```
#@markdown 引入必要的库
import torch
import itertools
import numpy as np
import torch.nn as nn
from tqdm import tqdm
import torch.optim as optim
from torch.nn import Parameter
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
```

引入必要的库

```
#@markdown 一种自己生成训练集的Dataset
# 辅助函数，用于生成训练样例的最优解
# 来自 https://gist.github.com/mlalevic/6222750
def tsp_opt(points):
    """
    O(2^n*n^2)
    :param points: List of (x, y) points
    :return: Optimal solution
    """

    def length(x_coord, y_coord):
        return np.linalg.norm(np.asarray(x_coord) - np.asarray(y_coord))

    # Calculate all lengths
    all_distances = [[length(x, y) for y in points] for x in points]
    # Initial value - just distance from 0 to every other point + keep the tr
    A = {(frozenset([0, idx+1]), idx+1): (dist, [0, idx+1]) for idx, dist in
    cnt = len(points)
    for m in range(2, cnt):
        B = {}
        for S in [frozenset(C) | {0} for C in itertools.combinations(range(1,
            for j in S - {0}:
                # This will use 0th index of tuple for ordering, the same as
                B[(S, j)] = min([(A[(S-{j}, k)][0] + all_distances[k][j], A[(
                                 for k in S if k != 0 and k != j])
        A = B
    res = min([(A[d][0] + all_distances[0][d[1]], A[d][1]) for d in iter(A)])
    return np.asarray(res[1])
# 生成训练集的DATASET
class TSP_Dataset(Dataset):
    def __init__(self, data_size, citys_size):
        """
        data_size: int, 训练集大小
        citys_size: int, 每个训练数据包含多少城市,
```

一种自己生成训练集的Dataset

```
        """
        super(TSP_Dataset, self).__init__()
        self.data_size = data_size
        self.citys_size = citys_size
        self.data = self.get_data()

    def __len__(self):
        return self.data_size

    def __getitem__(self, idx):
        return {'Point': self.data['Point'][idx], 'Slove':self.data['Slove']|

    def get_data(self):
        """
        随生成数据
        """
        point = np.random.rand(self.data_size, self.citys_size, 2)
        slove = [tsp_opt(citys) for citys in point]
        return dict(Point = point, Slove = slove)
```

```
#@markdown 使用论文提供的数据集的Dataset
# 尝试用论文里的使用数据集
class TSP_Dataset(Dataset):
    def __init__(self, citys_size, op='train'):
        """
        data_size: int, 训练集大小
        citys_size: int, 每个训练数据包含多少城市,
        op: str, 训练集还是测试集
        """
        super(TSP_Dataset, self).__init__()
        self.citys_size = citys_size
        if op == 'train':
            data = np.loadtxt(f'/content/drive/MyDrive/DATA/tsp_{citys_size}_
                              delimiter=' ',
                              usecols=tuple([i for i in range(citys_size * 3 +
        else:
            data = np.loadtxt(f'/content/drive/MyDrive/DATA/tsp_{citys_size}_
                              delimiter=' ',
                              usecols=tuple([i for i in range(citys_size * 3 +
        point = data[:, :citys_size * 2].reshape(-1, citys_size, 2)
        solve = data[:, citys_size * 2:].reshape(-1, citys_size)
        self.data = dict(Point = point, Slove = solve)

    def __len__(self):
        return self.data['Slove'].shape[0]

    def __getitem__(self, idx):
        return {'Point': self.data['Point'][idx], 'Slove': self.data['Slove']
```

使用论文提供的数据集的Dataset

```
#@markdown 定义模型
# Encoder Model
class Encoder(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, num_layers, dropout, bidire
        super(Encoder, self).__init__()

        # 用于接下来初始化h0, c0
        self.num_layers = num_layers*2 if bidirectional else num_layers

        #为了维持不管是否BRNN，输出的hidden_size 都等于hidden_dim, 这里的self.hidde
        self.hidden_dim = hidden_dim//2 if bidirectional else hidden_dim

        self.lstm = nn.LSTM(
            input_size = embedding_dim,
            hidden_size = self.hidden_dim,
            num_layers = num_layers,
            batch_first = True,
```

