نام و نام خانوادگی دانشجو: رحمت اله انصاری

شماره دانشجویی: ۹۹۱۲۳۷۷۳۳۱

Date: 1402-03-26

تمرین کار با شبکه عصبی در متلب

6- تابع زیر را درنظر بگیرید.

$$f(x,y) = (x-3.14)^2 + (y-2.72)^2 + \sin(3x+1.41) + \sin(4y-1.73)$$

این یک تابع محدب نیست و بنابراین یافتن حداقل آن دشوار است زیرا حداقل محلی یافت شده لزوما حداقل جهانی نیست.

7- برنامه ای بنویسید که مقدار بهینه تابع را به روش PSO بدست آورد. مقدار پیشنهادی برای پارامترها به صورت زیر است:

$$c1 = c2 = 0.1, w = 0.8$$

البته میتوانید مقدار پارامترها را در طول جستجو به گونه ای تغییر دهید که در مراحل اولیه، قدرت اکتشاف و در مراحل نهایی، قدرت استخراج بیشتر باشد.
کد را خودتان و بدون استفاده از کتابخانه های موجود بنویسید.
در صورت انجام مرحله اول، در صورت تمایل میتوانید مقدار بهینه تابع فوق را با استفاده از امکانات موجود در متلب یا پایتون بدست آورید و حاصل را مقایسه کنید.

https://www.mathworks.com/help/gads/particleswarm.html

https://pypi.org/project/pyswarms/#:~:text=PySwarms%20is%20an%20extensible%20research,implementing%20PSO%20in%20their%20problems.

ابتدای کد اول کدهای زیر را داریم:

```
import numpy as np
from pyswarm import pso

# Objective function
def objective_function(x):
    return (x[0] - 3.14) ** 2 + (x[1] - 2.72) ** 2 + np.sin(3 * x[0] + 1.41) +
np.sin(4 * x[1] - 1.73)
```

در اینجا میتوانیم ببینیم که نامپای را برای انجام عملیات‌های ریاضی و پایسوارم را برای پیاده سازی تابع pso فراخوانی کردیم. سپس تابع بهینه را درست کرده ایم.

```
# Set parameters
num_particles = 50
num_dimensions = 2
num_iterations = 100
c1 = 0.1
c2 = 0.1
w = 0.8
min_value = -10
max_value = 10
```

در این قسمت هم می‌بینیم که پارامترهایی که برای بهینه سازی مانند تعداد ذرات و ابعاد و تکرارها و مقادیر ضرایب شتاب c1 و c2 و وزن اینرسی w تنظیم کرده ایم. حداقل و حداکثر مقادیر را هم تعریف کرده‌ایم.

```
# Set bounds for the variables
lb = [min_value] * num_dimensions
ub = [max_value] * num_dimensions
```

کران پایینی (lb) و کران بالایی (ub) را برای متغیرها تنظیم کردیم. در اینجا، لیست هایی با حداقل و حداکثر مقادیر تکرار شده برای تعداد ابعاد ایجاد می کنیم.

```
# Perform PSO optimization
best_position, best_value = pso(objective_function, lb, ub,
maxiter=num_iterations, swarmsize=num_particles, phip=c1, phig=c2, omega=w)
```

برای انجام بهینه سازی pso تابع pso را از کتابخانه pyswarm فراخوانی کرده‌ایم. تابع هدف و کران پایین و بالا و حداکثر تعداد تکرار و تعداد ذرات و مقادیر شتاب و وزن اینرسی را به تابع پاس داده‌ایم. تابع بهترین موقعیت و مقدار تابع هدف را برمیگرداند.

```
print("Optimal position:", best_position)
print("Optimal value:", best_value)
```

دو مقدار بالا را چاپ می‌کنیم.

```
import numpy as np

# Objective function
def objective_function(x):
    return (x[0] - 3.14) ** 2 + (x[1] - 2.72) ** 2 + np.sin(3 * x[0] + 1.41) +
np.sin(4 * x[1] - 1.73)
```

اینجا همانند کد اول می‌باشد. پس نیازی به توضیح اضافه نیست.

```
# Set parameters
num_particles = 50
num_dimensions = 2
num_iterations = 100
c1 = 0.1
c2 = 0.1
w = 0.8
min_value = -10
max_value = 10
```

پارامترهای تابع pso را ست کرده‌ایم.

```
# Initialize particles
particles = np.random.uniform(low=min_value, high=max_value, size=(num_particles,
num_dimensions))
velocities = np.zeros((num_particles, num_dimensions))
best_positions = particles.copy()
best_values = np.zeros(num_particles)

# Perform PSO optimization
global_best_position = None
global_best_value = np.inf
```

موقعیت و سرعت ذرات را با استفاده از توزیع یکنواخت تصادفی در محدوده های مشخص شده راه اندازی میکنیم.

آرایه هایی را برای ذخیره بهترین موقعیت ها و مقادیر برای هر ذره ایجاد کردیم که در ابتدا به ترتیب در موقعیت فعلی و بی نهایت تنظیم شده اند.

