Raport (SNE) Python

Wojciech Kubiak

15 lipca 2020

1 Metoda gradientu

Zaimplementuj poniżej podany Algorytm Metody Gradientu. Za pomocą tego algorytmu zbadaj lokalne i globalne minimum następujących funkcji.

1.1 Implementacja

```
# Python
# Gradient descent
import random

# Function return true if summed difference between x_new and x_old is grater t
# else return false
def check_epsilon(x_new, x_old, epsilon):
    sum = 0.0
    for i in range(len(x_new)):
        sum += abs(x_new[i] - x_old[i])
    if sum < epsilon:
        return False
    return True

# Function to calculate value from first input function
def first_function(x_1, x_2):
    return 2*x_1**2 + x_2**2 - 2 * x_1 * x_2 - 2 * x_1 + 1

# Function to calculate first gradient
def calculate_first_gradient(x_old, epsilon, c):
    print("First gradient")
    print(f"Values: [{x_old[0], x_old[1]}]")
    # function we are calculating gradient for
    print("f(x_1, x_2) = 2x_1^2 + x_2^2 - 2x_1 * x_2 - 2x_1 + 1")
    x_new = list(x_old)  # copy x_old to x_new
    flag = True
    while(flag):
        # assign x_old elements to variables for brevity
        x = x_old[0]
        y = x_old[1]
        x_new[0] = x - c * (4 * x - 2 * y - 2) # derivative by x1
        x_new[1] = y - c * (2 * y - 2 * x) # derivative by x2
        flag = check_epsilon(x_new, x_old, epsilon) # check diference
        x_old = list(x_new)  # copy x_new to x_old
    print(f"Point({x_new[0]}, {x_new[1]})")
    print(f"Value: {first_function(x_new[0], x_new[1])}\n")
```

```
# Function to calculate value from second input function
def second_function(x_1, x_2):
    # x ** y => x to the power of y
    return x_1**4/2 - x_1**3/3 - x_1**2/2 + x_2**2 - 2*x_2 + 1

# Function to calculate second gradient
def calculate_second_gradient(x_old, epsilon, c):
    print("Second gradient")
    print(f"Values: [{x_old[0], x_old[1]}]")
    print("f(x_1, x_2) = x_1^4/2 - x_1^3/3 - x_1^2/2 + x_2^2 - 2x_2 + 1\n")
    flag = True
    while(flag):
        # assign x_old elements to variables for brevity
        x = x_old[0]
        y = x_old[1]
        x_new = list(x_old)  # copy x_old to x_new
        x_new[0] = x - c * (2 * x ** 3 - x ** 2 - x) # derivative by x1
        x_new[1] = y - c * (2 * y - 4) # derivative by x2
        flag = check_epsilon(x_new, x_old, epsilon) # derivative by x2
        x_old = list(x_new)  # copy x_new to x_old
    print(f"Point({x_new[0]}, {x_new[1]})")
    print(f"Value: {second_function(x_new[0], x_new[1])}\n")


def main():
    print("Gradient descent\n")
    # Predefine epsilon and c constant
    epsilon = 0.00001
    c = 0.01
    # Generate list of length 2 with random floats between -5 <= x < 5
    input_first = [round(random.uniform(-5, 5), 2) for i in range(2)]
    calculate_first_gradient(input_first, epsilon, c)
    # Generate list of length 2 with random floats between -3 <= x < 3
    input_second = [round(random.uniform(-2, 2), 2) for i in range(2)]
    calculate_second_gradient(input_second, epsilon, c)


if __name__ == "__main__":
    main()
```

1.2 Wynik uruchomienia

```
Wojciechs-Mac-Pro:Python wojciechkubiak$ python3 gradient.py
Gradient descent

First gradient
Values: [(-0.25, 2.87)]
f(x_1, x_2) = 2x_1^2 + x_2^2 - 2x_1 * x_2 - 2x_1 + 1
Point(1.0004937371983043, 1.0007988835683663)
Value: 3.368907282030875e-07

Second gradient
Values: [(-0.18, -1.54)]
```

```
f(x_1, x_2) = x_1^4/2 - x_1^3/3 - x_1^2/2 + x_2^2 - 2x_2 + 1

Point(-0.4994828268443053, 1.9999030466941128)
Value: 0.94772296987149

Wojciechs-Mac-Pro:Python wojciechkubiak$ python3 gradient.py
Gradient descent

First gradient
Values: [(1.7, -3.8)]
f(x_1, x_2) = 2x_1^2 + x_2^2 - 2x_1 * x_2 - 2x_1 + 1
Point(0.9995058630131678, 0.9992004695602072)
Value: 3.3743652272377744e-07

Second gradient
Values: [(0.56, -1.6)]
f(x_1, x_2) = x_1^4/2 - x_1^3/3 - x_1^2/2 + x_2^2 - 2x_2 + 1

Point(0.9999981084405191, 1.9995135906874189)
Value: 0.665694084640891
```

