰り返す \Leftrightarrow 複雑な関数を表現



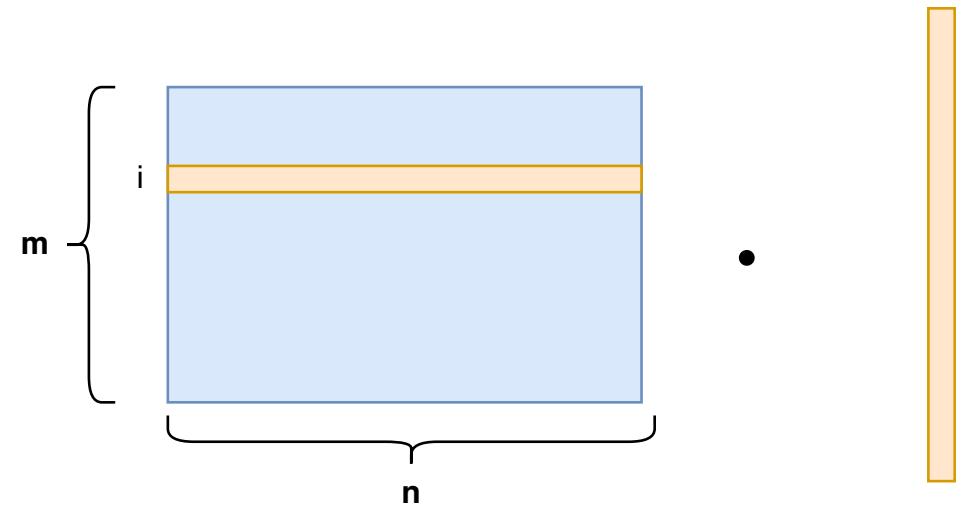
アイデア2. 和をとる

m 個の出力のひとつに注目してみる

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$



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

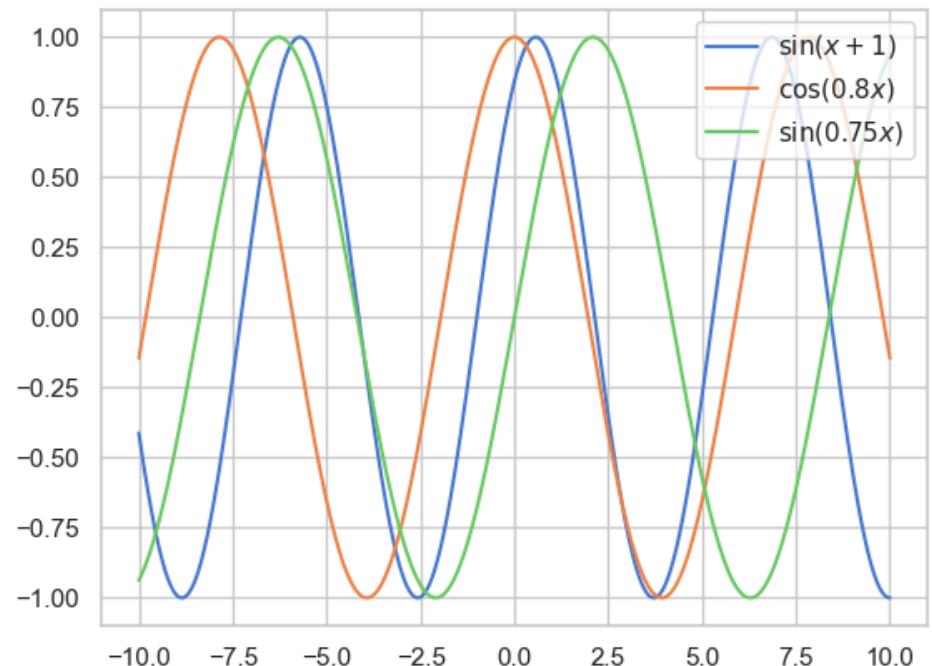


分解して考えると

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

は、

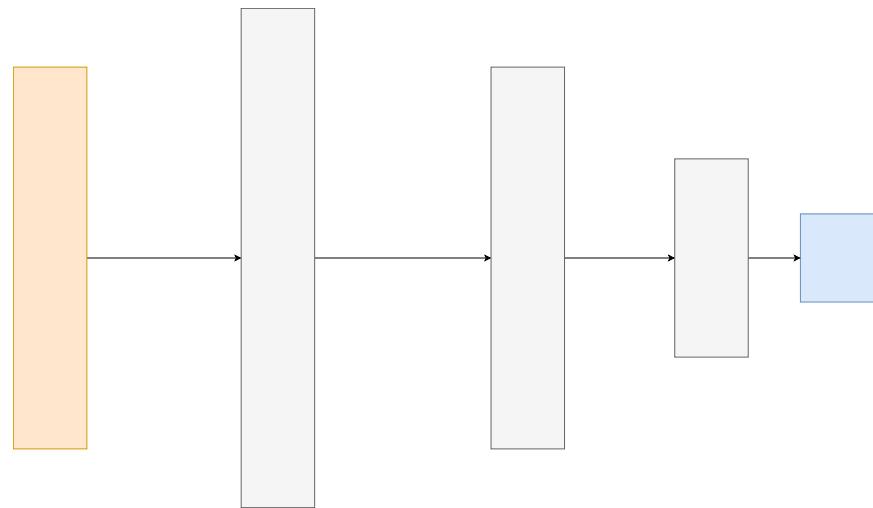
👉と同じことをしている



分解して考えると

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

この層の入力 x_j はそれまでの層で σ を通ってきたもの。



複雑な関数が生まれていた

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



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 $\rightarrow m_1, \rightarrow m_2, \rightarrow \dots, \rightarrow m_d$ 次元ベクトルへと変換されながら
計算が進んでいく)

