learning_rate

July 1, 2022

1 Learning rate - Problem 8

1.1 Description

1.1.1 Average time : 200 minutes

1.1.2 PDE

We will try to find the best learning rate to the problem 8 of the article: https://ieeexplore.ieee.org/document/712178

$\Delta\psi(x,y) + \psi(x,y) \cdot \frac{\partial\psi(x,y)}{\partial y} = f(x,y)$ on $\Omega = [0,1]^2$
where $f(x,y) = \sin(\pi x)(2 - \pi^2 y^2 + 2y^3 \sin(\pi x))$

1.1.3 Boundary conditions

$\psi(0,y) = \psi(1,y) = \psi(x,0) = 0$ and $\frac{\partial\psi}{\partial y}(x,1) = 2\sin(\pi x)$

1.1.4 Loss function

The loss to minimize here is $\mathcal{L} = ||\Delta\psi(x,y) + \psi(x,y) \cdot \frac{\partial\psi(x,y)}{\partial y} - f(x,y)||_2$

1.1.5 Analytical solution

The true function ψ should be $\psi(x,y) = y^2 sin(\pi x)$
This solution is the same of the problem 7

1.1.6 Approximated solution

We want find a solution $\psi(x,y) = A(x,y) + F(x,y)N(x,y)$ s.t:
$F(x,y) = \sin(x-1)\sin(y-1)\sin(x)\sin(y)$ $A(x,y) = y\sin(\pi x)$

2 Importing libraries

[ ]:
```
# Jax libraries
from jax import value_and_grad,vmap,jit,jacfwd
from functools import partial
from jax import random as jran
from jax.example_libraries import optimizers as jax_opt
from jax.nn import tanh, sigmoid, elu, relu, gelu
```

```python
from jax.lib import xla_bridge
import jax.numpy as jnp

# Others libraries
from time import time
import matplotlib.pyplot as plt
import numpy as np
import os
import pickle
print(xla_bridge.get_backend().platform)
```

gpu

3 Multilayer Perceptron

[ ]:
```python
class MLP:
    """
        Create a multilayer perceptron and initialize the neural network
    Inputs :
        A SEED number and the layers structure
    """

    # Class initialization
    def __init__(self,SEED,layers):
        self.key=jran.PRNGKey(SEED)
        self.keys = jran.split(self.key,len(layers))
        self.layers=layers
        self.params = []

    # Initialize the MLP weigths and bias
    def MLP_create(self):
        for layer in range(0, len(self.layers)-1):
            in_size,out_size=self.layers[layer], self.layers[layer+1]
            std_dev = jnp.sqrt(2/(in_size + out_size ))
            weights=jran.truncated_normal(self.keys[layer], -2, 2,␣
 ↪shape=(out_size, in_size), dtype=np.float32)*std_dev
            bias=jran.truncated_normal(self.keys[layer], -1, 1,␣
 ↪shape=(out_size, 1), dtype=np.float32).reshape((out_size,))
            self.params.append((weights,bias))
        return self.params

    # Evaluate a position XY using the neural network
    @partial(jit, static_argnums=(0,))
    def NN_evaluation(self,new_params, inputs):
        for layer in range(0, len(new_params)-1):
            weights, bias = new_params[layer]
```

```
            inputs = gelu(jnp.add(jnp.dot(inputs, weights.T), bias))
        weights, bias = new_params[-1]
        output = jnp.dot(inputs, weights.T)+bias
        return output

    # Get the key associated with the neural network
    def get_key(self):
        return self.key
```