定义模型

```
            dropout = dropout,
            bidirectional = bidirectional)

        self.h0 = Parameter(torch.zeros(1), requires_grad=False)
        self.c0 = Parameter(torch.zeros(1), requires_grad=False)

    def forward(self, embedded_inputs):

        batch_size = embedded_inputs.size(0)
        h_0 = self.h0.unsqueeze(0).unsqueeze(0).repeat(self.num_layers,
                                                       batch_size,
                                                       self.hidden_dim)
        c_0 = self.c0.unsqueeze(0).unsqueeze(0).repeat(self.num_layers,
                                                       batch_size,
                                                       self.hidden_dim)

        output, (h_n, c_n) = self.lstm(embedded_inputs, (h_0, c_0))
        return output, (h_n, c_n)
# Attention Layer
class Attention(nn.Module):
    def __init__(self, hidden_dim):
        super(Attention, self).__init__()

        self.h_t_linear = nn.Linear(hidden_dim, hidden_dim)
        self.h_s_linear = nn.Conv1d(hidden_dim, hidden_dim, 1, 1)

        # F.tanh 被 deprecated 了，要用nn.Tanh
        # https://discuss.pytorch.org/t/torch-tanh-vs-torch-nn-functional-tar
        # https://stackoverflow.com/questions/56723486/why-arent-torch-functi
        self.tanh = nn.Tanh()
        self.softmax = nn.Softmax()

        self.V = Parameter(torch.FloatTensor(hidden_dim), requires_grad=True)
        # 不要用uniform_，那是就地操作
        nn.init.uniform(self.V, -1, 1)
        # 不要用alpha[visted] = float('-inf'),好像也是就地操作?
        # self.inf = self._inf.unsqueeze(1).expand(*mask_size) 是为了避免就地操
        self._inf = Parameter(torch.FloatTensor([float('-inf')]), requires_gr

    def forward(self, h_s, h_t, visted):
        """
        h_s = encoder_output  (N, L, Hidden)
        h_t = h_t   (N, Hidden)
        visted :访问过的城市TrueFalse Tensor (N, L)就是N个L
        """
        # score = V * tanh(W[h_s, h_t])

        #(N, Hidden, L)
```

```
        h_s_ = self.h_s_linear(h_s.permute(0, 2, 1))
        #(N, Hidden, L)
        h_t_ = self.h_t_linear(h_t).unsqueeze(2).expand(-1, -1, h_s_.size(2))

        # V 需要是（N, 1, Hidden） 乘出来是（N, 1, L）, 再经过softmax就是概率
        V = self.V.unsqueeze(0).unsqueeze(0).expand(h_s_.size(0), -1, -1)
        # (N, L)
        alpha = torch.bmm(V, self.tanh(h_s_ + h_t_)).squeeze(1)

        # 将访问过的City alpha 变为-inf再softmax
        self.inf = self._inf.unsqueeze(1).expand(alpha.size(0), alpha.size(1))
        # 不能  alpha[visted] = float('-inf')  谨防就地操作
        if len(alpha[visted]) > 0:
            alpha[visted] = self.inf[visted]

        alpha_ = self.softmax(alpha)
        # c_t = h_s * alpha_ (N, Hidden)
        # 找到了，是attention层的内容
        c_t = torch.bmm(h_s_, alpha_.unsqueeze(2)).squeeze(2)
        return c_t, alpha_
# Decoder Model

class Decoder(nn.Module):
    def __init__(self, embedding_dim, hidden_dim):
        super(Decoder, self).__init__()

        self.embedding_dim = embedding_dim
        self.input0 = Parameter(torch.FloatTensor(embedding_dim), requires_gr
        # decoder的input0初始化为[-1, 1]随机数 (batch, embedding_dim)
        nn.init.uniform_(self.input0, -1, 1)

        self.i2h = nn.Linear(embedding_dim, 4*hidden_dim)
        self.h2h = nn.Linear(hidden_dim, 4*hidden_dim)

        self.att = Attention(hidden_dim)

        # 为 0（False）代表没被访问过
        self.visted = Parameter(torch.zeros(1), requires_grad=False)

        # h~t = tanh(Wc[c_t; h_t])
        self.Wc = nn.Linear(2*hidden_dim, hidden_dim)

        # 辅助判断是否visted
        self.help_visted = Parameter(torch.zeros(1), requires_grad=False)

        self.sigmoid =  nn.Sigmoid()
        self.tanh =  nn.Tanh()

