機械学習講習会 第二回

- 「勾配降下法」

traP アルゴリズム班 Kaggle部

2023/xx/xx

スケジュール

第1回: 学習
第2回: 勾配降下法
第3回: 自動微分とPyTorch
第4回: ニューラルネットワークの構造
第5回: ニューラルネットワークの学習と評価
第6回: Kerasを用いたニューラルネットワークの学習
第7回: ニューラルネットワーク発展

第二回: 学習

前回のまとめ

- アイスの売り上げを予測するには、気温から売り上げを予測する「関数」を構築するのが必要であった。
- 今回は関数の形として $f\left(x\right) = ax + b$ (一次関数)を仮定して、「関数」を求めることにした。
- この関数は、パラメータとしてa, bをもち、a, bを変えることで性質が変わるのがわかった。
- a, bを定めることで具体的に関数が定まる。
- このパラメータを決める基準として、「悪さ」の基準である損失関数を定めた。
- 損失関数の値を最小化するa, bを決めることを「学習」と呼ぶ。

損失関数の最小化

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

問題

次の関数$f(x)$の最小値を取るxを求めよ。

$$f(x) = x^2 + 4x + 6$$

問題

次の関数 $f(x)$の最小値を取るxを求めよ。

$$f(x) = x^2 + 4x + 6$$

解答

$f(x) = (x+2)^2 + 2$ より、 $x = -2$

一般の関数の最小化

- 平方完成で解けた

プログラムに起こすと...

```
def solve(a, b, c):
  return -b / (2 * a)
```

関数を与えられたら、簡単な数学の操作（紙と鉛筆だけ)で解けた！

→ 一般の関数ではどう？

プログラムのことばで書くと、二次関数の最小値を取るxを求める関数は
定数時間で動作するということ。

第二問

最小化してください。

$$f(x) = x^2 + e^{-x}$$

一般の関数の最小化

解:

$$f'(x) = 2x - e^{-x}$$

より最小値を与えるx_{min}の必要条件は

$$2x_{min} - e^{-x_{min}} = 0$$

:thinking_face:

一般の関数の最小化


Google
wolfram alpha
wolfram alpha
wolfram alpha 積分
wolfram alpha 微分方程式
wolfram alpha 微分

一般の関数の最小化


WolframAlpha 計算知能
2x = e^(-x)
自然言語
数学入力
拡張キーボード
例を見る
アップロード
ランダムな例を使う
Wolframの画期的なアルゴリズム，知識ベース，AIテクノロジーを使って，
専門家レベルの答を計算しましょう
数学›
科学・テクノロジー›
社会・文化›
日常生活›

一般の関数の最小化

実解

$$x = W\left(\frac{1}{2}\right)$$

解

出力の

一般の関数の最小化

？

ランベルトのW関数

出典: フリー百科事典『ウィキペディア（Wikipedia）』

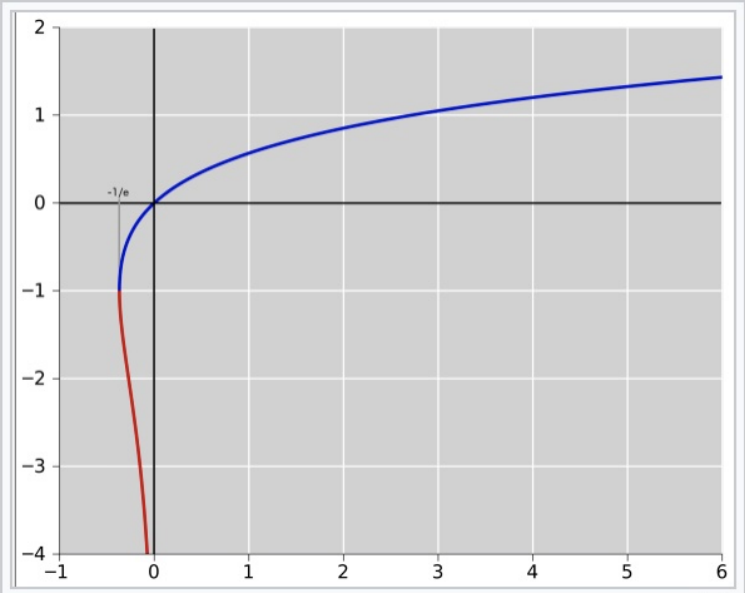
ランベルトのW函数（ランベルトのWかんすう、英: *Lambert W function*）あるいは**オメガ函数** (*ω function*)、対数積（*product logarithm*; 乗積対数）は、函数 $f(z) = ze^z$ の逆関係の分枝として得られる函数 W の総称である。ここで、e^z は指数函数、z は任意の複素数とする。すなわち、W は $z = f^{-1}(ze^z) = W(ze^z)$ を満たす。

