Anpassungsübung

Als nächstes versuchen wir, die Funktion `y(x) = ax + b` anzupassen.

```
import numpy as np
import math
```

x = np.linspace(-math.pi, math.pi, 20)

```
print(x)
```

```
[-3.14159265 -2.81089869 -2.48020473 -2.14951076 -1.8188168  -1.48812284
 -1.15742887 -0.82673491 -0.49604095 -0.16534698  0.16534698  0.49604095
  0.82673491  1.15742887  1.48812284  1.8188168   2.14951076  2.48020473
  2.81089869  3.14159265]
```

Beachte, dass es `linspace` heißt, nicht `linespace`. Dies ist ein Teil des Codes aus einem Beispiel in den `PyTorch`-Tutorials. Diese Dezimalzahlen sind möglicherweise nicht sehr intuitiv.

```
x = np.linspace(0, 100, 20)
```

```
import numpy as np
import math

x = np.linspace(0, 100, 20)
y = np.linspace(0, 100, 20)

print(x)
print(y)
```

So erhalten wir zwei Datensätze. Wie können wir diese grafisch darstellen?

Allerdings sind `x` und `y` tatsächlich gleich.

```
x = np.random.rand(2)
print(x)
```

```
[0.06094295 0.89674607]
```

Weiter geht’s mit den Änderungen.

```
x = np.random.rand(2)*100
print(x)
```

```
[39.6136151  66.15534011]
```

Weiter bearbeiten.

```
import numpy as np
import math
import matplotlib.pyplot as plt

x = np.random.rand(10)*100
y = np.random.rand(10)*100

plt.plot(x,y)
plt.show()
```

```
[20.1240488  59.69327146 58.05432614  3.14092909 82.86411091 43.23010476
 88.09796699 94.42222486 58.45253048 51.98479507]
[58.7129098   1.6457994  49.34115933 71.13738592 53.09736099 15.4485691
 45.12200319 20.46080549 67.48555147 91.10864978]
```

Es ist sichtbar, dass der Pfad von `(20.1,58.7)` nach `(59.7,1.6)` und dann nach `(58,49.3)` verläuft. Obwohl das Diagramm chaotisch aussieht, gibt es dennoch eine Regelmäßigkeit. Es handelt sich um einen durchgehenden Strich.

```
import numpy as np
import math
import matplotlib.pyplot as plt

x = np.random.rand(2)*100
y = np.random.rand(2)*100

print(x)
print(y)
```

```
plt.plot(x,y)
plt.show()
```

Beachte, dass die Skalen von `x` und `y` sich ständig ändern. Daher sind zwei scheinbar identische Geraden tatsächlich unterschiedlich. Wie können wir also `a` und `b` in der Gleichung `y(x) = ax + b` bestimmen? Angenommen, wir kennen zwei Punkte auf dieser Geraden. Beachte, dass wir dies einfach auf einem Schmierblatt lösen können. Subtrahiere die beiden Gleichungen, eliminiere `b` und bestimme `a`. Setze dann `a` in eine der Gleichungen ein, um `b` zu berechnen.

Kann man jedoch eine Vermutungsmethode verwenden? Mit der binären Suche. Lass es uns ausprobieren.

```
import numpy as np
import math
import matplotlib.pyplot as plt

x = np.random.rand(2)*100
y = np.random.rand(2)*100

a_max = 1000
a_min = -1000
b_max = 1000
b_min = -1000

def cal_d(da, b):
    y0 = x[0] * da + b
    y1 = x[1] * da + b
    d = abs(y0 - y[0]) + abs(y1 - y[1])
    return d

def cal_db(a, db):
    y0 = x[0] * a + db
    y1 = x[1] * a + db
    d = abs(y0 - y[0]) + abs(y1 - y[1])
    return d
```

Hinweis: Der Code scheint eine Funktion zu definieren, die den absoluten Unterschied zwischen den berechneten Werten `y0` und `y1` und den gegebenen Werten `y[0]` und `y[1]` berechnet. Es

ist jedoch unklar, woher die Variablen `x` und `y` stammen, da sie nicht als Parameter übergeben werden.

```
def avg_a():
    return (a_max + a_min) / 2

def avg_b():
    return (b_max + b_min) / 2

for i in range(100):
    a = avg_a()
    b = avg_b()
    max_d = cal_d(a_max, b)
    min_d = cal_d(a_min, b)
    if max_d < min_d:
        a_min = a
    else:
        a_max = a

    a = avg_a()
    max_db = cal_db(a, b_max)
    min_db = cal_db(a, b_min)
    if max_db < min_db:
        b_min = b
    else:
        b_max = b

print(x)
print(y)
print('a = ', avg_a())
print('b = ', avg_b())
print(avg_a() * x[0] + avg_b())
print(avg_a() * x[1] + avg_b())
```

Führe es aus.

```
[42.78912791 98.69284173]
```

```
[68.95535212 80.89946202]
a =  11.71875
b =  -953.125
-451.68990725289063
203.4317390671779
```

Die Ergebnisse wichen jedoch stark voneinander ab.

Lassen Sie uns das Problem vereinfachen. Angenommen, `y(x) = ax`. Gegeben sei eine Menge von `x`, `y`. Wir sollen `a` bestimmen. Obwohl wir es direkt berechnen könnten, lassen Sie uns raten.

```
import numpy as np
import math
import matplotlib.pyplot as plt
from numpy.random import rand, randint

x = randint(100)
y = randint(100)

a_max = 1000
a_min = -1000

def cal_d(da):
    y0 = x * da
    return abs(y0 - y)

def avg_a():
    return (a_max + a_min) / 2

for i in range(1000):
    a = avg_a()
    max_d = cal_d(a_max)
    min_d = cal_d(a_min)
    if max_d < min_d:
        a_min = a
    else:
        a_max = a
```

```
print(x)
print(y)
print(avg_a())
print(avg_a()*x)
```

□□□□□□□□□□□□□□□□□□

```
96
61
0.6354166666666667
61.00000000000001
```

Allerdings schreibt man normalerweise `for i in range(15):`, was bedeutet, dass 15 Iterationen ziemlich genau sind. Warum? Beachten Sie, dass unsere `x` und `y` beide im Bereich von 0 bis 100 liegen. Daher liegt auch der Wert von `a` im Bereich von 0 bis 100. Zum Beispiel `x=1, y=99` und `x=99, y=1`. Daher können die Anfangswerte von `a_min` und `a_max` optimiert werden. Beachten Sie, dass `1/99` gleich `0.01` ist. Daher reicht eine Genauigkeit von bis zu `0.01` aus, was ungefähr 2^n entspricht, das etwa 10000 ist. log2(10000)=13.28. Das bedeutet, dass eine Einstellung von etwa 14 ausreichend ist.