4 Two dimensional PDE operators

[ ]:
```
class PDE_operators2d:
    """
        Class with the most common operators used to solve PDEs
    Input:
        A function that we want to compute the respective operator
    """

    # Class initialization
    def __init__(self,function):
        self.function=function

    # Compute the two dimensional laplacian
    def laplacian_2d(self,params,inputs):
        fun = lambda params,x,y: self.function(params, x,y)
        @partial(jit)
        def action(params,x,y):
            u_xx = jacfwd(jacfwd(fun, 1), 1)(params,x,y)
            u_yy = jacfwd(jacfwd(fun, 2), 2)(params,x,y)
            return u_xx + u_yy
        vec_fun = vmap(action, in_axes = (None, 0, 0))
        laplacian = vec_fun(params, inputs[:,0], inputs[:,1])
        return laplacian

    # Compute the partial derivative in x
    @partial(jit, static_argnums=(0,))
    def du_dx(self,params,inputs):
        fun = lambda params,x,y: self.function(params, x,y)
        @partial(jit)
        def action(params,x,y):
            u_x = jacfwd(fun, 1)(params,x,y)
            return u_x
        vec_fun = vmap(action, in_axes = (None, 0, 0))
        return vec_fun(params, inputs[:,0], inputs[:,1])

    # Compute the partial derivative in y
```

```
    @partial(jit, static_argnums=(0,))
    def du_dy(self,params,inputs):
        fun = lambda params,x,y: self.function(params, x,y)
        @partial(jit)
        def action(params,x,y):
            u_y = jacfwd(fun, 2)(params,x,y)
            return u_y
        vec_fun = vmap(action, in_axes = (None, 0, 0))
        return vec_fun(params, inputs[:,0], inputs[:,1])
```

5 Physics Informed Neural Networks

[ ]:
```
class PINN:
    """
    Solve a PDE using Physics Informed Neural Networks
    Input:
        The evaluation function of the neural network
    """

    # Class initialization
    def __init__(self,NN_evaluation):
        self.operators=PDE_operators2d(self.solution)
        self.laplacian=self.operators.laplacian_2d
        self.NN_evaluation=NN_evaluation
        self.dsol_dy=self.operators.du_dy

    # Definition of the function A(x,y) mentioned above
    @partial(jit, static_argnums=(0,))
    def A_function(self,inputX,inputY):
        return jnp.multiply(inputY,jnp.sin(jnp.pi*inputX)).reshape(-1,1)

    # Definition of the function F(x,y) mentioned above
    @partial(jit, static_argnums=(0,))
    def F_function(self,inputX,inputY):
        F1=jnp.multiply(jnp.sin(inputX),jnp.sin(inputX-jnp.ones_like(inputX)))
        F2=jnp.multiply(jnp.sin(inputY),jnp.sin(inputY-jnp.ones_like(inputY)))
        return jnp.multiply(F1,F2).reshape((-1,1))

    # Definition of the function f(x,y) mentioned above
    @partial(jit, static_argnums=(0,))
    def target_function(self,inputs):
        return jnp.multiply(jnp.sin(jnp.pi*inputs[:,0]),2-jnp.pi**2*inputs[:
↪,1]**2+2*inputs[:,1]**3*jnp.sin(jnp.pi*inputs[:,0])).reshape(-1,1)

    # Compute the solution of the PDE on the points (x,y)
    @partial(jit, static_argnums=(0,))
```

```
    def solution(self,params,inputX,inputY):
        inputs=jnp.column_stack((inputX,inputY))
        NN = vmap(partial(jit(self.NN_evaluation), params))(inputs)
        F=self.F_function(inputX,inputY)
        A=self.A_function(inputX,inputY)
        return jnp.add(jnp.multiply(F,NN),A).reshape(-1,1)

    # Compute the loss function
    @partial(jit, static_argnums=(0,))
    def loss_function(self,params,batch):
        targets=self.target_function(batch)
        laplacian=self.laplacian(params,batch).reshape(-1,1)
        dsol_dy_values=self.dsol_dy(params,batch)[:,0].reshape((-1,1))
        preds=laplacian+jnp.multiply(self.solution(params,batch[:,0],batch[:
↪,1]),dsol_dy_values).reshape(-1,1)
        return jnp.linalg.norm(preds-targets)

    # Train step
    @partial(jit, static_argnums=(0,))
    def train_step(self,i, opt_state, inputs):
        params = get_params(opt_state)
        loss, gradient = value_and_grad(self.loss_function)(params,inputs)
        return loss, opt_update(i, gradient, opt_state)
```