上記の方程式で、$z' = ze^z$ と置きかえれば、任意の複素数 z' に対する W 函数（一般には W 関係）の定義方程式

$$z' = W(z')e^{W(z')}$$

を得る。

函数 f は単射ではないから、関係 W は（0 を除いて）多価である。仮に実数値の W に注意を制限するとすれば、複素変数 z は実変数 x に取り換えられ、関係の定義域は区間 $x \geq -1/e$ に限られ、また開区間 $(-1/e, 0)$ 上で二価の函数になる。さらに制約条件として $W \geq -1$ を追加すれば一価函数 $W_0(x)$ が定義されて、$W_0(0) = 0$ および $W_0(-1/e) = -1$ を得る。それと同時に、下側の枝は $W \leq -1$ であって、$W_{-1}(x)$ と書かれる。これは $W_{-1}(-1/e) = -1$ から $W_{-1}(-0) = -\infty$ まで単調減少する。



$W(x)$ のグラフの $W > -4$ および $x < 6$ の部分。$W \geq -1$ なる上の枝を主枝 W_0 と言い、$W \leq -1$ なる下側の分枝を W_{-1} という。

ランベルト W 関係は初等函数では表すことができない[1]。ランベルト W は組合せ論において有用で、例えば木の数え上げに用いられる。指数函数を含む様々な方程式（例えばプランク分布、ボーズ–アインシュタイン分布、フェルミ–ディラック分布などの最大値）を解くのに用いられ、また $y'(t) = qy(t-1)$ のような遅延微分方程式（英語版）の解としても生じる。生化学において、また特に酵素動力学において、ミカエリ

一般の関数の最小化

いいたかったこと
→ このレベルの単純な関数でも、
最小値を与える式を構成するのはむずかしい

そもそもの目的

損失

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

を最小化したかった。

効いてくる条件

1. 厳密に最小値を得る必要があるか？
2. fはどんな関数か？

⬇

1. 厳密に最小値を得る必要はない
2. fは非常に複雑になりうる

そもそも

1. 厳密に最小値を得る必要はない

数学の答案で最小値 1 になるところを 1.001と答えたら当然 ❌

機械学習では、誤差 1% と 1.001%は事実上同じ

2. fは非常に複雑になりうる

第一回では $f(x) = ax + b$の形を考えたが...

(特にニューラルネットワーク以降) は複雑になる

$$L(W_1, W_2, W_3, \boldsymbol{b_1}, \boldsymbol{b_2}, \boldsymbol{b_3}) = \frac{1}{n} \sum (\boldsymbol{y} - W_3 \sigma(W_2 \sigma(W_1 \boldsymbol{x} + \boldsymbol{b_1}) + \boldsymbol{b_2}) + \boldsymbol{b_3}))^2$$

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

(というか、普段我々が使っている数学の記号では書けなくなる)

