Optimisation Algorithms for Data Analysis Week 8 Assignment

Jiaming Deng 22302794

(a)

(i) Firstly, the input parameters of the global random search algorithm are the function, the amount of data, the dataset, and the number of iterations. The algorithm is implemented by first reading the data of the dataset into the l and u arrays, where l stores the minimum value and u stores the maximum value. The best x and f values are initialised as X, and F as 0 and the largest int type value respectively. Next it iterate through itr_times several times, each time taking a random number from the range of l and u and putting it into the array, and calculating the function value, updating the best x and f values if the function value is smaller than the best value.

(ii) Firstly the two functions used in this problem are as follows.

$$f_1(x, y) = 3(x-3)^4 + 9(y-8)^2$$

 $f_2(x, y) = Max(x-3, 0) + 9 \cdot |y-8|$

I use sympy to initialise the two variables x and y, define two equations based on the above functions and use the diff() function to get the partial derivatives of the two functions.

$$\frac{\partial f_1}{\partial x} = 12(x-3)^3, \ \frac{\partial f_1}{\partial y} = 18y - 144$$

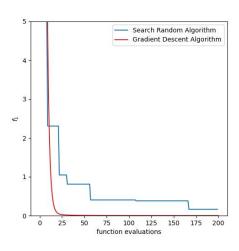
$$\frac{\partial f_2}{\partial x} = \theta(x-3), \ \frac{\partial f_2}{\partial y} = 9 \cdot \text{sign}(y-2)$$

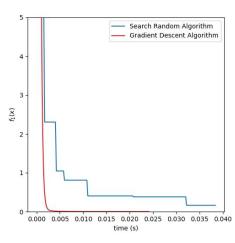
The gradient descent algorithm used in this problem was implemented in the Week 2 assignment and uses a parameter alpha of 0.01. Measuring the execution time and

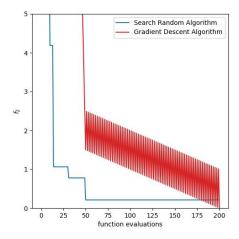
comparing the gradient descent algorithm with the global random search algorithm is not easy because the execution times of the two algorithms are very short and the differences are small. I therefore used the timeit function, which specifies the number of times the function is executed and counts the time, and I specified that each algorithm is executed 500 times. The results of the experiment are as follows.

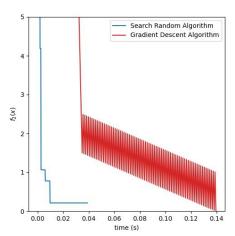
	Gradient Descent (s)	Global Random Search (s)
f1	0.1371	0.1743
f2	0.6604	0.1701

Next I plot the change in function value with the number of function evaluations and the change in function value with time for $f_1(x)$ and $f_2(x)$ respectively. For $f_1(x)$, the Gradient Descent algorithm converged to a minimum around the 30th iteration in about 0.005s, but the Global Search algorithm did not converge to a minimum even after 200 iterations. For $f_2(x)$, both the Gradient Descent algorithm and the global search algorithm did not converge to a minimum in 200 iterations. The Gradient Descent algorithm behaves as a jagged structure with poor stability after the 50th iteration. The Global Search algorithm converged quickly in a few iterations, but mostly showed no convergence because of the instability of the random numbers generated in the algorithm.