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

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

...

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

出力層

.....

ここで、

$u^{(i)}$ 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

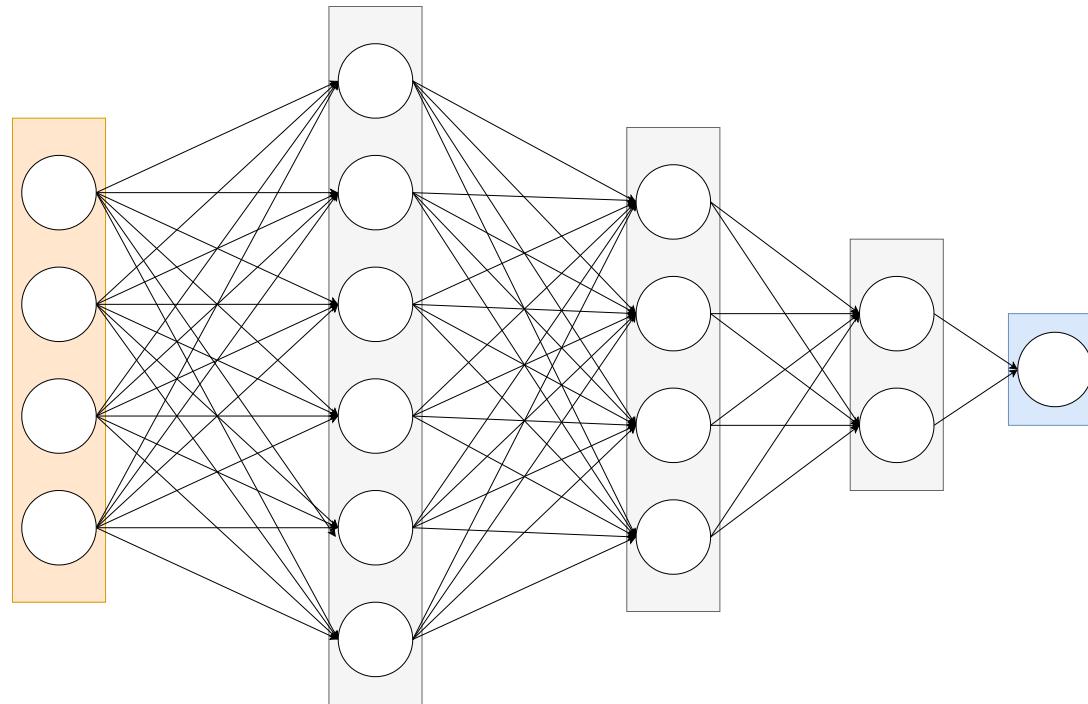
というわけで

「基になる関数」も
学習で求めよう

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ptron (Multi Layer Perceptron, MLP) という



そのほかの用語たち

用語	意味
MLP (Multi Layer Perceptron)	全結合層のみからなるニューラルネットワーク
DNN (Deep Neural Network)	複数の隠れ層を持つニューラルネットワーク
ANN (Artificial Neural Network)	人工ニューラルネットワーク。本来の意味のニューラルネットワーク(動物の神経回路)と区別するためこういう名前が使われることがある

ニューラルネットワークの性質

そもそもの動機:

直線だけしか表現できないのは困る。いろいろな関数が表現できるようになりたい。



どれくらいの範囲の関数が表現できるようになったのか？

ニューラルネットワークの万能近似性

.....