✅ 非常に広い範囲の関数に対して

そこそこ小さい値を返してくれる方法

勾配降下法

微分係数の定義

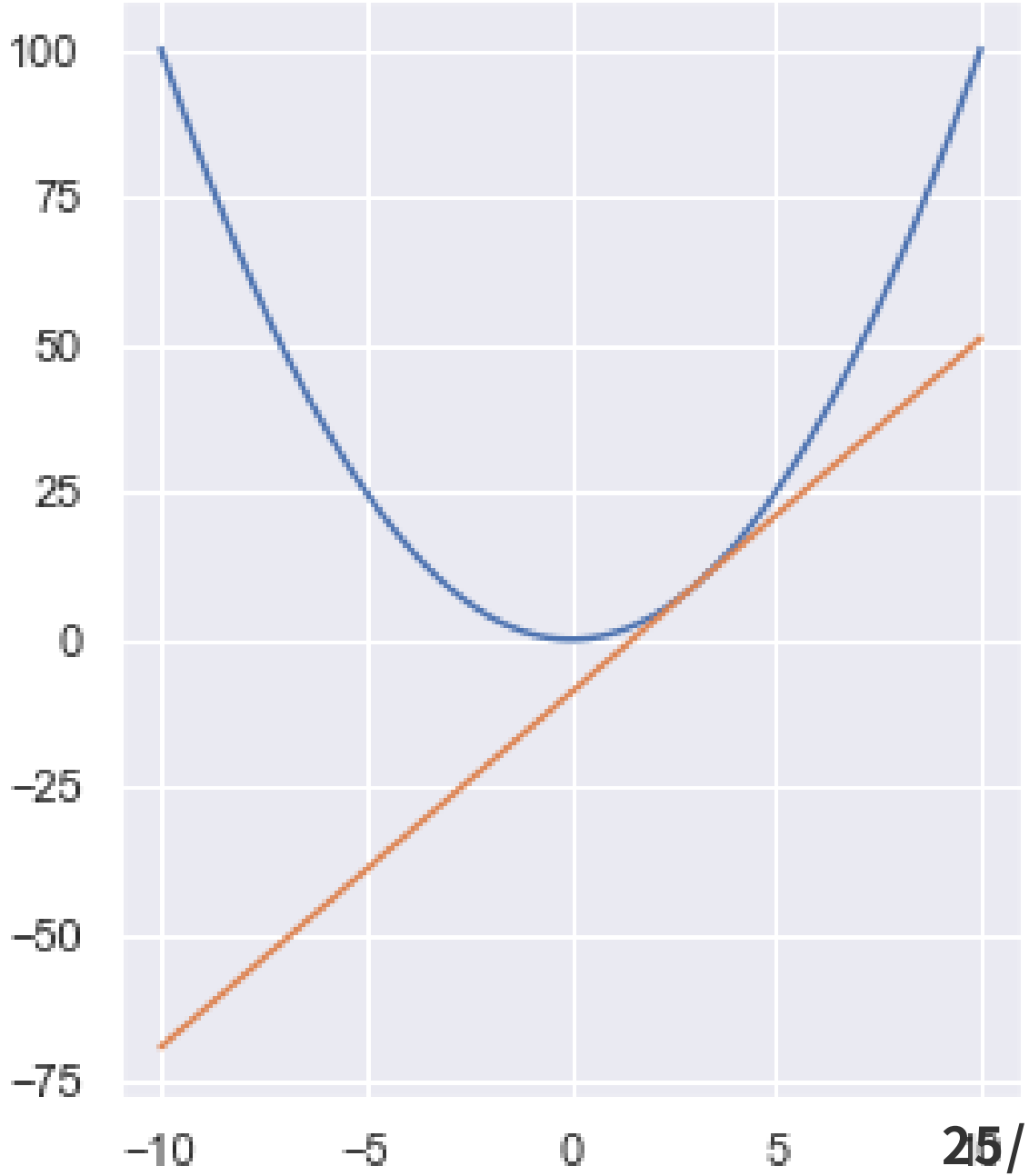
$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

微分係数

- 微分係数 $f'(x)$ は、
  xにおける接線の傾き

  ⬇

  $-f'(x)$**方向に関数を少し動かすと、関数の値はすこし小さくなる**



「傾き」で値を更新してみる

例) $f(x) = x^2$

$x = 3$ で $f(3) = 9,\ f'(3) = 6$

$\therefore -f'(x)$ は負の方向

↓

すこし負の方向にxを動かしてみる

$f(2.9) = 8.41 < 9$

✅　小さくなった

「傾き」で値を更新してみる

例) $f(x) = x^2$

$x = 2.9$ で $f(3) = 8.41,\ f'(2.9) = 5.8$

$\therefore -f'(x)$ は負の方向

↓

すこし負の方向にxを動かしてみる

$f(2.7) = 7.29 < 8.41$

✅　小さくなった

「傾き」で値を更新してみる

これを繰り返すことで小さい値まで到達できそう！

- ちゃんと定式化します

勾配降下法

関数 $f(x)$ と、初期値 x_0 が与えられたとき、
次の式で x を更新する

$$x_{n+1} = x_n - \eta f'(x_n)$$

(η は**学習率**と呼ばれる定数)

正確にはこれは最急降下法と呼ばれるアルゴリズムで、「勾配降下法」は勾配を使った最適化手法の総称として用いられることが多いと思います。ですがここでは「勾配降下法」という手法をきっちりと把握して欲しいのであえてこう呼びます。(そこまで目くじらを立てる人はいないと思いますし、勾配降下法あるいは勾配法と言われたらたいていの人がこれを思い浮かべると思います。)

勾配降下法

マイナーチェンジが大量にある...
(実際に使われるやつは第五回で予定)

$$x_{n+1} = x_n - \eta f'(x_n)$$

抑えておきたいこと

1. 値が $-f'(x)$ の方向に更新される
2. 更新幅が微分係数に比例 [1]
3. 学習率によって更新幅を制御する

---

[1] 大抵はそうなのですが、固定幅で収束が早いという主張の手法もあったりして (https://arxiv.org/abs/2302.06675)　一概には言えないのですが、大体この通りであることは確かです。