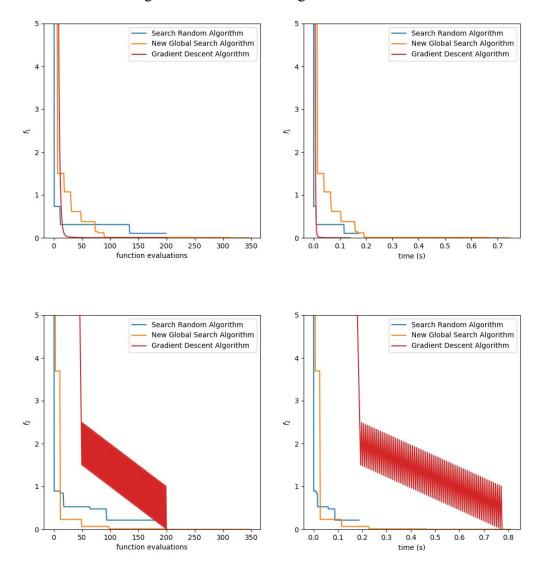


(i) The algorithm in this problem is based on the Global Random Search algorithm with some improvements. First, the minimum and maximum values of the generated numbers are specified according to the arrays 1 and u and an array of length N is randomly generated, the elements of the array are vectors of length 2 which are the x and y parameters of the function and the function value is calculated. Next, in line with the Global Random Search algorithm, if the function value is smaller than the previously saved minimum function value then the minimum function value is updated. Finally, in order to select M results, it is updated to N-M/M values by multiplying by a random number between 0.7 and 1.3.

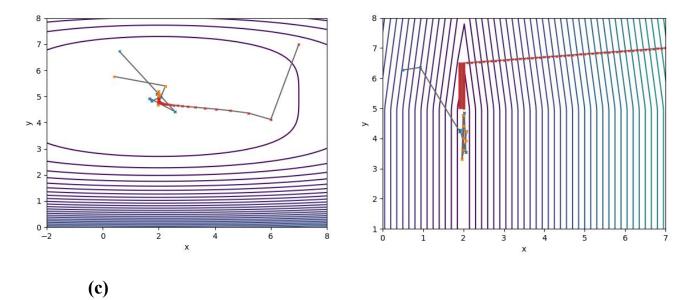
```
x_list = []
f_list = []
current_x_list = []
for i in range(N):
    tmp = []
    for j in range(n):
        tmp.append(uniform(l[j], u[j]))
    current_x_list.append(tmp)
current_f_list = [0] * N
for i in range(N):
    current_x = current_x_list[i]
    current_f = f(current_x[0], current_x[1])
    if current_f < F:</pre>
        X = current_x
        F = current_f
    x_list.append(X)
    f_list.append(F)
opt_times = (N - M) // M
for _ in range(itr_times):
    current_f_list, current_x_list = sort_lists(current_f_list, current_x_list)
    for i in range(M):
        current_x = current_x_list[i]
        for j in range(opt_times):
            x_{plus} = [x * uniform(0.7, 1.3) for x in current_x]
            k = M + (i * opt_times) + j
            current_x_list[k] = x_plus
            current_f_list[k] = f(x_plus[0], x_plus[1])
            if current_f_list[k] < F:</pre>
                X = current_x_list[k]
                F = current_f_list[k]
            x_list.append(X)
            f_list.append(F)
return x_list, f_list
```

(ii) Firstly, I set the parameters N to 50, M to 20, and the number of iterations to 15 for the algorithm after improving Global Random Search, and similarly to question (a) use the timeit function to measure the running time of the algorithm and plot the change in function value with the number of function evaluations and the change in function value with time for $f_1(x)$ and $f_2(x)$ respectively and plot the change in x for $f_1(x)$ and $f_2(x)$ respectively. The results obtained are shown in the figure below.

As can be seen in the figure below, for $f_1(x)$, only the random search algorithm did not converge to a minimum during the iterations, while the other two algorithms did. The Gradient Descent algorithm reached a minimum at the 50th iteration and the New Global Search algorithm dropped to a minimum at the 100th iteration. For $f_2(x)$, only the New Global Search algorithm converged to a minimum at the 100th iteration, while the other two algorithms did not converge to a minimum.

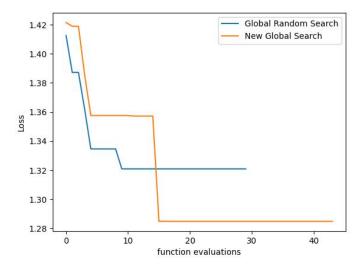


As can be seen from the contour plot, for $f_1(x)$ the Gradient Descent algorithm converges smoothly but the other two algorithms are very tortuous. For $f_2(x)$, the New Global Search algorithm converges quickly to a minimum, and the Gradient Descent algorithm converges in a jagged fashion.