結論

直線 \Leftrightarrow なんでも ※

※ ざっくりとした表現です。

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 ※

※ ざっくりとした表現です。

今日のまとめ

- 我々の学習手法は、 $f(x) = ax + b$ というモデルの構造自体に直接依存しているわけではなかった
- そして、 $f(x) = ax + b$ というモデルの構造では直線しか表現することができないので、違う形を考えることにした
- 「基になる」簡単な関数の、合成と和を考えることでかなり複雑な関数も表現できることがわかった
- 「基になる」関数の選び方を考える上で、この関数自体もパラメータによって変化させるモデルとして、ニューラルネットワークを導入した
- ニューラルネットワークは非常に幅広い関数を表現できることがわかった

次回予告

.....

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) 説明

- ニューラルネットワークの表現能力は 1980年代後半～1990年代後半くらいまで盛んに研究されていた
- いろいろな条件でいろいろな結果を得ている
- ここではおそらく最も有名である Cybenko による定理 [1] を紹介する

[1] Cybenko, George. "Approximation by superpositions of a sigmoidal function." Mathematics of control, signals and systems 2.4 (1989): 303-314.

準備

定義1. シグモイド型関数

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

を満たす関数を「シグモイド型関数」と呼ぶ。

$I = [0, 1]^d$ として、 C を I 上の連続関数全体の集合とする。

定理 (Cybenko, 1989)

任意の $f \in C$, $\varepsilon > 0$ に対して、ある $g(x) = \sum_{i=1}^n a_i \sigma(b_i x + c_i)$ が存在して

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

主張

.....