6 Initialize neural network

[ ]:
```
# Neural network parameters
SEED = 351
n_features, n_targets = 2, 1             # Input and output dimension
layers = [n_features,30,n_targets]       # Layers structure

# Initialization
NN_MLP=MLP(SEED,layers)
params = NN_MLP.MLP_create()             # Create the MLP
NN_eval=NN_MLP.NN_evaluation             # Evaluate function
solver=PINN(NN_eval)
key=NN_MLP.get_key()
```

7 Train parameters

[ ]:
```
batch_size = 50
num_batches = 100000
report_steps=1000
```

8 Learning rate

```
[ ]: init, end, interval_lenght = 0, 6, 25
     # Learning rate values
     intervals = jnp.array([jnp.linspace(10**(-i),10**(-i)/
      ↪interval_lenght,interval_lenght) for i in range(init,end)])
     learning_rate = jnp.unique(jnp.sort(intervals.reshape(-1,1)[:,0]))
     print(len(learning_rate))
     print(learning_rate)
```

```
147
[4.0000000e-07 7.9999961e-07 1.1999998e-06 1.6000000e-06 1.9999995e-06
 2.3999999e-06 2.7999999e-06 3.1999996e-06 3.5999997e-06 4.0000000e-06
 4.3999994e-06 4.7999997e-06 5.2000000e-06 5.5999994e-06 5.9999998e-06
 6.3999996e-06 6.7999995e-06 7.1999998e-06 7.5999997e-06 7.9999954e-06
 7.9999991e-06 8.3999994e-06 8.7999997e-06 9.2000000e-06 9.5999994e-06
 9.9999997e-06 1.1999997e-05 1.6000000e-05 1.9999994e-05 2.3999997e-05
 2.7999999e-05 3.1999996e-05 3.5999998e-05 3.9999999e-05 4.3999997e-05
 4.7999998e-05 5.1999999e-05 5.5999993e-05 5.9999998e-05 6.3999993e-05
 6.7999994e-05 7.1999995e-05 7.5999997e-05 7.9999962e-05 7.9999991e-05
 8.3999999e-05 8.7999993e-05 9.2000002e-05 9.5999989e-05 9.9999997e-05
 1.1999998e-04 1.6000001e-04 1.9999997e-04 2.3999999e-04 2.8000001e-04
 3.1999996e-04 3.6000001e-04 3.9999999e-04 4.3999997e-04 4.7999999e-04
 5.2000000e-04 5.5999996e-04 5.9999997e-04 6.4000004e-04 6.8000000e-04
 7.2000001e-04 7.6000002e-04 7.9999957e-04 8.0000004e-04 8.3999999e-04
 8.8000001e-04 9.2000008e-04 9.6000003e-04 1.0000000e-03 1.1999998e-03
 1.6000000e-03 1.9999996e-03 2.3999996e-03 2.7999999e-03 3.1999995e-03
 3.5999997e-03 3.9999997e-03 4.0000002e-03 4.3999995e-03 4.7999998e-03
 5.2000000e-03 5.5999993e-03 5.9999996e-03 6.3999998e-03 6.7999992e-03
 7.1999999e-03 7.5999997e-03 7.9999967e-03 7.9999985e-03 8.3999997e-03
 8.8000000e-03 9.1999993e-03 9.5999995e-03 9.9999998e-03 1.1999999e-02
 1.6000001e-02 1.9999998e-02 2.3999998e-02 2.8000001e-02 3.1999998e-02
 3.5999998e-02 3.9999999e-02 4.0000003e-02 4.3999996e-02 4.7999997e-02
 5.2000001e-02 5.5999998e-02 5.9999999e-02 6.4000003e-02 6.7999996e-02
 7.1999997e-02 7.6000005e-02 7.9999961e-02 7.9999998e-02 8.3999999e-02
 8.8000000e-02 9.2000000e-02 9.6000001e-02 1.0000000e-01 1.1999998e-01
 1.6000000e-01 1.9999996e-01 2.3999998e-01 2.8000000e-01 3.1999996e-01
 3.5999998e-01 4.0000001e-01 4.3999997e-01 4.7999999e-01 5.1999998e-01
 5.5999994e-01 5.9999996e-01 6.3999999e-01 6.7999995e-01 7.1999997e-01
 7.5999999e-01 7.9999995e-01 8.3999997e-01 8.8000000e-01 9.2000002e-01
 9.5999998e-01 1.0000000e+00]
```