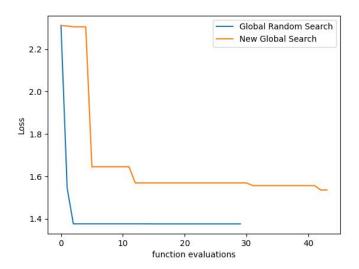
In this question, I use the downloaded conv net model and apply the Global Random Search Algorithm and the New Global Search Algorithm from the last two problems, where the parameters are chosen to be 20 for N, 5 for M, and 3 for the number of iterations, and use mini-batch size, adam parameters, and number of epochs as hyperparameters respectively.

(i) The mini-batch size was chosen to range from 0 to 128, and the parameters of the fixed adam algorithm were $\alpha = 0.01$, $\beta 1 = 0.9$, $\beta 2 = 0.99$, and fixed epochs of 10. The New Global Search algorithm performs better than the Global Random Search algorithm and the best batch size is around 15.

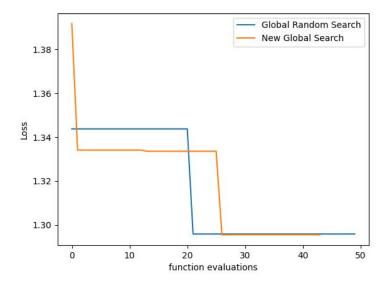


(ii) For the Adam algorithm, I chose parameters α ranging from 0.1 to 0.001, β 1 ranging from 0.25 to 0.99, β 2 ranging from 0.9 to 0.999, a fixed small batch size of 15 and an epoch of 15 for the experiment. The global random search algorithm performed better than the new global search algorithm The global random search

algorithm performed better than the new global search algorithm. The best choice of parameters was $\alpha = 0.001$. $\beta 1 = 0.9$ and $\beta 2 = 0.999$.



(iii) I chose epochs in the range of 10 to 30. According to the experimental results of the above two problems, the specified mini-batch size is 15, Adam's parameters $\alpha = 0.001$, $\beta 1 = 0.9$ and $\beta 2 = 0.999$. The Global Random Search algorithm performed slightly better than the New Global Search algorithm, with the best epochs being around 21.



Appendix

a(i)

```
1. import sys
2. from random import uniform
3.
4.
5. def random search(f, n, data range, N):
6.
      1 = []
7.
      u = []
8.
      for Range in data range:
9.
           l.append(Range[0])
10.
       for Range in data range:
11.
           u.append(Range[1])
12.
       X = 0
13.
       F = sys.maxsize
14.
       x list = []
15.
       f list = []
16.
       for i in range(N):
17.
           data x = []
18.
           for j in range(n):
19.
                x = uniform(l[j], u[j])
20.
                data x.append(x)
21.
           data f = f(data x[0], data x[1])
22.
           if data f < F:</pre>
23.
                X = data x
24.
                F = data f
25.
           x list.append(X)
26.
           f list.append(F)
27.
       return x list, f list
```

a(ii)

```
1. from random import uniform
2. from timeit import timeit
3. import matplotlib
4. from a_i import random_search
5.
6. matplotlib.use('TkAgg')
7. import matplotlib.pyplot as plt
8. import numpy as np
9. import sys
10.
11.
```

```
12.def gradient descent(f, df, n, x0, iter times, alph
  a=0.01):
13.
      x = x0
      f = f(x0[0], x0[1])
14.
15.
      x list = []
16.
      f list = []
17.
      x list.append(x0)
18.
      f list.append(f)
19.
       for i in range(iter times):
20.
           for j in range(n):
21.
               x[j] = alpha * df[j](x[j])
22.
           f = f(x[0], x[1])
23.
           x list.append(x)
24.
           f list.append(f)
25.
       return x list, f list
26.
27.
28.def func 1():
       f = lambda x, y: 3 * (x - 3) ** 4 + 9 * (y - 8)
30.
   dx = lambda x: 12 * (x - 3) ** 3
      dy = lambda y: 18 * y - 144
31.
32.
      return f, (dx, dy)
33.
34.
35.def func 2():
       f = lambda x, y: 9 * abs(y - 8) + max(0, x - 3)
37.
      dx = lambda x: np.heaviside(x - 3, 0)
      dy = lambda y: 9 * np.sign(y - 8)
38.
39.
       return f, (dx, dy)
40.
41.
42.if name == ' main ':
       f1, df1 = func 1()
43.
44.
      f2, df2 = func 2()
45.
      n = 2
46.
      data range = [[4, 8], [0, 3]]
47.
      x0 = [7, 7]
48.
      iter times = 200
49.
       New Global Search N, New Global Search M, New G
  lobal Search n = 20, 5, 10
```