平易に書くと、
どんな連続関数も隠れ層が一つのニューラルネットワークで十分に近似できる

ステップ1. シグモイド型関数をつかった階段関数のつくりかた

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

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

σ はシグモイド型関数 $\Rightarrow b_i$ をものすごく大きくするとどうなるか？

証明ステップ1

$$b_i = 99999999999999999999999999999999$$

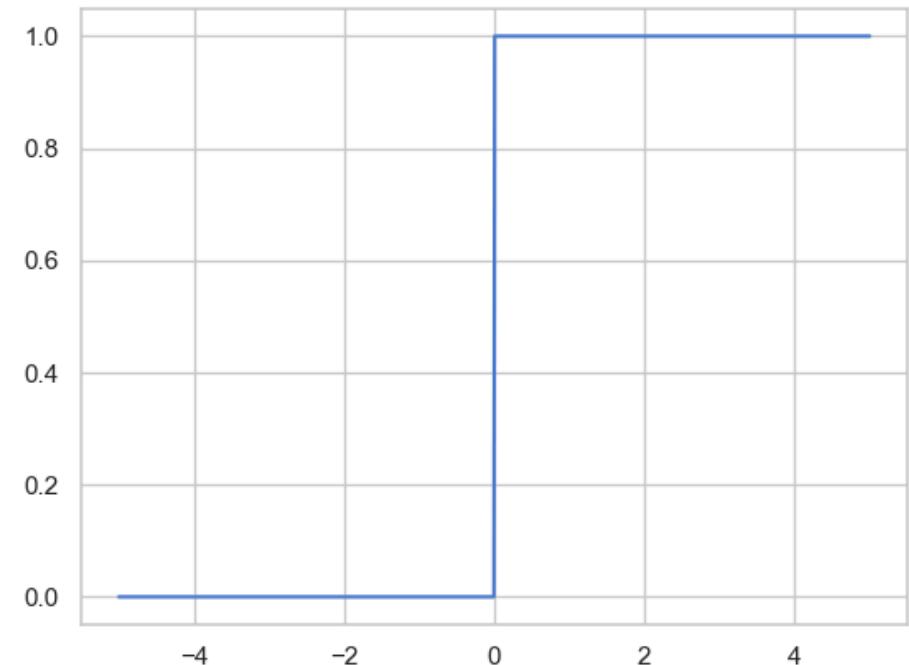
とする。

すると、 $x_i - \frac{c_i}{b_i}$ が少しでも正なら

$$\sigma(b_i x + c_i) = 1$$

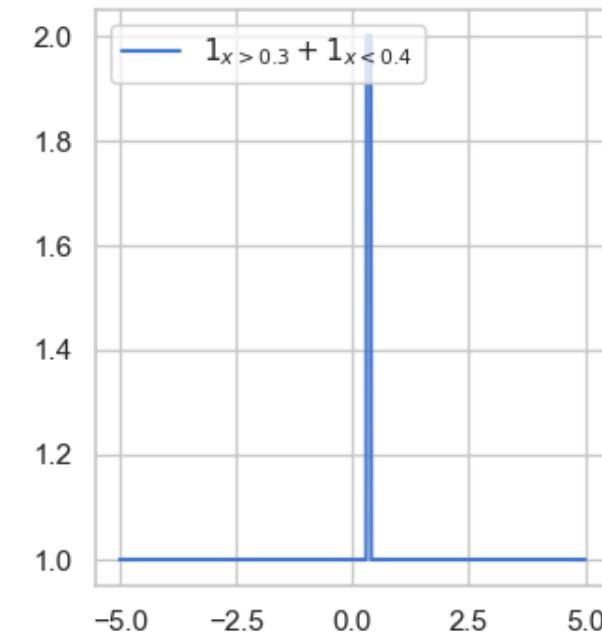
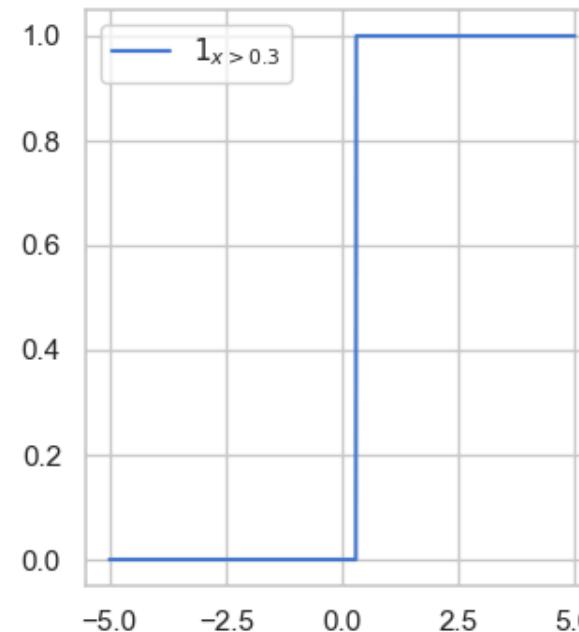
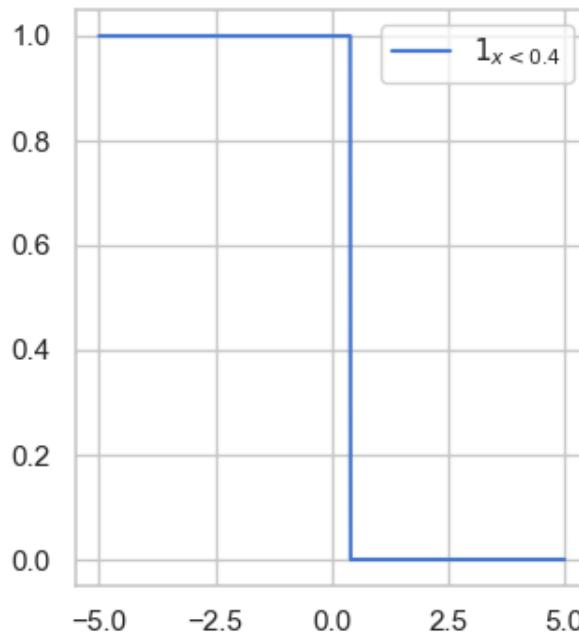
負なら