9 Solving PDE

[ ]:
```python
# Main loop find the best learning rate
counter=0
min_index=jnp.inf
min_loss_value = jnp.inf
minimum_loss=[]

# Create a file to save the learning rate
file_data_learn=open('./learning_rate','w')
file_data_learn.close()

# Create a file to save the last value of the loss function
file_data_loss=open('./loss_function','w')
file_data_loss.close()

for i in range(len(learning_rate)):
    loss_history = []
    opt_init, opt_update, get_params = jax_opt.adam(learning_rate[i])

    NN_MLP=MLP(SEED,layers)
    params = NN_MLP.MLP_create()            # Create the MLP
    NN_eval=NN_MLP.NN_evaluation            # Evaluate function
    solver=PINN(NN_eval)                    # Use PINN on the problem 8
    key=NN_MLP.get_key()                    # Get the key of NN

    opt_state = opt_init(params)            # Initialize opt_state

    for ibatch in range(0,num_batches):
        ran_key, batch_key = jran.split(key)
        XY_train = jran.uniform(batch_key, shape=(batch_size, n_features),␣
 ↪minval=0, maxval=1)

        loss, opt_state = solver.train_step(ibatch,opt_state, XY_train)
        loss_history.append(float(loss))

        #if ibatch%report_steps==report_steps-1:
            #print("Epoch n°{}: ".format(ibatch+1), loss.item())

    print("iteration",i+1,"of",len(learning_rate))
    print("loss =",loss_history[num_batches-1],"learning rate␣
 ↪=",learning_rate[i])
    minimum_loss.append(loss_history[num_batches-1])

    # Get the index for the best learning rate
    if loss_history[num_batches-1]<min_loss_value:
        min_loss_value = loss_history[num_batches-1]
```

```
        min_index=i
        print('minimum value =',minimum_loss[i])

    # Save the learning rate
    file_data_learn=open('./learning_rate','a')
    file_data_learn.write(str(learning_rate[i])+',')
    file_data_learn.close()

    # Save the last value of the loss function
    file_data_loss=open('./loss_function','a')
    file_data_loss.write(str(loss_history[num_batches-1])+',')
    file_data_loss.close()
```

```
iteration 1 of 99
loss = 0.0056283138692379 learning rate = 9.599999e-05
minimum value = 0.0056283138692379
iteration 2 of 99
loss = 0.0048086014576256275 learning rate = 1e-04
minimum value = 0.0048086014576256275
iteration 3 of 99
loss = 0.00437261164188385 learning rate = 0.00011999998
minimum value = 0.00437261164188385
iteration 4 of 99
loss = 0.00505151366814971 learning rate = 0.00016000001
iteration 5 of 99
loss = 0.0052724299021065235 learning rate = 0.00019999997
iteration 6 of 99
loss = 0.005549023859202862 learning rate = 0.00024
iteration 7 of 99
loss = 0.005764973349869251 learning rate = 0.00028
iteration 8 of 99
loss = 0.005953838117420673 learning rate = 0.00031999996
iteration 9 of 99
loss = 0.006143741775304079 learning rate = 0.00036
iteration 10 of 99
loss = 0.006300668697804213 learning rate = 0.0004
iteration 11 of 99
loss = 0.0064275446347892284 learning rate = 0.00043999997
iteration 12 of 99
loss = 0.0065496391616761684 learning rate = 0.00048
iteration 13 of 99
loss = 0.006680157035589218 learning rate = 0.00052