```

```
114
115     def forward(self, embedded_inputs, decoder_h_0, decoder_c_0, encoder_outp
116         batch_size = embedded_inputs.size(0)
117         input_length = embedded_inputs.size(1)
118 
119         # (N, embedding)
120         decoder_input = self.input0.unsqueeze(0).expand(batch_size, -1)
121 
122         # 初始 h, c 为encoder最后一个状态
123         decoder_h = decoder_h_0
124         decoder_c = decoder_c_0
125 
126         # (N, L)
127         visted = self.visted.unsqueeze(0).expand(batch_size, input_length)
128 
129         # (N, L)
130         help_visted = self.help_visted.repeat(input_length)
131         for i in range(input_length):
132             help_visted.data[i] = i
133         help_visted = help_visted.unsqueeze(0).expand(batch_size, -1).long()
134 
135 
136         # 最后要返回城市序列 pointers 和 每一级的output = alpha
137         outputs = []
138         pointers = []
139 
140         def step(x, h, c):
141             """
142             模拟一个lstm cell，然后增加一个layer层
143             x: (N, embeding_dim)
144             h: (N, Hidden)
145             c: (N, Hidden)
146             """
147             # (N, 4 * Hidden)
148             gates = self.i2h(x) + self.h2h(h)
149             # (N, Hidden)
150             input, forget, cell, out = gates.chunk(4,1)
151 
152             forget = self.sigmoid(forget)
153             input = self.sigmoid(input)
154             cell = self.tanh(cell)
155             out = self.sigmoid(out)
156 
157             c_t = (c * forget) + (input * cell)
158             # (N, Hidden)
159             h_t = self.tanh(c_t) * out
160 
161             # h_s = encoder_output, h_t, visited 进入 attn 层， 获得c_t(N, Hidd
```

```
            c_t, alpha = self.att(encoder_output, h_t, torch.eq(visted, 1))
            hidden_t = self.tanh(self.Wc(torch.cat((c_t, h_t), 1)))

            return hidden_t, c_t, alpha

        for _ in range(input_length):
            decoder_h, decoder_c, alpha = step(decoder_input, decoder_h, decc
            # 将访问过的alpha 置 0,然后求最大的alpha的坐标 indices(N)
            # 不能alpha[torch.eq(visted, 1)] = 0，谨防就地操作
            alpha_ = alpha * (1 - visted)
            val, indices = alpha_.max(1)
            # alpha_ (N, L)
            # 求得该次应该访问的城市，用indices(N),help_visted(N, L)更新visted (N,
            # tmp (N, L)
            tmp  = (help_visted == (indices.unsqueeze(1).expand(-1, input_len
            visted = 1 -  (1 - visted) * (1 - tmp.float())

            # 通过indeices(N), embedding_inputs(N, L, embedding_dim) 获得下一次
            decoder_input = embedded_inputs[tmp.unsqueeze(2).expand(-1, -1, s

            # 为了cat合并出想要的效果，这里需要做一些处理
            outputs.append(alpha_.unsqueeze(0))
            pointers.append(indices.unsqueeze(1))

        # (N, L, L（概率数组）) 表示每次的输出的alpha数组(L)， 每个aplha代表该次预测的
        outputs = torch.cat(outputs).permute(1, 0, 2)
        # (N, L) 表示一个最优解
        pointers = torch.cat(pointers, 1)

        return (outputs, pointers), (decoder_h, decoder_c)

class PtrNet(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, num_layers, dropout, bidire
        """
        hidden_dim : encoder与decoder共用，隐状态feature数
        num_layers : encoder使用，拥用于init LSTM，代表堆叠层数
        dropout : encoder使用，拥用于init LSTM，代表丢弃数
        bidirectional : encoder使用，拥用于init LSTM，代表是否双向输入
        """
        super(PtrNet, self).__init__()
        self.bidir = bidirectional
        self.embedding = nn.Linear(2, embedding_dim)
        self.encoder = Encoder(embedding_dim, hidden_dim, num_layers, dropout
        self.decoder = Decoder(embedding_dim, hidden_dim)

    def forward(self, inputs):
        """
        inputs：(batch_size, seq_length, 2) 全部城市的坐标
```

```
        """
        batch_size = inputs.size(0)
        seq_length = inputs.size(1)

        # 对输入的数据进行embedding编码
        # (batch_size, seq_length, embedding_dim)
        embedded_inputs = self.embedding(  inputs.view(batch_size * seq_lengt

        #将embedded_inputs送入 encoder 中
        encoder_output, (encoder_h_n, encoder_c_n) = self.encoder(embedded_ir

        # decoder 需要的有:
        # encoder_output, (所有encoder的隐状态，用于传入attention)
        # decoder_h_0 = encoder_h_n, decoder_c_0 = encoder_c_n,
        # seq_length循环次数,
        # embedded_inputs, 作为每次选出的城市，输入下一次循环中

        # embedded_inputs, encoder_h_n, encoder_c_n, encoder_output, seq_leng
        if self.bidir:
            decoder_h = torch.cat( (encoder_h_n[-1],encoder_h_n[-2]), dim=-1)
            decoder_c = torch.cat( (encoder_c_n[-1],encoder_c_n[-2]), dim=-1)
        else:
            decoder_h = encoder_h_n[-1]
            decoder_c = encoder_c_n[-1]

        (outputs, pointers), (decoder_h, decoder_c) = self.decoder(embedded_i
                                                                   decoder_h,
                                                                   decoder_c,
                                                                   encoder_output)
        return  outputs, pointers