- 50. global_random_search_time1 = timeit(lambda: ran
 dom_search(f1, n, data_range, N=iter_times), number
 =500)
- 51. gradient_descent_time1 = timeit(lambda: gradien
 t_descent(f1, df1, n, x0, iter_times=iter_times), n
 umber=500)
- 52. global_random_search_time2 = timeit(lambda: ran dom_search(f2, n, data_range, N=iter_times), number =500)
- 53. gradient_descent_time2 = timeit(lambda: gradien
 t_descent(f2, df2, n, x0, iter_times=iter_times), n
 umber=500)
- 54. print(f'Global Random Search for f1(x) = {globa
 l random search time1}')
- 55. print(f'Gradient Descent for f1(x) = {gradient_ descent time1}')
- 56. print(f'Global Random Search for f2(x) = {globa
 l_random_search_time2}')
- 57. print(f'Gradient Descent for f2(x) = {gradient_ descent time2}')
- 58.
- 59. global_random_search_x_list, global_random_sear
 ch_f_list = random_search(f1, n, data_range, N=iter
 _times)
- 60. gradient_descent_x_list, gradient_descent_f_lis
 t = gradient_descent(f1, df1, n, x0, iter_times=ite
 r_times)
- 61. global_random_search_x_list_2, global_random_se
 arch_f_list_2 = random_search(f2, n, data_range, N=
 iter times)
- 62. gradient_descent_x_list_2, gradient_descent_f_l
 ist_2 = gradient_descent(f2, df2, n, x0, iter_times
 =iter times)
- 63. global_random_search_x_list_ = list(range(len(g
 lobal random search f list)))
- 64. gradient_descent_x_list_ = list(range(len(gradient_descent_f_list)))