$$\sigma(b_i x + c_i) = 0.$$



証明ステップ2. 矩形関数の作り方

- ✓ すると、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！**

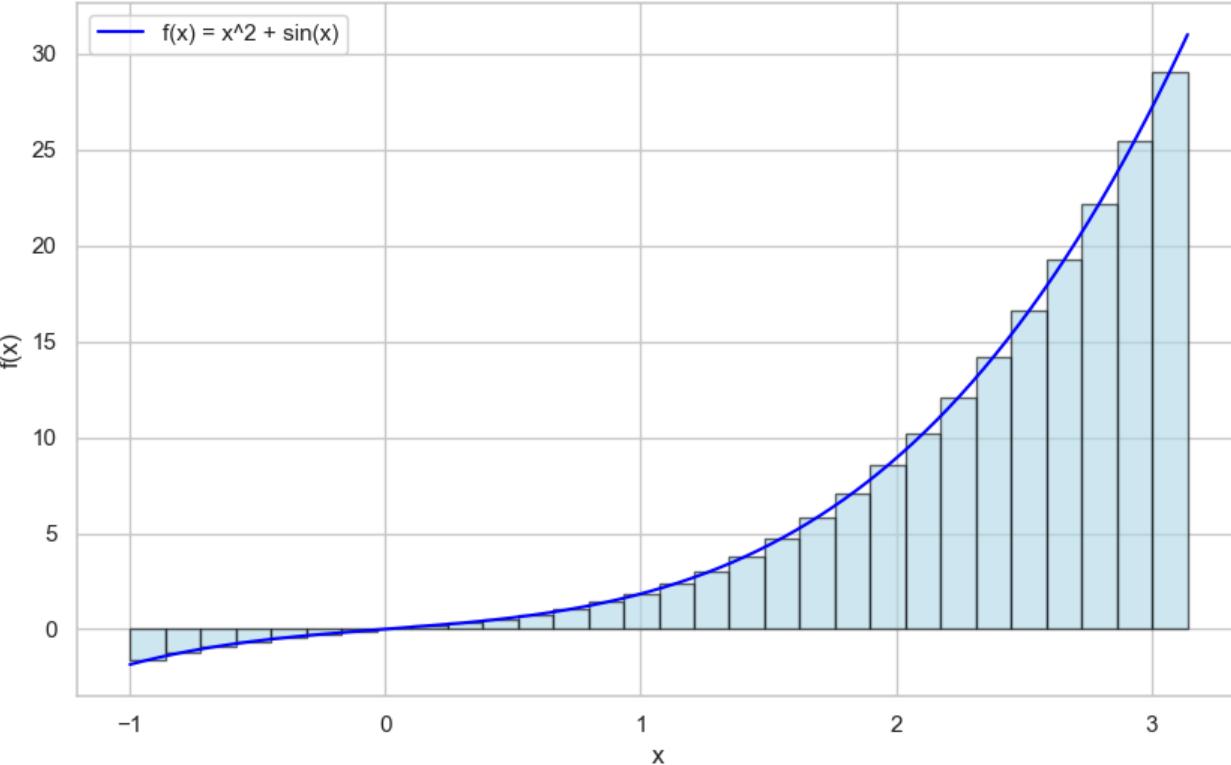


証明ステップ3. ウイニングラ ン

.....

✓ これさえできればもうOK

連続関数を全て**矩形関数の和**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モデルはニューラルネットワークだけ？

⇒ 全然ふつうに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、もうすこし議論を進めていく必要がある

「深さ」は必要？

この結果の主張:

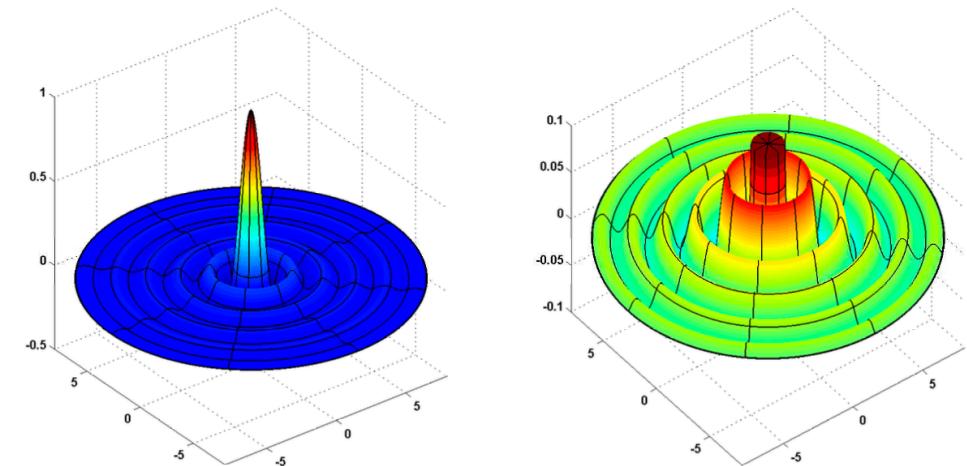
十分幅が広い「隠れ層」が一つあれば十分

世の中の主張:

たくさんの層があるNNがよく機能する

▽ なぜ？

A. 層を深くすると指数関数的に表現力が上がり、幅を広くすると多項式的に表現力が上がる。 [1]



[1]

Eldan, Ronen, and Ohad Shamir. "The power of depth for feedforward neural networks." Conference on learning theory. PMLR, 2016.

画像も同論文より