```

```
train_size = None # 训练集大小,每次epoch进行train_size个数据的训练
seq_length = 10  # 单个训练数据的城市数目

# myDataset = TSP_Dataset(train_size , seq_length)
myDataset = TSP_Dataset(seq_length)
train_size = myDataset.__len__()
```

```
batch_size = 512  # batch大小，每epoch进行 train_size/batch_size 次batch
myDataloader = DataLoader(dataset = myDataset, batch_size = batch_size, shuf
```

```
embedding_dim = 128
hidden_dim = 512
num_layers = 4
dropout = 0.001
bidirectional = True
mymodel = PtrNet(embedding_dim, hidden_dim, num_layers, dropout, bidirectiona
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: UserWarni
```

```
lr = 0.0005
CCE = torch.nn.CrossEntropyLoss()
mymodel_optim = optim.Adam(filter(lambda p: p.requires_grad, mymodel.paramete
```

```
#@markdown 辅助函数，用于根据提供的旅游路线计算总路线长度
#@markdown 计算最优解的TSP-length及ptrnet的TSP-length
# 计算 TSP 长度，在结果中显示

def fun(points, target, predict):
    """
    points: [N, seq_length, 2]
    target: [N, seq_length]
    predict: [N, seq_length]
    return:
    op_tsp.avg
    pre_tsp.avg
    """
    def dis(x , y):
            return np.linalg.norm(np.asarray(x) - np.asar
    def cal(p, vis):
        return np.sum([ dis(p[vis[i]], p[vis[(i + 1)%vis.
    op_tsp = [cal(p, i) for p, i in zip(points,target)]
    pre_tsp = [cal(p, i) for p, i in zip(points,predict)]
    tsp_loss = dis(op_tsp , pre_tsp) / target.size(0)
    return np.average(op_tsp), np.average(pre_tsp), tsp_l
```