- 67. plt.plot(global_random_search_x_list_, global_r andom_search_f_list, label='Global Random Search Al gorithm', color='tab:blue')

```
68. plt.plot(gradient descent x list , gradient des
      cent f list, label=f'Gradient Descent Algorithm', c
      olor='tab:red')
   69.
          plt.xlabel('function evaluations')
    70.
          plt.ylabel('f1')
    71.
          plt.legend()
    72.
          plt.show()
    73.
   74.
           plt.plot(global random search x list 2 , global
      random search f list 2, label='Global Random Searc
      h Algorithm',
   75.
                    color='tab:blue')
   76.
           plt.plot(gradient descent x list 2 , gradient d
      escent f list 2, label=f'Gradient Descent Algorithm
      ', color='tab:red')
   77.
          plt.xlabel('function evaluations')
    78.
          plt.ylabel('f2')
    79.
          plt.legend()
    80.
          plt.show()
b(i)
   1. from random import uniform
   2. import sys
   3.
   4.
   5. def Sort(a, b):
          return map(list, zip(*sorted(zip(a, b))))
   7.
   9. def new global search(f, n, data range, N, M, itr t
      imes):
   10.
          l = []
   11.
          u = []
    12.
          for Range in data range:
   13.
               l.append(Range[0])
   14.
          for Range in data range:
   15.
               u.append(Range[1])
    16.
          X = 0
   17.
          F = sys.maxsize
    18.
          x list = []
   19.
           f list = []
   20.
          current x list = []
   21.
          for i in range(N):
   22.
               tmp = []
```

```
23.
               for j in range(n):
    24.
                   tmp.append(uniform(l[j], u[j]))
    25.
               current x list.append(tmp)
    26.
           current f list = [0] * N
    27.
           for i in range(N):
    28.
               current x = current x list[i]
    29.
               current f = f(current x[0], current x[1])
    30.
               current f list[i] = current f
    31.
               if current f < F:</pre>
    32.
                   X = current x
    33.
                   F = current f
    34.
               x list.append(X)
    35.
               f list.append(F)
    36.
           opt times = (N - M) // M
    37.
           for in range(itr times):
               current f list, current_x_list = Sort(curre
    38.
      nt f list, current x list)
    39.
               for i in range(M):
    40.
                   current x = current x list[i]
    41.
                   for j in range(opt times):
    42.
                        x plus = [x * uniform(0.7, 1.3) for
       x in current x]
    43.
                        k = M + (i * opt times) + j
    44.
                        current x list[k] = x plus
    45.
                        current f list[k] = f(x plus[0], x
      plus[1])
    46.
                        if current f list[k] < F:</pre>
    47.
                            X = current x list[k]
    48.
                            F = current f list[k]
    49.
                        x list.append(X)
    50.
                        f list.append(F)
    51.
           return x list, f list
b(ii)
    1. from timeit import timeit
    2. import matplotlib
    3.
    4. from a i import random search
    5. from a ii import gradient descent
    6. from b i import new global search
    7.
    8. matplotlib.use('TkAgg')
    9. import matplotlib.pyplot as plt
    10.import numpy as np
```

```
11.
12.
13.def func 1():
      f = lambda x, y: 3 * (x - 3) ** 4 + 9 * (y - 8)
   ** 2
15.
      dx = lambda x: 12 * (x - 3) ** 3
      dy = lambda y: 18 * y - 144
16.
17.
      return f, (dx, dy)
18.
19.
20.def func 2():
      f = lambda x, y: 9 * abs(y - 8) + max(0, x - 3)
21.
      dx = lambda x: np.heaviside(x - 3, 0)
22.
23.
      dy = lambda y: 9 * np.sign(y - 8)
24.
      return f, (dx, dy)
25.
26.
27.if name == ' main ':
28.
    f1, df1 = func 1()
29.
      f2, df2 = func 2()
30.
      n = 2
31.