iteration 14 of 99
loss = 0.006793366279453039 learning rate = 0.00055999996
iteration 15 of 99
loss = 0.006896598730236292 learning rate = 0.00059999997
iteration 16 of 99
```

```
loss = 0.0069457595236599445 learning rate = 0.00064000004
iteration 17 of 99
loss = 0.007037411909550428 learning rate = 0.00068
iteration 18 of 99
loss = 0.007094191387295723 learning rate = 0.00072
iteration 19 of 99
loss = 0.00715602608397603 learning rate = 0.00076
iteration 20 of 99
loss = 0.007218529935926199 learning rate = 0.0007999996
iteration 21 of 99
loss = 0.007215419318526983 learning rate = 0.00080000004
iteration 22 of 99
loss = 0.007279032841324806 learning rate = 0.00084
iteration 23 of 99
loss = 0.00733697647228837 learning rate = 0.00088
iteration 24 of 99
loss = 0.007450229488313198 learning rate = 0.0009200001
iteration 25 of 99
loss = 0.007429111283272505 learning rate = 0.00096000003
iteration 26 of 99
loss = 0.007506238296627998 learning rate = 0.001
iteration 27 of 99
loss = 0.007746902294456959 learning rate = 0.0011999998
iteration 28 of 99
loss = 0.008289686404168606 learning rate = 0.0016
iteration 29 of 99
loss = 0.008984302170574665 learning rate = 0.0019999996
iteration 30 of 99
loss = 0.009617257863283157 learning rate = 0.0023999996
iteration 31 of 99
loss = 0.01011795923113823 learning rate = 0.0028
iteration 32 of 99
loss = 0.0107788797467947 learning rate = 0.0031999995
iteration 33 of 99
loss = 0.011304347775876522 learning rate = 0.0035999997
iteration 34 of 99
loss = 0.0116304662078619 learning rate = 0.0039999997
iteration 35 of 99
loss = 0.011619005352258682 learning rate = 0.004
iteration 36 of 99
loss = 0.011790497228503227 learning rate = 0.0043999995
iteration 37 of 99
loss = 0.011702973395586014 learning rate = 0.0047999998
iteration 38 of 99
loss = 0.011884546838700771 learning rate = 0.0052
iteration 39 of 99
loss = 0.012837653048336506 learning rate = 0.0055999993
iteration 40 of 99
```

```
loss = 0.012833688408136368 learning rate = 0.0059999996
iteration 41 of 99
loss = 0.013390054926276207 learning rate = 0.0064
iteration 42 of 99
loss = 0.013657047413289547 learning rate = 0.006799999
iteration 43 of 99
loss = 0.013788939453661442 learning rate = 0.0072
iteration 44 of 99
loss = 0.013915683142840862 learning rate = 0.0075999997
iteration 45 of 99
loss = 0.014332626946270466 learning rate = 0.007999997
iteration 46 of 99
loss = 0.014518260024487972 learning rate = 0.0079999985
iteration 47 of 99
loss = 0.014350385405123234 learning rate = 0.0084
iteration 48 of 99
loss = 0.014661336317658424 learning rate = 0.0088
iteration 49 of 99
loss = 0.015009942464530468 learning rate = 0.009199999
iteration 50 of 99
loss = 0.015280765481293201 learning rate = 0.0095999995
iteration 51 of 99
loss = 0.015574362128973007 learning rate = 0.01
iteration 52 of 99
loss = 0.015930278226733208 learning rate = 0.011999999
iteration 53 of 99
loss = 0.018411044031381607 learning rate = 0.016
iteration 54 of 99
loss = 0.02647589147090912 learning rate = 0.019999998
iteration 55 of 99
loss = 0.026135995984077454 learning rate = 0.023999998