辅助函数，用于根据提供的旅游路线计算总路线长度

计算最优解的TSP-length及ptrnet的TSP-length

```
# 开始训练

num_epochs = 1  # epoch数
losses = []
writer = SummaryWriter()

for epoch in range(num_epochs):
    batch_loss = []
```

```
    batch_op_tsp = []
    batch_pre_tsp = []
    batch_tsp_loss = []
    iterator = tqdm(myDataloader, unit='Batch')
    for i, item in enumerate(iterator):
        iterator.set_description('Epoch %i/%i' % (epoch+1, num_epochs))

        # torch.autograd.set_detect_anomaly(True)
        outputs, pointers = mymodel(item['Point'].float())
        # outputs (N, L, L)
        # item['Slove'] (N, L)

        op_tsp, pre_tsp, tsp_loss = fun(item['Point'], item['Slove'].long() -

        # (N*L, L)
        outputs  = outputs.contiguous().view(-1, outputs.size()[-1])
        # (N*L)
        target = item['Slove'].view(-1).long() - 1
        loss = CCE(outputs, target)


        mymodel_optim.zero_grad()
        loss.backward()
        mymodel_optim.step()

        losses.append(loss.data.item())
        batch_loss.append(loss.data.item())
        batch_op_tsp.append(op_tsp)
        batch_pre_tsp.append(pre_tsp)
        batch_tsp_loss.append(tsp_loss)

        iterator.set_postfix( {'loss' : loss.data.item(),
                               'op_tsp' : op_tsp,
                               'pre_tsp' : pre_tsp,
                               'tsp_loss' : tsp_loss })
        writer.add_scalar('Loss/train', tsp_loss, i)

    iterator.set_postfix({
        'loss' : np.average(batch_loss),
        'op_tsp' : np.average(batch_op_tsp),
        'pre_tsp' :np.average(batch_pre_tsp),
        'tsp_loss' : np.average(batch_tsp_loss)})


```

```
Epoch 1/1:   0%|          | 0/1954 [00:00<?, ?Batch/s]/usr/local/lib/pytho
Epoch 1/1: 100%|██████████| 1954/1954 [4:28:01<00:00,  8.23s/Batch, loss=2
```

```
writer.close()
torch.save(mymodel.state_dict(), '/content/drive/MyDrive/MODEL/ptrnet10.pth')
```

```
# 测试


mymodel.load_state_dict(torch.load('/content/drive/MyDrive/MODEL/ptrnet10.pth

writer = SummaryWriter()

TestDataset = TSP_Dataset(10, 'test')

test_batch_loss = []
test_batch_op_tsp = []
test_batch_pre_tsp = []
test_batch_tsp_loss = []
iterator = tqdm(zip(TestDataset.data['Point'], TestDataset.data['Slove']))

with torch.no_grad():
    for i, (points, solve) in  enumerate(iterator):
        points = torch.tensor(points).unsqueeze(0)
        solve = torch.tensor(solve).unsqueeze(0)
        outputs, pointers = mymodel(points.float())
        # outputs (N, L, L)
        # item['Slove'] (N, L)

        op_tsp, pre_tsp, tsp_loss = fun(points, solve.long() - 1, pointers)

        # (N*L, L)
        outputs  = outputs.contiguous().view(-1, outputs.size()[-1])
        # (N*L)
        target = solve.view(-1).long() - 1
        loss = CCE(outputs, target)

        test_batch_loss.append(loss.data.item())
        test_batch_op_tsp.append(op_tsp)
        test_batch_pre_tsp.append(pre_tsp)
        test_batch_tsp_loss.append(tsp_loss)

        iterator.set_postfix( {'loss' : loss.data.item(),
                               'op_tsp' : op_tsp,
                               'pre_tsp' : pre_tsp,
                               'tsp_loss' : tsp_loss })
        writer.add_scalar('Loss/test', tsp_loss, i)

    iterator.set_postfix({
```

```
        'loss' : np.average(test_batch_loss),
        'op_tsp' : np.average(test_batch_op_tsp),
        'pre_tsp' :np.average(test_batch_pre_tsp),
        'tsp_loss' : np.average(test_batch_tsp_loss)})