بهترین موقعیت و ارزش جهانی را راه‌اندازی کردیمو در ابتدا روی بی‌نهایت و هیچ تنظیم شده است.

```
for _ in range(num_iterations):
    for i in range(num_particles):
        current_position = particles[i]
        current_value = objective_function(current_position)

        if current_value < best_values[i]:
            best_positions[i] = current_position
            best_values[i] = current_value

        if current_value < global_best_value:
            global_best_position = current_position
            global_best_value = current_value

        r1 = np.random.random(num_dimensions)
        r2 = np.random.random(num_dimensions)

        cognitive_component = c1 * r1 * (best_positions[i] - current_position)
        social_component = c2 * r2 * (global_best_position - current_position)

        velocities[i] = w * velocities[i] + cognitive_component +
social_component
        particles[i] = current_position + velocities[i]

print("Optimal position:", global_best_position)
print("Optimal value:", global_best_value)
```

بهینه سازی pso را با تکرار بر روی تعداد مشخصی انجام می‌دهیم. در هر تکرار روی هر ذره تکرار را انجام میدهیم. سرعت ذره را بر اساس مولفه‌های اجتماعی و شناختی آپدیت می‌کنیم. همچنین موقعیت ذره را هم بر اساس سرعت آپدیت شده آپدیت می‌کنیم.

مقدار تابع هدف را برای موقعیت فعلی ارزیابی میکنیم. اگر مقدار فعلی بهتر باشه بهترین موقعیت و مقدار هر ذره را آپدیت میکنیم. اگر مقدار فعلی بهترین است بهترین موقعیت و ارزش جهانی را هم آپدیت میکنیم.

اعداد تصادفی r1 و r2 را با استفاده از تابع random برای هر مسئله تولید میکنمی که برای محاسبه مولفه‌های شناختی و اجتماعی در الگوریتم pso استفاده میشود. با استفاده از فرمول مولفه شناختی و مولفه اجتماعی را به روز میکنیم. مولفه شناختی نشان دهنده جاذبه ذره به سمت بهترین موقعیت شخصی خود است و با ضریب شناختی c1 وزن می‌شود. مولفه اجتماعی نشان دهنده جاذبه ذره به سمت بهترین موقعیت جهانی است و با ضریب شتاب اجتماعی c2 وزن می‌شود.

سپس سرعت ذره را با استفاده از سرعت قبلی وزن شده با وزن اینرسی به روز کرده و اجزای شناختی و اجتماعی را اضافه میکنیم. موقعیت ذره را با اضافه کردن سرعت به روز شده به موقعیت فعلی به روز میکنیم.

در نهایت موقعیت و مقدار بهینه بدست آمده از بهینه سازی pso را چاپ میکنیم.

```python
import numpy as np
from scipy.optimize import minimize

# Objective function
def objective_function(x):
    return (x[0] - 3.14) ** 2 + (x[1] - 2.72) ** 2 + np.sin(3 * x[0] + 1.41) +
np.sin(4 * x[1] - 1.73)

# Initial guess for optimization
initial_guess = [0, 0]

# Perform optimization
result = minimize(objective_function, initial_guess, method='BFGS')
optimal_position = result.x
optimal_value = result.fun

print("Optimal position (SciPy):", optimal_position)
print("Optimal value (SciPy):", optimal_value)
```

در این کد هم ابتدا کتابخانه‌های نامپای و scipy بخش بهینه سازی آن را فراخوانی کردیم و تابع هدف را تعریف کردیم. سپس یک حدس اولیه را برای بهینه سازی تنظیم کرده و بهینه سازی را با استفاده از تابع minimize از توابع scipy.optimize انجام میدهیم. روش مورد استفاده در اینجا BFGS است. موقعیت و مقدار بهینه بدست آمده از بهینه سازی scipy را هم چاپ میکنیم.

روش BFGS (Broyden Fletcher Goldfarb Shanno) یک الگوریتم بهینه سازی محبوب است که برای مسائل بهینه سازی بدون محدودیت استفاده می‌شود. این روش متعلقه به کلاس روش های شبه نیوتونی است و به عنوان یک روش بهینه سازی مرتبه دوم شناخته می‌شود. BFGS معکوس ماتریس Hessian را که مشتقات مرتبه دوم تابع هدف را نشان می‌دهد. تقریب میزند تا به طور مکرر جهت جستجو را آپدیت کند و را ه حل بهینه را پیدا کند.

```
PS C:\DATA\uni\CI\new\practice\answer> python .\code-with-scipy.py
Optimal position (SciPy): [3.18515531 3.12980267]
Optimal value (SciPy): -1.8083520359223573
PS C:\DATA\uni\CI\new\practice\answer> python .\code-with-pyswarm.py
Stopping search: Swarm best objective change less than 1e-08
Optimal position: [3.18515342 3.12981201]
Optimal value: -1.808352035157038
PS C:\DATA\uni\CI\new\practice\answer> python .\code-without-pyswarm.py
Optimal position: [3.16563496 3.12795079]
Optimal value: -1.806212101915508
```