iteration 56 of 99
loss = 0.02557310089468956 learning rate = 0.028
iteration 57 of 99
loss = 0.01927211880683899 learning rate = 0.031999998
iteration 58 of 99
loss = 0.025239434093236923 learning rate = 0.036
iteration 59 of 99
loss = 0.042144015431404114 learning rate = 0.04
iteration 60 of 99
loss = 0.03771758824586868 learning rate = 0.040000003
iteration 61 of 99
loss = 0.03965884819626808 learning rate = 0.043999996
iteration 62 of 99
loss = 0.02181864343583584 learning rate = 0.047999997
iteration 63 of 99
loss = 0.04423406347632408 learning rate = 0.052
iteration 64 of 99
```

```
loss = 0.05317981541156769 learning rate = 0.055999998
iteration 65 of 99
loss = 0.042907778173685074 learning rate = 0.06
iteration 66 of 99
loss = 0.05452097952365875 learning rate = 0.064
iteration 67 of 99
loss = 0.04155665636062622 learning rate = 0.067999996
iteration 68 of 99
loss = 0.04972462356090546 learning rate = 0.072
iteration 69 of 99
loss = 0.04367397353053093 learning rate = 0.076000005
iteration 70 of 99
loss = 0.07090074568986893 learning rate = 0.07999996
iteration 71 of 99
loss = 0.042590875178575516 learning rate = 0.08
iteration 72 of 99
loss = 0.08229882270097733 learning rate = 0.084
iteration 73 of 99
loss = 0.06757473945617676 learning rate = 0.088
iteration 74 of 99
loss = 0.09713171422481537 learning rate = 0.092
iteration 75 of 99
loss = 0.07808905094861984 learning rate = 0.096
iteration 76 of 99
loss = 0.08564911037683487 learning rate = 0.1
iteration 77 of 99
loss = 0.11132657527923584 learning rate = 0.11999998
iteration 78 of 99
loss = 0.11153824627399445 learning rate = 0.16
iteration 79 of 99
loss = 0.14020493626594543 learning rate = 0.19999996
iteration 80 of 99
loss = 0.22732415795326233 learning rate = 0.23999998
iteration 81 of 99
loss = 0.11888231337070465 learning rate = 0.28
iteration 82 of 99
loss = 0.12719812989234924 learning rate = 0.31999996
iteration 83 of 99
loss = 0.33327990770339966 learning rate = 0.35999998
iteration 84 of 99
loss = 0.18740606307983398 learning rate = 0.4
iteration 85 of 99
loss = 0.36471953988075256 learning rate = 0.43999997
iteration 86 of 99
loss = 0.7714767456054688 learning rate = 0.48
iteration 87 of 99
loss = 0.6909691095352173 learning rate = 0.52
iteration 88 of 99
```

```
loss = 0.7433273196220398 learning rate = 0.55999994
iteration 89 of 99
loss = 0.439883828163147 learning rate = 0.59999996
iteration 90 of 99
loss = 0.9097825288772583 learning rate = 0.64
iteration 91 of 99
loss = 0.6054460406303406 learning rate = 0.67999995
iteration 92 of 99
loss = 0.6783227920532227 learning rate = 0.71999997
iteration 93 of 99
loss = 0.6899469494819641 learning rate = 0.76
iteration 94 of 99
loss = 2.1728146076202393 learning rate = 0.79999995
iteration 95 of 99
loss = 1.0038481950759888 learning rate = 0.84
iteration 96 of 99
loss = 1.763105869293213 learning rate = 0.88
iteration 97 of 99
loss = 2.2033491134643555 learning rate = 0.92
iteration 98 of 99
loss = 2.211721897125244 learning rate = 0.96
iteration 99 of 99
loss = 3.3589024543762207 learning rate = 1.0
```