```

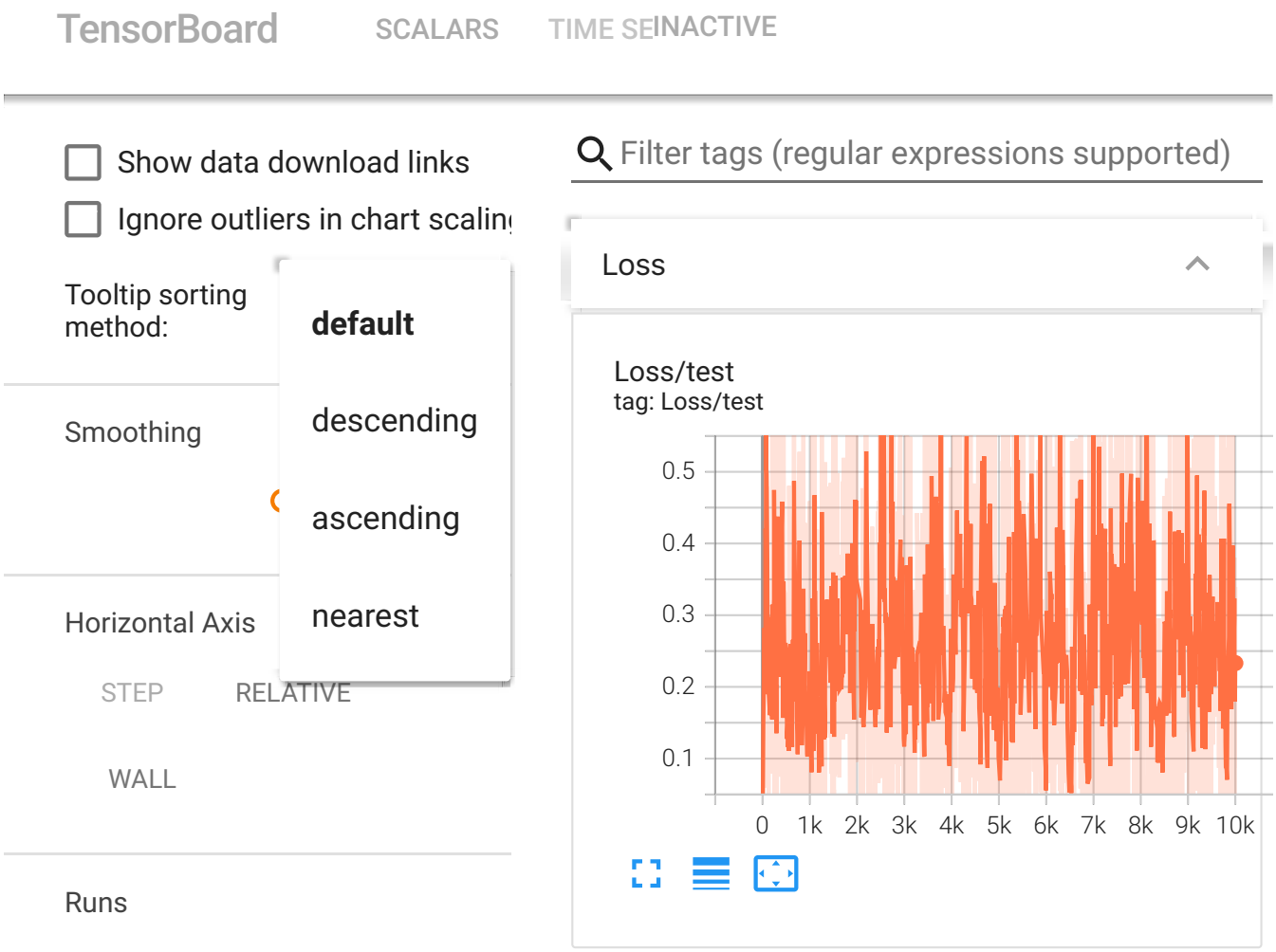
```
0it [00:00, ?it/s]/usr/local/lib/python3.7/dist-packages/ipykernel_launche
10000it [06:03, 27.47it/s, loss=2.32, op_tsp=2.88, pre_tsp=3.22, tsp_loss=
```

```
writer.close()
```