勾配降下法のお気持ち

1. 値が $-f'(x)$ の方向に更新される

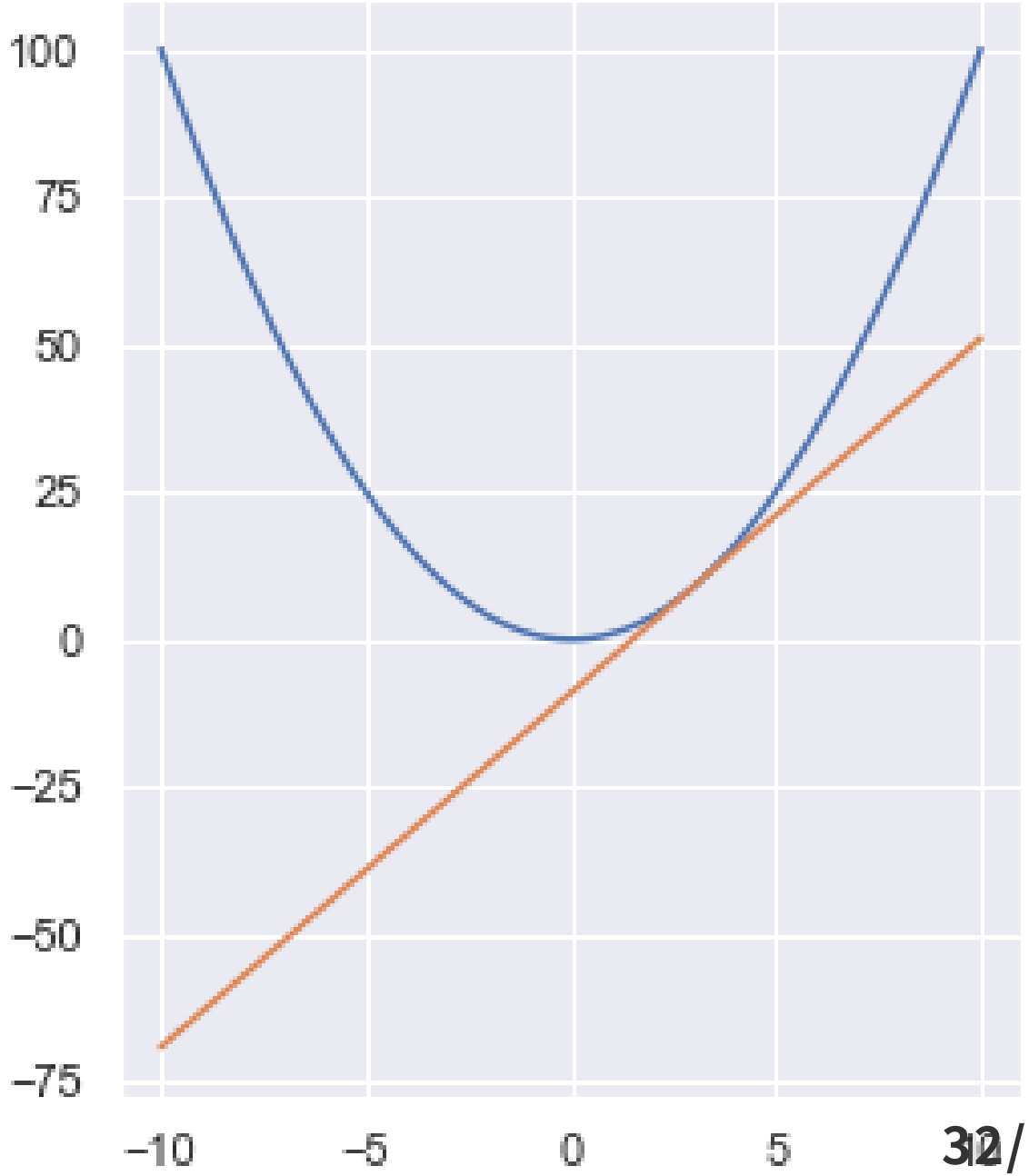
さっきの説明の通りです。

勾配降下法のお気持ち

2. 更新幅が微分係数に比例

最小値から遠い $\leftrightarrow$ 勾配が大きいため

(実際に使われる手法ではこの傾向は相当弱く入っている印象)



勾配降下法のお気持ち

3. 学習率によって更新幅を制御する

✅ 微分係数はあくまで「その点の情報」

傾向が成り立つのはその周辺だけ

⬇

ちょっとずつ更新していく必要がある

⬇

小さな値 **学習率** η をかけることで少しずつ更新する

勾配降下法のココがすごい！

✅ その式を(解析的に)解いた結果が何であるか知らなくても、
導関数さえ求められれば解を探しにいける！

収束性や局所最適解の話をしたい人がいると思いますが、グッと堪えてしばしお待ちください...