      data range = [[4, 8], [0, 3]]
32.
      x0 = [7, 7]
33.
      iter times = 200
34.
      New Global Search N, New Global Search M, New G
  lobal Search n = 20, 5, 10
35.
      global random search time1 = timeit(lambda: ran
  dom search(f1, n, data range, N=iter times), number
  =500)
      gradient descent time1 = timeit(lambda: gradien
36.
  t descent(f1, df1, n, x0, iter times=iter times), n
  umber=500)
37.
      new global search time1 = timeit(lambda: new gl
  obal search (f1, n, data range, N=New Global Search
  N, M=New Global Search M,
38.
      itr times=New Global Search n), number=500)
      global random search time2 = timeit(lambda: ran
39.
  dom search (f2, n, data range, N=iter times), number
  =500)
      gradient descent time2 = timeit(lambda: gradien
  t descent(f2, df2, n, x0, iter times=iter times), n
  umber=500)
```

```
41.
      new global search time2 = timeit(
42.
          lambda: new global search(f2, n, data range,
   N=New Global Search N, M=New Global Search M,
43.
                                     itr times=New Glo
  bal Search n), number=500)
     print(f'Global Random Search for f1(x) = {globa
  l random search time1}')
      print(f'Gradient Descent for f1(x) = {gradient
45.
  descent time1}')
      print(f'New Global Search for f1(x) = {new glob
46.
  al search time1}')
47.
      print(f'Global Random Search for f2(x) = {globa}
  1 random search time2}')
49.
      print(f'Gradient Descent for f2(x) = {gradient
  descent time2}')
      print(f'New Global Search for f2(x) = {new glob
  al search time2}')
51.
     global random search x list, global random sear
  ch f list = random search(f1, n, data range, N=iter
  times)
53. gradient descent x list, gradient descent f lis
  t = gradient descent(f1, df1, n, x0, iter times=ite
  r times)
      new global search x list, new global search f l
  ist = new global search(f1, n, data range, New Glob
  al Search N, New Global Search M, New Global Search
  n)
55.
      global random search x list 2, global random se
  arch f list 2 = random search(f2, n, data range, N=
  iter times)
     gradient descent x list 2, gradient descent f l
  ist 2 = gradient descent(f2, df2, n, x0, iter times
  =iter times)
     new global search x list 2, new global search f
  list 2 = new global search(f1, n, data range, New
  Global Search N,
58.
                           New Global Search M, New Gl
  obal Search n)
59.
      global random search x list = list(range(len(g
  lobal random search f list)))
```

```
ent descent f list)))
      new global search x list = list(range(len(new
62.
  global search x list)))
      global random search x list 2 = list(range(len
63.
  (global random search f list 2)))
      gradient descent_x_list_2_ = list(range(len(gra
64.
  dient descent f list 2)))
      new global search x list 2 = list(range(len(ne
  w global search x list 2)))
66.
67.
      plt.plot(global random search x list , global r
  andom search f list, label='Global Random Search Al
  gorithm',
68.
               color='tab:blue')
      plt.plot(gradient descent x list , gradient des
  cent f list, label=f'Gradient Descent Algorithm', c
  olor='tab:red')
      plt.plot(new global search x list , new global
  search f list, label=f'New Global Search Algorithm',
   color='tab'
71.
           ':orange')
72.
      plt.xlabel('function evaluations')
73.
      plt.ylabel('f1')
74.
     plt.legend()
75.
      plt.show()
76.
77.
      plt.plot(global random search_x_list_2_, global
  random search f list 2, label='Search Random Algor
  ithm',
78.
               color='tab:blue')
      plt.plot(gradient descent x list 2 , gradient d
79.
  escent f list 2, label=f'Gradient Descent Algorithm
  ١,
80.
              color='tab:red')
      plt.plot(new global search x list , new global
81.
  search f list, label=f'New Global Search Algorithm',
   color='tab'
82.
           ':orange')
83.
      plt.xlabel('function evaluations')
```