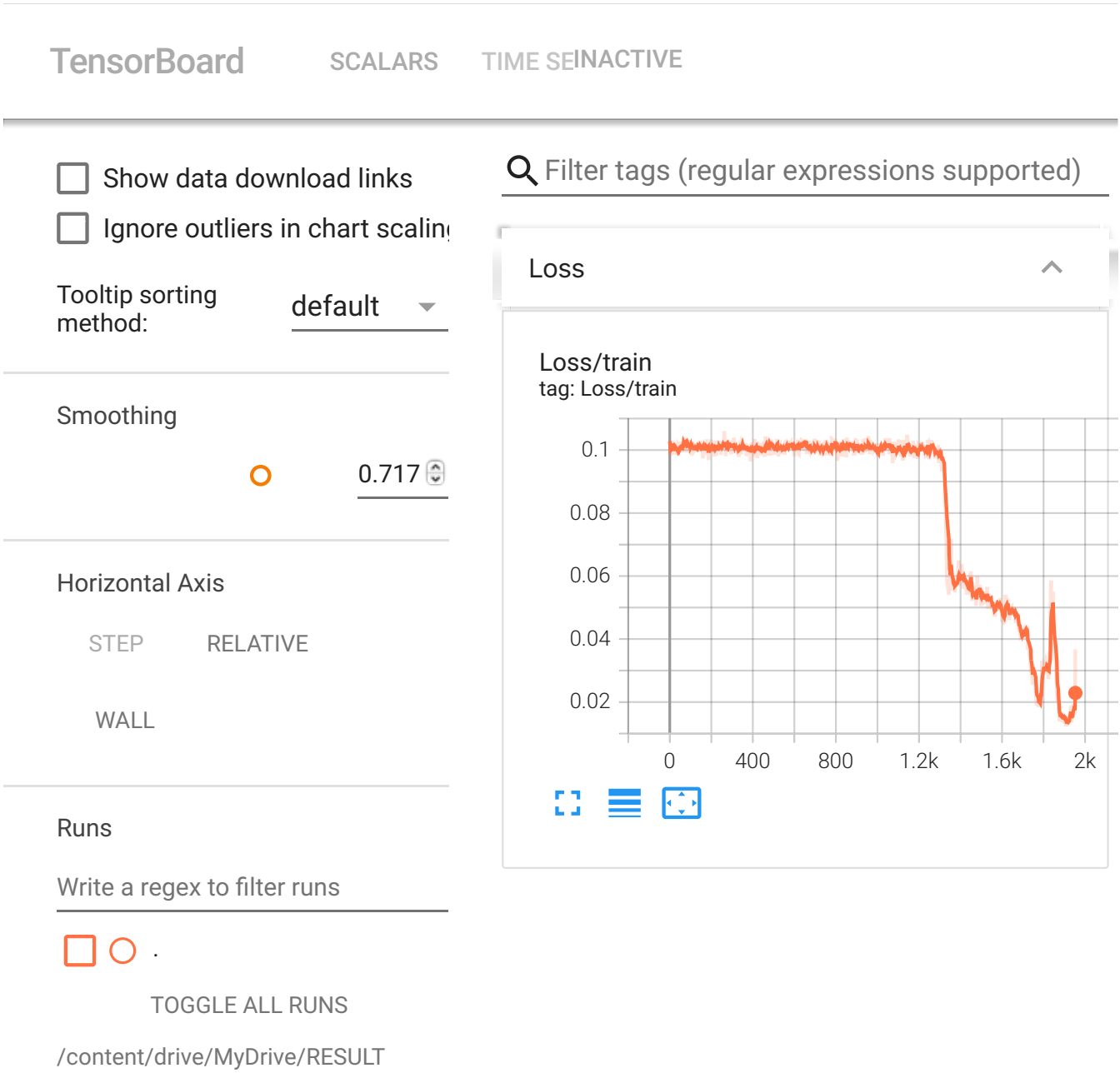
```
%load_ext tensorboard
```

```
%tensorboard --logdir '/content/runs/Nov05_02-02-26_a920a9a57765'
# tsp 10 test
```

```
Reusing TensorBoard on port 6011 (pid 720), started 0:02:31 ago. (Use '!kil
```



```
%tensorboard --logdir '/content/drive/MyDrive/RESULT/ptrnet10.pth'
# tsp 10 train
```



```
%tensorboard --logdir '/content/runs/Nov04_04-45-42_194b542fa822'
# tsp 5 test&train
```

```
1
```

```
1
```

0 秒 完成时间：上午10:16