実際にやってみる

$$f(x) = x^2$$

初期値として、$x_0 = 3$
学習率として、$\eta = 0.1$ を設定。(この二つは自分で決める！)

$$x_1 = x_0 - \eta f'(x_0) = 3 - 0.1 \times 6 = 2.4$$
$$x_2 = x_1 - \eta f'(x_1) = 2.4 - 0.1 \times 4.8 = 1.92$$
$$x_3 = x_2 - \eta f'(x_2) = 1.92 - 0.1 \times 3.84 = 1.536$$
$$\dots$$
$$x_{100} = 0.00000000006111107929$$

✅ **最小値を与える $x = 0$ に非常に近い値が得られた！**

第二問

最小化してください。

$$f(x) = x^2 + e^{-x}$$

実際にやってみる2

$$f'(x) = 2x - e^{-x}$$

初期値として $x = 3$, 学習率として $\eta = 0.0005$ を設定。

$x_1 = 2.997024893534184$

$\dots$

$x_{10000} = 0.3517383210080008$

ヨシ！

Pythonによる実装

$x_{n+1} = x_n - \eta f'(x_n)$ をコードに起こす

```
from math import exp

x = 3
#  (注意: $\eta$は、学習率(learning rate)の略である lr としています。)
lr = 0.0005

# xでの微分係数
def grad(x):
    return 2 * x - exp(-x)

for i in range(10001):
    # 更新式
    x = x - lr * grad(x)
    if i % 1000 == 0:
        print('x_', i, '=', x)
```

Pythonによる実装

```
x_ 0 = 2.997024893534184
x_ 1000 = 1.1617489280037716
x_ 2000 = 0.576046627929 5902
x_ 3000 = 0.4109554481889124
x_ 4000 = 0.36713277266602845
x_ 5000 = 0.3557211225 4324284
x_ 6000 = 0.3527650719 6575846
x_ 7000 = 0.352000400939733
x_ 8000 = 0.3518026668897593
x_ 9000 = 0.3517515401706734
x_ 10000 = 0.3517383210080008
```