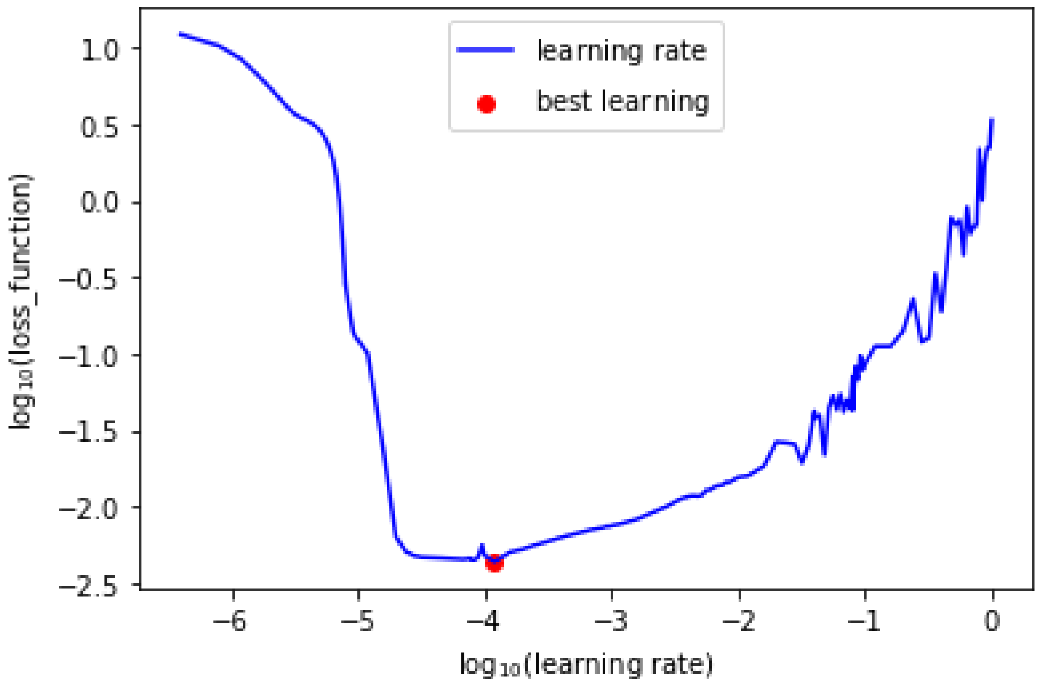
10 Plot learning rate optimization

[ ]:
```
min_index=50
```

```
learning_rate=[4e-07,7.999996e-07,1.1999998e-06,1.6e-06,1.9999995e-06,2.
↪3999999e-06,2.8e-06,3.1999996e-06,3.5999997e-06,4e-06,4.3999994e-06,4.
↪7999997e-06,5.2e-06,5.5999994e-06,5.9999998e-06,6.3999996e-06,6.
↪7999995e-06,7.2e-06,7.5999997e-06,7.999995e-06,7.999999e-06,8.399999e-06,8.
↪8e-06,9.2e-06,9.599999e-06,1e-05,1.1999997e-05,1.6e-05,1.9999994e-05,2.
↪3999997e-05,2.7999999e-05,3.1999996e-05,3.5999998e-05,4e-05,4.3999997e-05,4.
↪7999998e-05,5.2e-05,5.5999993e-05,6e-05,6.399999e-05,6.7999994e-05,7.
↪1999995e-05,7.6e-05,7.999996e-05,7.999999e-05,8.4e-05,8.799999e-05,9.2e-05,9.
↪599999e-05,1e-04,0.00011999998,0.00016000001,0.00019999997,0.00024,0.00028,0.
↪00031999996,0.00036,0.0004,0.00043999997,0.00048,0.00052,0.00055999996,0.
↪00059999997,0.00064000004,0.00068,0.00072,0.00076,0.0007999996,0.
↪00080000004,0.00084,0.00088,0.0009200001,0.00096000003,0.001,0.0011999998,0.
↪0016,0.0019999996,0.0023999996,0.0028,0.0031999995,0.0035999997,0.
↪0039999997,0.004,0.0043999995,0.0047999998,0.0052,0.0055999993,0.
↪0059999996,0.0064,0.006799999,0.0072,0.0075999997,0.007999997,0.0079999985,0.
↪0084,0.0088,0.009199999,0.0095999995,0.01,0.011999999,0.016,0.019999998,0.
↪023999998,0.028,0.031999998,0.036,0.04,0.040000003,0.043999996,0.047999997,0.
↪052,0.055999998,0.06,0.064,0.067999996,0.072,0.076000005,0.07999996,0.08,0.
↪084,0.088,0.092,0.096,0.1,0.11999998,0.16,0.19999996,0.23999998,0.28,0.
↪31999996,0.35999998,0.4,0.43999997,0.48,0.52,0.55999994,0.59999996,0.64,0.
↪67999995,0.71999997,0.76,0.79999995,0.84,0.88,0.92,0.96,1.0]
```

```
minimum_loss = [12.199068069458008,10.287569999694824,8.428057670593262,6.
↪696137428283691,5.538702964782715,4.687373161315918,4.059501647949219,3.
↪656656265258789,3.457155704498291,3.325516700744629,3.174725294113159,2.
↪993105411529541,2.772662401199341,2.507611036300659,2.195760726928711,1.
↪8345900774002075,1.426107406616211,0.9818812012672424,0.5550472140312195,0.
↪2830400764942169,0.2830425798892975,0.21830421686172485,0.
↪17070499062538147,0.13883444666862488,0.12758783996105194,0.
↪12315638363361359,0.1000934392213821 4,0.022998584434390068,0.
↪006351709831506014,0.005078324116766453,0.004771950654685497,0.
↪004673386458307505,0.004661232698708773,0.004636757075786591,0.
↪004623145796358585,0.004604278597980738,0.0045907096937298775,0.
↪004563584458082914,0.004564367700368166,0.004526973236352205,0.
↪004533765837550163,0.004557443782687187,0.004603053443133831,0.
↪00447818823158741,0.0044846506789326667,0.00450295116752386 1,0.
↪004616164602339268,0.00500076450407505,0.0056283138692379,0.