1.3 Analiza działania programu

Losujemy punkt K(x1, x2) i sprawdzamy pochodna w punkcie K aby określić nachylenie funkcji. Następnie dopasowujemy parametry w taki sposób aby przesunąć je w stronę minimum lokalnego. Od parametru odejmujemy stałą c (tempo uczenia) pomnożoną przez wartość pochodnej w danym punkcie z parametru. Działanie programu powtarzamy dopóki różnica między nowymi a starymi parametrami nie będzie mniejsza od epsilona. Tępo uczenia nie może być za duże ponieważ możemy pominąć minumum lokalne przy kolejnej iteracji.

2 Maszyna Boltzmanna (MB)

Zaimplementuj poniżej podany Algorytm MB. Zbadać należność zachowania MB od stałej tempetatury $T > 0$ (Zob. dwie Uwagi zaraz przed i po Twierdzeniem 6.1.1, Notatki 6). Lepiej byłoby, wyniki przestawione $0 \to$ _ i $0 \to *$.

2.1 Implementacja

```
# Python
import random

euler = 2.718281828459


def main():
    temperatures = [0.01, 0.1, 1, 3, 10]

    z_list = [1 if x < 10 else 0 for x in range(20)]

    vec_c = get_c_vector(z_list)
    vec_w = get_w_vector(vec_c)
    theta = get_theta(vec_c)

    for temperature in temperatures:
```

```
        print("Current temperature: {}".format(temperature))
        x = [[] for _ in range(11)]
        x[0] = gen_random_vector()
        t = 0

        while(t < 10):

            beta = gen_beta_vector()
            for i in range(20):

                if(beta[i] >= 0 and beta[i] <= f_uit(vec_w[i], x[t], theta[i],
                    x[t+1].append(1)
                elif(beta[i] <= 1 and beta[i] >= f_uit(vec_w[i], x[t], theta[i]
                    x[t+1].append(0)

            t += 1
            print_vector(x[t])


def f_uit(w, x, theta, temperature):

    s = 0
    uit = 0
    for j in range(20):
        s += w[j]*x[j]
        uit = s - theta

    return 1 / (1 + euler ** (-(uit/temperature)))


def get_theta(vec):
    ''' Return theta value for vector'''
    sum = [0]*20
    for i in range(20):
        for j in range(20):
            sum[i] += vec[i][j]
    return sum


def get_w_vector(vec):
    ''' Compute vector "w" from vector "c"'''
    w = [[] for _ in range(len(vec))]
    for i in range(20):
        for j in range(20):
            w[i].append(2*vec[i][j])
    return w


def get_c_vector(vec):
    '''Generate "c" vector from "z" list '''
    c = [[] for _ in range(len(vec))]
    for i in range(len(vec)):
```

```
        for j in range(len(vec)):
            if i != j:
                c[i].append((vec[i] - 0.5) * (vec[j] - 0.5))
            else:
                c[i].append(0.0)
    return c


def gen_random_vector():
    '''Generate random list, each element is 0 or 1.'''
    return [random.randint(0, 1) for x in range(20)]


def gen_beta_vector():
    '''Generate random list, each element is range from 0 to 1.'''
    return [random.random() for x in range(20)]


def print_vector(vec):
    for i in range(len(vec)):
        if (vec[i] <= 0.0):
            print(" _ ", end='')
        else:
            print(" * ", end='')
    print(" ")


if __name__ == "__main__":
    main()
```