gradient descent x list = list(range(len(gradi

61.

```
plt.ylabel('f2')
    85.
          plt.legend()
    86.
          plt.show()
c(i)
   1. from tensorflow import keras
   2. from keras import regularizers
   3. from keras.layers import Dense, Dropout, Flatten
   4. from keras.layers import Conv2D
   5. from keras.losses import CategoricalCrossentropy
   6. from keras.optimizers import Adam
   7. import matplotlib
   8. matplotlib.use('TkAgg')
   9. import matplotlib.pyplot as plt
   10.from b i import new global search
   11.from a i import random search
   12.
   13.def get model loss(batch size, alpha, beta1, beta2,
       epochs):
   14.
         # Model / data parameters
    15.
          num classes = 10
    16.
          input shape = (32, 32, 3)
   17.
   18.
          # the data, split between train and test sets
    19.
           (x train, y train), (x test, y test) = keras.da
      tasets.cifar10.load data()
   20.
         n = 5000
   21.
          x train = x train[1:n]
   22.
         y train = y train[1:n]
   23.
          # x test=x test[1:500]; y test=y test[1:500]
   24.
   25.
          # Scale images to the [0, 1] range
   26.
          x train = x train.astype("float32") / 255
   27.
          x test = x test.astype("float32") / 255
   28.
          print("orig x train shape:", x train.shape)
   29.
   30.
           # convert class vectors to binary class matrice
   31.
          y train = keras.utils.to categorical(y train, n
      um classes)
   32.
           y test = keras.utils.to categorical(y test, num
       classes)
   33.
          model = keras.Sequential()
```

```
34. model.add(Conv2D(16, (3, 3), padding='same', in
  put shape=x train.shape[1:], activation='relu'))
      model.add(Conv2D(16, (3, 3), strides=(2, 2), pa
35.
  dding='same', activation='relu'))
       model.add(Conv2D(32, (3, 3), padding='same', ac
36.
  tivation='relu'))
37.
      model.add(Conv2D(32, (3, 3), strides=(2, 2), pa
  dding='same', activation='relu'))
      model.add(Dropout(0.5))
38.
39.
      model.add(Flatten())
      model.add(Dense(num classes, activation='softma
40.
  x', kernel regularizer=regularizers.11(0.0001)))
41.
      optimizer = Adam(learning rate=alpha, beta 1=be
  ta1, beta 2=beta2)
       model.compile(loss="categorical crossentropy",
42.
  optimizer=optimizer,
43.
          metrics=["accuracy"])
44.
      y predicts = model.predict(x test)
45.
      loss = CategoricalCrossentropy()
46.
      return loss(y test, y predicts).numpy()
47.
48.
49.if
     name == ' main ':
50.
     n = 5
51.
      data range = [
           [1, 128],
52.
53.
           [0.001, 0.001],
54.
           [0.9, 0.9],
           [0.99, 0.99],
55.
56.
           [15, 15]
57.
      1
       global random search x list, global random sear
  ch f list = random search(get model loss, n, data r
  ange, N=30)
       new global search x list, new global search f l
59.
  ist = new global search(get model loss, n, data ran
  qe, N=12, M=4,
60.
                           itr times=4)
61.
       global random search x list , new global search
  x list = list(range(len(global random search f li
  st))), list(range(len(new global search f list)))
```

```
62. plt.plot(global random search x list , global r
      andom search f list, label='Global Random Search')
   63.
          plt.plot(new global search x list , new global
      search f list, label='New Global Search')
   64.
          plt.xlabel('function evaluations')
          plt.ylabel('loss')
   65.
    66.
          plt.legend()
   67.
          plt.show()
c(ii)
   1. from tensorflow import keras
   2. from keras import regularizers
   3. from keras.layers import Dense, Dropout, Flatten
   4. from keras.layers import Conv2D
   5. from keras.losses import CategoricalCrossentropy
   6. from keras.optimizers import Adam
   7. import matplotlib
   8. matplotlib.use('TkAgg')
   9. import matplotlib.pyplot as plt
   10.from b i import new global search
   11.from a i import random search
   12.
   13.def get model loss(batch size, alpha, beta1, beta2,
       epochs):
   14.
          # Model / data parameters
    15.
          num classes = 10
   16.
          input shape = (32, 32, 3)
    17.
   18.
          # the data, split between train and test sets
    19.
           (x train, y train), (x test, y test) = keras.da
      tasets.cifar10.load data()
   20.
          n = 5000
   21.
          x train = x train[1:n]
   22.
          y train = y train[1:n]
   23.
          # x test=x test[1:500]; y test=y test[1:500]
   24.
   25.
          # Scale images to the [0, 1] range
   26.
          x train = x train.astype("float32") / 255
   27.
          x test = x test.astype("float32") / 255
   28.
          print("orig x train shape:", x train.shape)
   29.
    30.
          # convert class vectors to binary class matrice
```

```
31.
       y train = keras.utils.to categorical(y train, n
  um classes)
32.
       y test = keras.utils.to categorical(y test, num
  classes)
33.
      model = keras.Sequential()
      model.add(Conv2D(16, (3, 3), padding='same', in
  put shape=x train.shape[1:], activation='relu'))
35.
       model.add(Conv2D(16, (3, 3), strides=(2, 2), pa