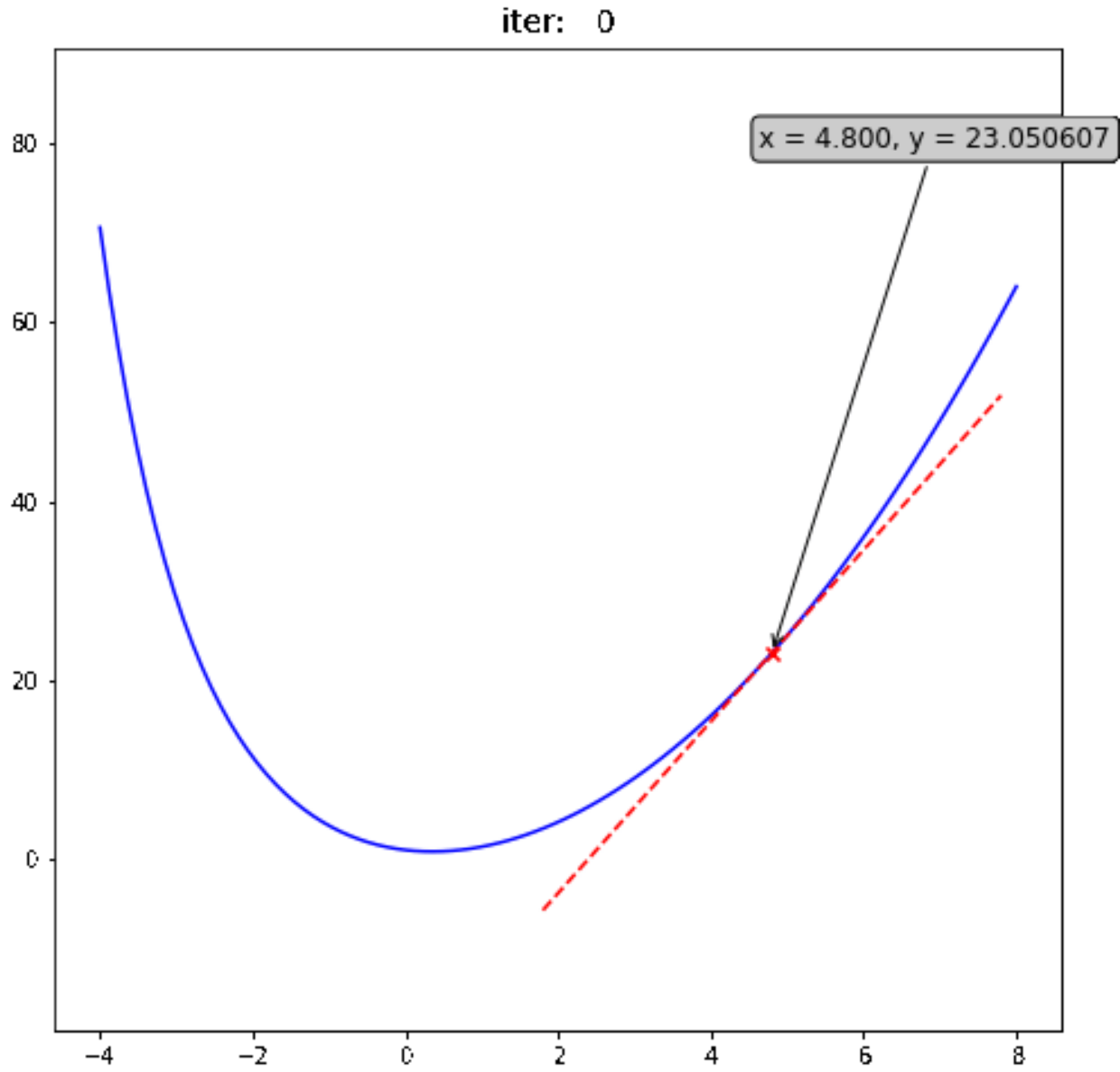
Pythonによる実装

配布ソースコード:

`gradient_decent.ipynb`

では勾配降下法の様子を可視化できる `Logger` というクラスを用意

最適化の様子をgifで見ることができます(内容は後述)

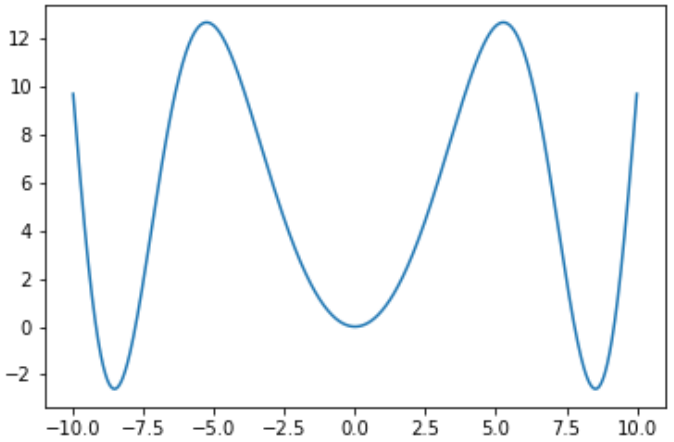


常に上手くいく？　ー凸関数ー

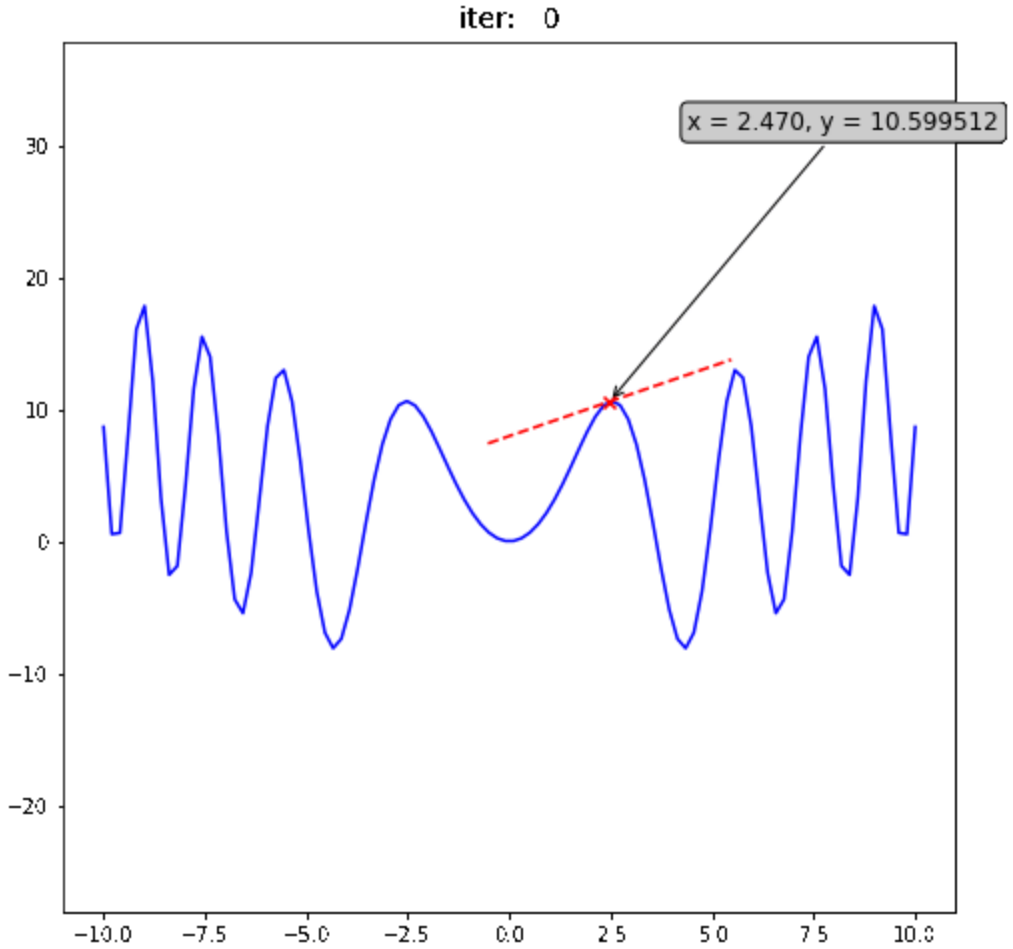
ここまで紹介した関数は、実はすべて勾配降下法が非常にうまくいく関数(凸関数と呼ばれる関数)