2.2 Wynik uruchomienia

```
Wojciechs-Mac-Pro:Python wojciechkubiak$ python3 boltzman.py
Current temperature: 0.01
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
Current temperature: 0.1
 * _ _ _ * * * * _ _ * _ * _ * _ _ _ _ *
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
 * * * * * * * * * * _ _ _ _ _ _ _ _ _ _
```

```
 *  *  *  *  *  *  *  *  *  *  _  _  _  _  _  _  _  _  _  _
Current temperature: 1
 _  _  _  *  _  _  _  _  _  _  *  *  *  *  *  *  *  _  *  *
 _  *  _  _  _  _  _  _  _  _  *  *  *  *  *  *  *  *  *  *
 _  _  _  _  _  _  _  _  _  _  *  *  *  *  *  *  *  *  *  *
 _  _  _  _  _  _  _  _  _  _  *  *  *  *  *  *  *  *  *  *
 _  _  _  _  _  _  _  _  _  _  *  *  *  *  *  *  *  *  *  *
 _  _  _  _  _  _  _  _  _  _  *  *  *  *  *  *  *  *  *  *
 _  _  _  _  *  _  _  _  _  _  *  *  *  *  *  *  *  *  *  *
 _  _  _  _  _  _  _  _  _  _  *  *  *  *  *  *  *  *  *  *
 _  _  _  _  _  _  _  _  _  _  *  *  *  *  *  *  *  *  *  *
 _  _  _  _  _  _  _  _  _  _  *  *  *  _  *  *  *  *  *  *
Current temperature: 3
 *  *  _  *  *  *  _  _  _  _  *  _  *  *  *  *  *  *  *  *
 _  _  _  _  _  _  *  *  _  _  _  *  *  *  _  *  _  *  _  *
 *  _  _  _  _  *  _  _  _  _  *  _  _  _  *  _  *  *  *  _
 _  _  _  *  *  _  _  _  _  *  _  *  *  *  *  _  *  _  *  *
 _  *  _  _  *  _  _  _  _  _  _  _  _  *  _  *  _  *  _  *
 _  *  _  _  *  _  *  _  *  _  _  *  _  *  *  _  *  _  *  *
 _  *  *  _  _  _  *  *  _  _  _  _  *  *  _  _  _  _  _  *
 *  _  *  _  _  *  *  *  *  _  *  _  _  _  *  *  _  *  _  _
 _  *  *  _  *  *  _  _  _  *  _  _  *  *  *  _  _  *  _  _
 *  *  _  *  _  _  *  *  *  *  _  *  *  _  _  _  _  *  _  _
Current temperature: 10
 _  *  _  *  *  _  *  *  *  _  _  _  *  *  *  _  _  *  *  _
 *  _  *  _  _  *  *  *  _  _  _  *  *  *  *  *  _  _  _  _
 *  _  _  *  _  *  _  *  _  _  _  _  *  *  _  _  *  *  *  *
 *  *  _  _  _  *  *  *  *  *  _  _  *  _  _  _  _  *  *  _
 *  *  _  *  _  *  *  _  *  _  *  _  _  _  *  _  _  _  *  *
 *  *  *  _  _  _  _  _  *  _  _  _  *  _  _  *  *  _  _  *
 _  _  *  *  *  *  *  _  *  _  _  _  *  _  *  _  _  *  *  _
 *  _  *  _  *  *  *  _  *  *  _  *  _  *  _  *  *  *  _  _
 *  _  _  _  *  *  _  _  _  *  _  *  *  _  *  _  *  _  _  _
 *  _  _  *  _  *  _  *  *  _  _  _  _  _  *  *  _  _  _  *
Wojciechs-Mac-Pro:Python wojciechkubiak$
```

2.3 Analiza działania programu

Stany Maszyny Boltzmana obliczamy za pomocą funkcji

$$x_i(t+1) = \begin{cases} 1, & \text{gdy } 0 \leqslant \beta_i \leqslant f(u_i(t)) \\ 0, & \text{gdy } f(u_i(t)) \leqslant \beta_i \leqslant 1 \end{cases}$$

gdzie $\beta_i \in [0,1]$, funkcja $f(u_i(t))$ wygląda następująco

$$f(u_i) = \frac{1}{1 + e^{-\frac{u_i(t)}{T}}}$$

W naszym przypadku $u_i(t)$ obliczamy z

$$u_i(t) = \left\{ \sum_{j=1}^{20} w_{ij} x_j(t) \right\} - \theta_i$$

dla wartość 0.25 dla $x_j = 0$ oraz -0.25 dla $x_j = 1$. Dla temperatur bliskich zero otrzymamy

$$lim_{T \to 0} f(0.25) = lim_{T \to 0} \frac{1}{1 + e^{-\frac{0.25}{T}}} = lim_{x \to -\infty} \frac{1}{1 + e^{x}} = lim_{x \to 0} \frac{1}{1 + x} = 1$$

$$lim_{T\to 0}f(-0.25) = lim_{T\to 0}\frac{1}{1+e^{\frac{0.25}{T}}} = lim_{x\to+\infty}\frac{1}{1+e^x} = lim_{x\to+\infty}\frac{1}{1+x} = 0$$

Dla takich temperatur stan maszyny będzie zmieniał się na przeciwny. Stan każdego neuronu x_i zmieni się w następujący sposób:

$$x_i(t) = 0 \Longrightarrow x_i(t+1) = 1$$

Wartości funkcji $f(u_i)$ zbliżają się do $\frac{1}{2}$ kiedy temperatura rośnie.

$$lim_{T\to+\infty}f(\pm 0.25) = lim_{T\to+\infty}\frac{1}{1+e^{\frac{\pm 0.25}{T}}} = lim_{x\to\pm 0}\frac{1}{1+e^x} = lim_{x\to 1}\frac{1}{1+x} = \frac{1}{2}$$

Kolejne stany maszyny stają się bardziej losowe, gdyż prawdopodobieństwo $P(f(u_i) \leqslant \beta) \to \frac{1}{2}$ i $P(\beta \leqslant f(u_i)) \to \frac{1}{2}$.