**计算智能 作业5**

2021年11月11日

**流程：**

1、初始化权值（A,D,U0）

2、计算N个城市的距离矩阵dxy

3、初始化神经网络的输入电压Uxi和输出电压Vxi

4、利用动力微分方程计算：dUxi/dt

5、由一阶欧拉方法更新计算：Uxi(t+1) = Uxi(t) + dUxi/dt \* step

6、由非线性函数sigmoid更新计算：Vxi(t) = 0.5 \* (1 + th(Uxi/U0))

7、计算能量函数E

8、检查路径是否合法

图表, 折线图

描述已自动生成

文本

描述已自动生成

Code：

import numpy as np

from matplotlib import pyplot as plt

# 代价函数（具有三角不等式性质）

def price\_cn(vec1, vec2):

return np.linalg.norm(np.array(vec1) - np.array(vec2))

def calc\_distance(path):

dis = 0.0

for i in range(len(path) - 1):

dis += distance[path[i]][path[i+1]]

return dis

# 得到城市之间的距离矩阵

def get\_distance(citys):

N = len(citys)

distance = np.zeros((N, N))

for i, curr\_point in enumerate(citys):

line = []

[line.append(price\_cn(curr\_point, other\_point)) if i != j else line.append(0.0) for j, other\_point in enumerate(citys)]

distance[i] = line

return distance

# 动态方程计算微分方程du

def calc\_du(V, distance):

a = np.sum(V, axis=0) - 1 # 按列相加

b = np.sum(V, axis=1) - 1 # 按行相加

t1 = np.zeros((N, N))

t2 = np.zeros((N, N))

for i in range(N):

for j in range(N):

t1[i, j] = a[j]

for i in range(N):

for j in range(N):

t2[j, i] = b[j]

# 将第一列移动到最后一列

c\_1 = V[:, 1:N]

c\_0 = np.zeros((N, 1))

c\_0[:, 0] = V[:, 0]

c = np.concatenate((c\_1, c\_0), axis=1)

c = np.dot(distance, c)

return -A \* (t1 + t2) - D \* c

# 更新神经网络的输入电压U

def calc\_U(U, du, step):

return U + du \* step

# 更新神经网络的输出电压V

def calc\_V(U, U0):

return 1 / 2 \* (1 + np.tanh(U / U0))

# 计算当前网络的能量

def calc\_energy(V, distance):

t1 = np.sum(np.power(np.sum(V, axis=0) - 1, 2))

t2 = np.sum(np.power(np.sum(V, axis=1) - 1, 2))

idx = [i for i in range(1, N)]

idx = idx + [0]

Vt = V[:, idx]

t3 = distance \* Vt

t3 = np.sum(np.sum(np.multiply(V, t3)))

e = 0.5 \* (A \* (t1 + t2) + D \* t3)

return e

# 检查路径的正确性

def check\_path(V):

newV = np.zeros([N, N])

route = []

for i in range(N):

mm = np.max(V[:, i])

for j in range(N):

if V[j, i] == mm:

newV[j, i] = 1

route += [j]

break

return route, newV

# 可视化画出哈密顿回路和能量趋势

def draw\_H\_and\_E(citys, H\_path, energys):

fig = plt.figure()

# 绘制哈密顿回路

ax1 = fig.add\_subplot(121)

plt.xlim(0, 7)

plt.ylim(0, 7)

for (from\_, to\_) in H\_path:

p1 = plt.Circle(citys[from\_], 0.2, color='red')

p2 = plt.Circle(citys[to\_], 0.2, color='red')

ax1.add\_patch(p1)

ax1.add\_patch(p2)

ax1.plot((citys[from\_][0], citys[to\_][0]), (citys[from\_][1], citys[to\_][1]), color='red')

ax1.annotate(text=to\_, xy=citys[to\_], xytext=(-8, -4), textcoords='offset points', fontsize=20)

ax1.axis('equal')

ax1.grid()

# 绘制能量趋势图

ax2 = fig.add\_subplot(122)

ax2.plot(np.arange(0, len(energys), 1), energys, color='red')

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

#citys = np.array([[2, 6], [2, 4], [1, 3], [4, 6], [5, 5], [4, 4], [6, 4], [3, 2]])

citys = np.loadtxt('./p01\_xy.txt', dtype=np.double, delimiter=' ')

#distance = get\_distance(citys)

distance = np.loadtxt('./p01\_d.txt', delimiter=' ')

print(distance)

N = len(citys)

# 设置初始值

A = N \* N

D = N / 2

U0 = 0.0009 # 初始电压

step = 0.0001 # 步长

num\_iter = 100000 # 迭代次数

# 初始化神经网络的输入状态（电路的输入电压U）

U = 1 / 2 \* U0 \* np.log(N - 1) + (2 \* (np.random.random((N, N))) - 1)

# 初始化神经网络的输出状态（电路的输出电压V）

V = calc\_V(U, U0)

energys = np.array([0.0 for x in range(num\_iter)]) # 每次迭代的能量

best\_distance = np.inf # 最优距离

best\_route = [] # 最优路线

H\_path = [] # 哈密顿回路

# 开始迭代训练网络

for n in range(num\_iter):

# 利用动态方程计算du

du = calc\_du(V, distance)

# 由一阶欧拉法更新下一个时间的输入状态（电路的输入电压U）

U = calc\_U(U, du, step)

# 由sigmoid函数更新下一个时间的输出状态（电路的输出电压V）

V = calc\_V(U, U0)

# 计算当前网络的能量E

energys[n] = calc\_energy(V, distance)

# 检查路径的合法性

route, newV = check\_path(V)

if len(np.unique(route)) == N:

route.append(route[0])

dis = calc\_distance(route)

if dis < best\_distance:

H\_path = []

best\_distance = dis

best\_route = route

[H\_path.append((route[i], route[i + 1])) for i in range(len(route) - 1)]

print('第{}次迭代找到的次优解距离为：{}，能量为：{}，路径为：'.format(n, best\_distance, energys[n]))

[print(v, end=',' if i < len(best\_route) - 1 else '\n') for i, v in enumerate(best\_route)]

if len(H\_path) > 0:

draw\_H\_and\_E(citys, H\_path, energys)

else:

print('没有找到最优解')