✅ 勾配降下法があまりうまくいかない関数もある

例) $f(x) = \dfrac{x^2}{10} + 10\sin\left(\dfrac{x^2}{4}\right)$

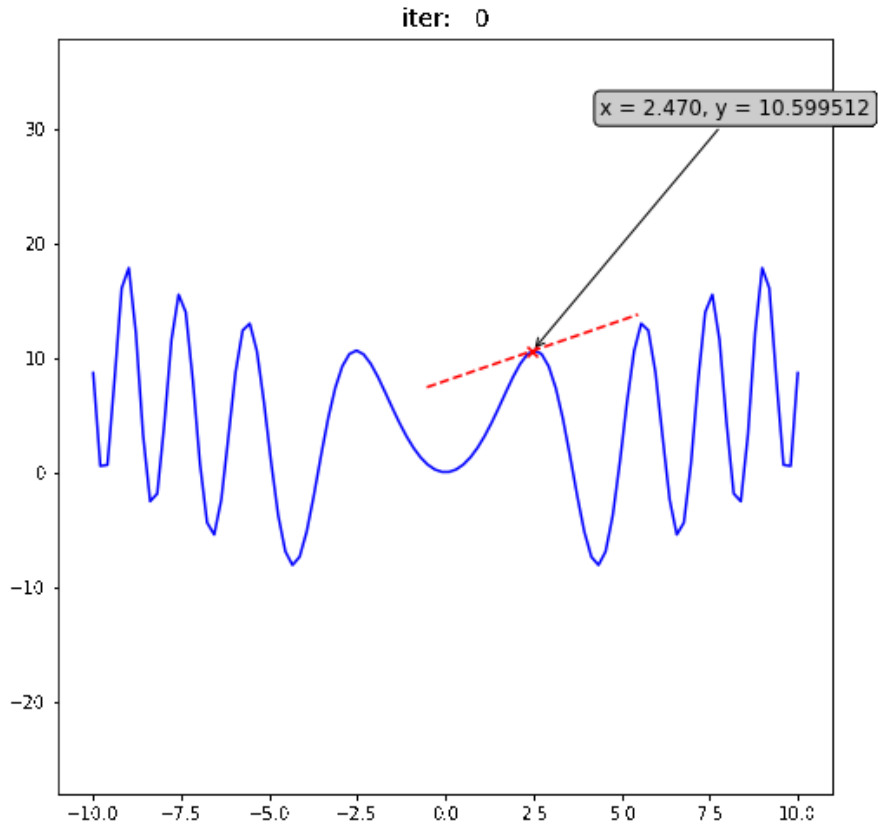


常に上手くいく？　ー凸関数ー



常に上手くいく？　ー凸関数ー

局所最適解 … 付近では最小値だが、全体の最小値ではない
大域最適解 … 全体で最小値



常に上手くいく？　ー凸関数ー

単純な勾配降下法では、局所最適解に陥ってしまう
⇨ なるべく局所最適解にならないよう色々と工夫(詳しくは第5回)