↪0048086014576256275,0.004372611641883 85,0.005051513668 14971,0.
↪005272429902106 5235,0.0055490238592028 62,0.0057649733498692 51,0.
↪005953838117420673,0.006143741775304079,0.006300668697804213,0.
↪0064275446347892284,0.0065496391616761684,0.0066801570355892 18,0.
↪006793366279453039,0.006896598730236292,0.0069457595236599445,0.
↪007037411909550428,0.007094191387295723,0.00715602608397603,0.
↪007218529935926199,0.007215419318526983,0.007279032841324806,0.
↪00733697647228837,0.007450229488313198,0.00742911128327250 5,0.
↪007506238296627998,0.007746902294456959,0.008289686404168606,0.
↪008984302170574665,0.009617257863283157,0.01011795923113823,0.
↪0107788797467947,0.011304347775876522,0.0116304662078619,0.
↪011619005352258682,0.011790497228503227,0.011702973395586014,0.
↪011884546838700771,0.012837653048336506,0.012833688408136368,0.
↪013390054926276207,0.013657047413289547,0.013788939453661442,0.
↪013915683142840862,0.014332626946270466,0.014518260024487972,0.
↪014350385405123234,0.014661336317658424,0.015009942464530468,0.
↪015280765481293201,0.015574362128973007,0.015930278226733208,0.
↪018411044031381607,0.02647589147090912,0.026135995984077454,0.
↪02557310089468956,0.01927211880683899,0.025239434093236923,0.
↪042144015431404114,0.03771758824586868,0.03965884819626808,0.
↪0218186434358 3584,0.04423406347632408,0.05317981541156769,0.
↪042907778173685074,0.05452097952365875,0.04155665636062622,0.
↪04972462356090546,0.04367397353053093,0.07090074568986893,0.
↪042590875178575516,0.08229882270097733,0.06757473945617676,0.
↪09713171422481537,0.07808905094861984,0.08564911037683487,0.
↪11132657527923584,0.11153824627399445,0.14020493626594543,0.
↪22732415795326233,0.11888231337070465,0.12719812989234924,0.
↪33327990770339966,0.18740606307983398,0.36471953988075256,0.
↪7714767456054688,0.6909691095352173,0.7433273196220398,0.439883828163147,0.
↪9097825288772583,0.6054460406303406,0.6783227920532227,0.6899469494819641,2.
↪1728146076202393,1.0038481950759888,1.763105869293213,2.2033491134643555,2.
↪211721897125244,3.3589024543762207]
```

```python
fig, ax = plt.subplots(1, 1)
__=ax.plot(np.log10(learning_rate),np.log10(minimum_loss),color='blue')
__=ax.scatter(np.log10(learning_rate[min_index]),np.
 ↪log10(minimum_loss[min_index]),color='red')
legend = ax.legend([r'${\rm learning \ rate}$',r'${\rm best \ learning }$'])
xlabel = ax.set_xlabel(r'$\log_{10}{\rm (learning \ rate)}$')
ylabel = ax.set_ylabel(r'$\log_{10}{\rm (loss\_function)}$')
#title = ax.set_title(r'${\rm Learning \ rate \ optimization}$')
plt.show()
print("best learning rate =",learning_rate[min_index],"\nloss␣
 ↪=",minimum_loss[min_index])
```




```
best learning rate = 0.00011999998
loss = 0.00437261164188385
```