  dding='same', activation='relu'))
       model.add(Conv2D(32, (3, 3), padding='same', ac
36.
  tivation='relu'))
37.
       model.add(Conv2D(32, (3, 3), strides=(2, 2), pa
  dding='same', activation='relu'))
      model.add(Dropout(0.5))
38.
39.
      model.add(Flatten())
      model.add(Dense(num classes, activation='softma
40.
  x', kernel regularizer=regularizers.11(0.0001)))
41.
       optimizer = Adam(learning rate=alpha, beta 1=be
  tal, beta 2=beta2)
42.
       model.compile(loss="categorical crossentropy",
  optimizer=optimizer,
43.
           metrics=["accuracy"])
44.
      y predicts = model.predict(x test)
45.
       loss = CategoricalCrossentropy()
46.
      return loss(y test, y predicts).numpy()
47.
48.
49.if
     name == ' main ':
50.
      n = 5
51.
      data range = [
52.
          [15, 15],
53.
           [0.1, 0.001],
54.
           [0.9, 0.25],
55.
           [0.999, 0.99],
56.
           [15, 15]
57.
58.
       global random search x list, global random sear
  ch f list = random search(get model loss, n, data r
  ange, N=30)
59.
      new global search x list, new global search f l
  ist = new global search(get model loss, n, data ran
  ge, N=12, M=4,
60.
                           itr times=4)
```

```
61.
         global random search x list , new global search
      x list = list(range(len(global random search f li
      st))), list(
   62.
              range(len(new global search f list)))
   63.
          plt.plot(global random search x list , global r
      andom search f list, label='Global Random Search')
   64.
          plt.plot(new global search x list , new global
      search f list, label='New Global Search')
   65.
          plt.xlabel('function evaluations')
   66.
          plt.ylabel('loss')
    67.
          plt.legend()
    68.
          plt.show()
c(iii)
   1. from tensorflow import keras
   2. from keras import regularizers
   3. from keras.layers import Dense, Dropout, Flatten
   4. from keras.layers import Conv2D
   5. from keras.losses import CategoricalCrossentropy
   6. from keras.optimizers import Adam
   7. import matplotlib
   8. matplotlib.use('TkAgg')
   9. import matplotlib.pyplot as plt
   10.from b i import new global search
   11.from a i import random search
   13.def get model loss (batch size, alpha, beta1, beta2,
       epochs):
   14.
          # Model / data parameters
   15.
          num classes = 10
   16.
          input shape = (32, 32, 3)
   17.
         # the data, split between train and test sets
    18.
   19.
           (x train, y train), (x test, y test) = keras.da
      tasets.cifar10.load data()
   20.
         n = 5000
   21.
          x train = x train[1:n]
   22.
          y train = y train[1:n]
   23.
          # x test=x test[1:500]; y test=y test[1:500]
   24.
   25.
          # Scale images to the [0, 1] range
   26.
          x train = x train.astype("float32") / 255
   27.
          x test = x test.astype("float32") / 255
```

```
28.
     print("orig x train shape:", x train.shape)
29.
      # convert class vectors to binary class matrice
30.
31.
       y train = keras.utils.to categorical(y train, n
  um classes)
      y test = keras.utils.to categorical(y test, num
32.
  classes)
33.
      model = keras.Sequential()
      model.add(Conv2D(16, (3, 3), padding='same', in
34.
  put shape=x train.shape[1:], activation='relu'))
      model.add(Conv2D(16, (3, 3), strides=(2, 2), pa
35.
  dding='same', activation='relu'))
      model.add(Conv2D(32, (3, 3), padding='same', ac
  tivation='relu'))
      model.add(Conv2D(32, (3, 3), strides=(2, 2), pa
37.
  dding='same', activation='relu'))
38.
      model.add(Dropout(0.5))
39.
      model.add(Flatten())
40.
      model.add(Dense(num classes, activation='softma
  x', kernel regularizer=regularizers.11(0.0001)))
41.
      optimizer = Adam(learning rate=alpha, beta 1=be
  tal, beta 2=beta2)
42.
       model.compile(loss="categorical crossentropy",
  optimizer=optimizer,
43.
          metrics=["accuracy"])
44.
      y predicts = model.predict(x test)
45.
      loss = CategoricalCrossentropy()
46.
      return loss(y test, y predicts).numpy()
47.
48.
49.if name == ' main ':
50.
      n = 5
51.
      data range = [
52.
           [15, 15],
53.
           [0.001, 0.001],
54.
           [0.9, 0.9],
55.
           [0.999, 0.999],
56.
           [30, 10]
57.
58.
       global random search x list, global random sear
  ch f list = random search (get model loss, n, data r
  ange, N=30)
```

```
59. new global search x list, new global search f l
  ist = new global search(get model loss, n, data ran
  ge, N=12, M=4,
60.
                          itr times=4)
61.
     global random search x list , new global search
  x list = list(range(len(global random search f li
  st))), list(
          range(len(new global search f list)))
63.
   plt.plot(global random search x list , global r
  andom search f list, label='Global Random Search')
      plt.plot(new global search x list , new global
  search f list, label='New Global Search')
65.
      plt.xlabel('function evaluations')
      plt.ylabel('loss')
66.
67.
      plt.legend()
68.
     plt.show()
```