- 最急降下法

$$x_{n+1} = x_n - \eta f'(x_n)$$

- Momentum

$$v_{n+1} = \alpha v_n - \eta f'(x_n)$$

$$x_{n+1} = x_n + v_{n+1}$$

工夫して「陥りにくく」なってるだけで陥りはします。

多変数関数への応用

多変数関数の場合は、微分係数→勾配ベクトル　に置き換えればOK

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \eta \nabla f(\boldsymbol{x}_n)$$

一年生はちょうど微分積分学第一でやるころ？かと思ったので大きくは扱いませんでしたが、
一変数の場合できちんと理解できていれば問題はないはずです。

第三問

最小化してください。

$$\frac{-\log(\frac{1}{1+e^{-x}}+1)}{(x^2+1)}$$

第三回　自動微分

勾配降下法はうまくいくか？ 2

その他のアルゴリズム

(ここ以降では深層学習の学習の話をします)

さまざまな関数を最小化するアルゴリズムは勾配降下法だけではない(焼きなまし法など)

⬇

これらでは学習はうまくいかないのか？

結論: あまりうまくいかない(とされている)

その他のアルゴリズム

そもそも... 深層学習モデルのような膨大なパラメータを持つモデルの損失関数をうまく小さくすることは、
現代の理論では非常に難しいと思われている

⬇

「え、でも世の中いろんなモデルがうまく行ってますよね。。。」

⬇

実はなぜうまく行っているのか誰もわかっていない

その他のアルゴリズム

- なぜ確率的勾配降下法(第五回でやります)は他の手法と違いうまく収束するのか？
- なぜ膨大なパラメータを持つモデルの学習がうまくいくのか？

↓

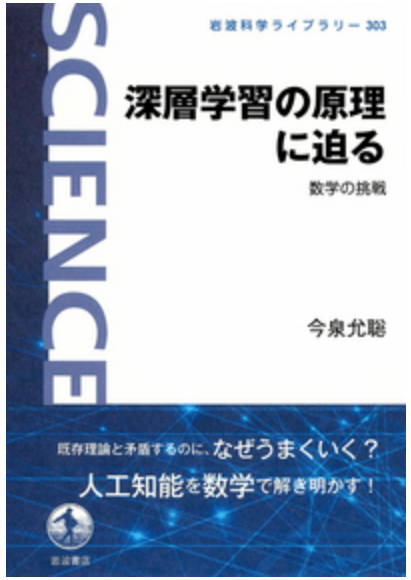
- 確率的勾配降下法の勾配の変動の確率分布は損失関数が十分小さい点に集中する？
- 膨大なパラメータを持つモデルはかえって学習しやすくなる？

誰もわかってない😄

興味がある人向け

【基調講演】『深層学習の原理の理解に向けた理論の試み』 今泉 允聡（東大)
https://www.slideshare.net/MLSE/ss-237278350

深層学習の原理に迫る
https://www.iwanami.co.jp/book/b570597.html



演習

1. 勾配降下法を使った線形回帰(一変数ver)

以下のようなデータがあります。

(来店者数) = $a\times$ (広告費)

という関係があると仮定したとき、aを勾配降下法によって求めて、広告費から来店者数を予測するモデルを学習させてください。

(データは次のページ)

広告費	売り上げ
$10	$12
$2	$4
$5	$8
$10	$12
$10	$11
$5	$4

pythonにコピペする用:

```
x = [10, 2, 5, 10, 10, 5]
y = [12, 4, 8, 12, 11, 4]
```

解答

損失関数をaについて微分します。

$$\frac{\partial}{\partial a}\sum_{i=1}^{n}(y_i - ax_i)^2 = -2\sum_{i=1}^{n}x_i(y_i - ax_i)$$

あとはPythonで実装すればいいです。

```
# 損失関数
def loss(a):
    n = len(x)
    s = 0
    for i in range(n):
        s += (y[i] - a * x[i])**2

    return s / n
```

```
# 損失関数の勾配
def dloss(a):
    s = 0
    for i in range(n):
        s += -2 * x[i] * y[i] + 2 * a * x[i]**2

    return s / n


# 勾配降下法
x = -4
lr = 0.001
for i in range(100):
    x -= lr * g(x)
    print(x, '|', loss(x))
```

次へ

ちなみに、配布したソースコードの `gradient_decent` 関数を使うと学習の様子が見れます。

```
logger = gradient_decent(loss, dloss, 50, -4, lr=0.001, x_range=np.linspace(-10, 10, 100))
logger.gif("linreg.gif", fps=10)
preview("linreg.gif")
```