实验四：Pytorch 实现玉米基因表达量预测模型

Hu Lab

2025 年 4 月 10 日

1. 实验背景及数据

1.1 实验背景

基因表达是生物学中的核心问题之一。基因在表达过程中，先转录成 mRNA，再翻译成蛋白质，这里基因转录出 mRNA 的量称为基因表达量。基因的转录受多种因素的影响，称为转录调控，包括顺式调控，反式调控及环境的影响等等。反式调控是指细胞中对基因表达有影响的蛋白质对基因的调控，顺式调控是指与基因在同一条 DNA 链上的 DNA 序列对基因的调控。顺式调控是影响基因表达的主要因素，也是我们的研究对象。这些调控序列 (称为调控元件) 根据其位置及功能的不同，可以分为启动子，增强子，沉默子等。启动子是转录起始位点 (TSS) 上游约 1kb 的 DNA 序列，其具有起始转录的能力，增强子可能位于基因上游远端，基因下游或者内含子区，对基因的表达起增强作用，沉默子对基因表达起抑制作用。另外，不同的调控元件有其特有的序列特征，这是 DNA 序列可以用来深度学习建模的基础。

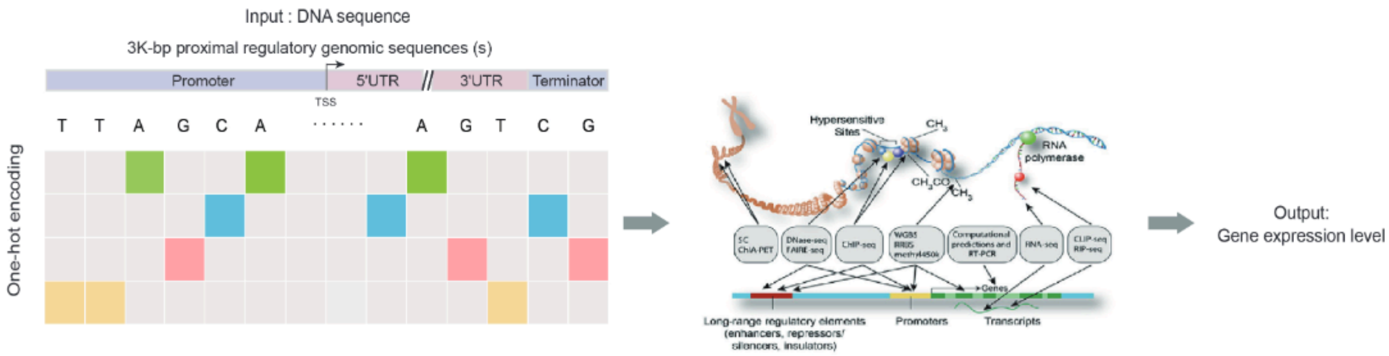


图 1: 实验背景

1.2 实验数据

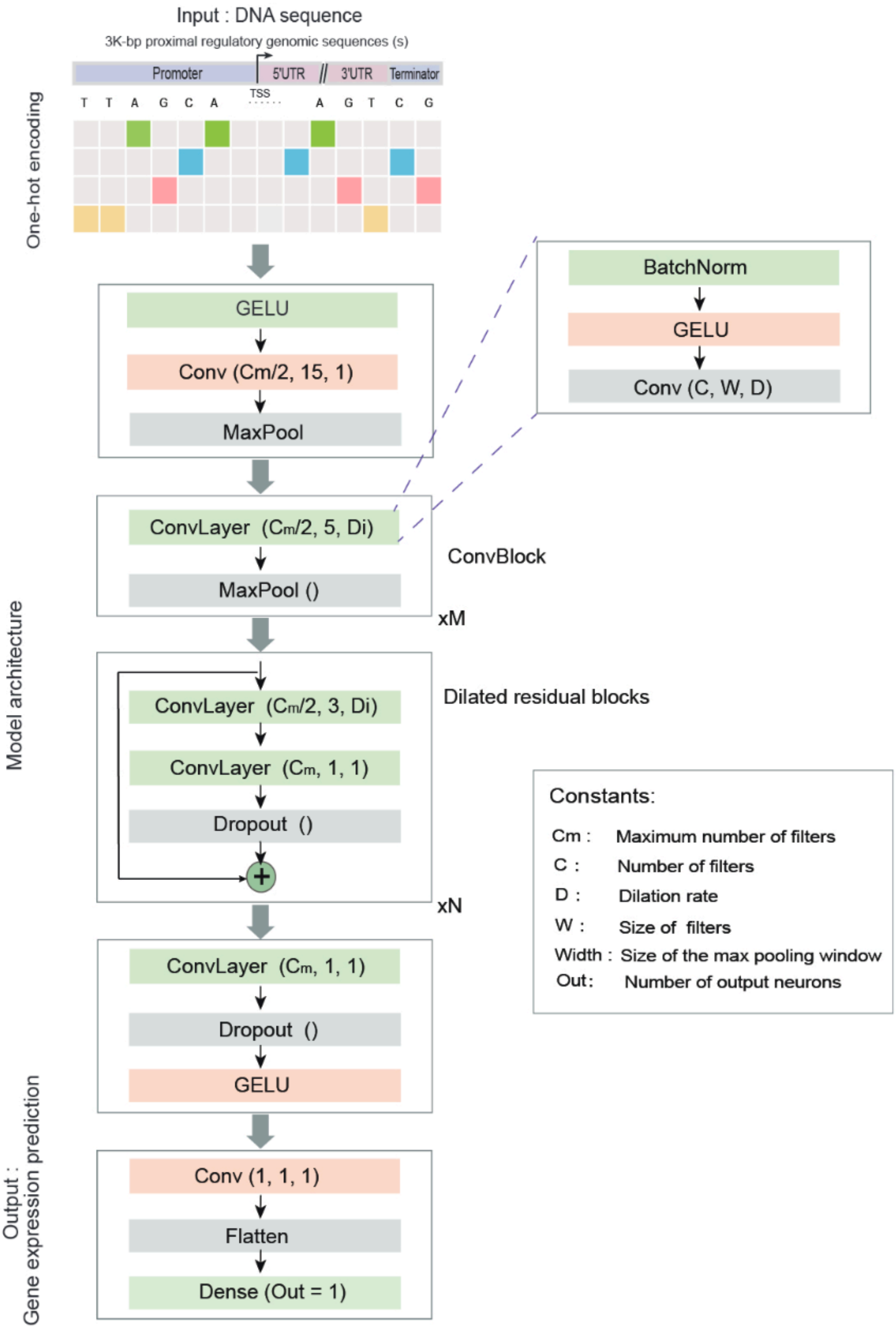
本实验以玉米基因 TSS 附近上下游的四个调控区共 3kb 的 DNA 序列的 DNA 序列作经过 one-hot 编码为输入，one-hot 编码是将每种碱基 (有 A、T、C 和 G 四种) 映射成一个四维向量，其中一个维度是 1，剩余三个维度是 0。本实验中使用的标签基因表达量，是通过 RNA-seq 实验数据处理得到。output(标签) 为每个基因通过实验测得的表达量，单位为 TPM，代表着基因转录出 mRNA 的量。我们使用的基因数为 37979 个，本实验将数据划分为训练集和测试集，分别占 80% 和 20%，每一个样本对应一个基因。本实验拟通过搭建深度残差卷积网络，来实现对玉米基因表达量的预测。

2. 模型介绍

用 python 的 Pytorch 模块实现空洞残差卷积网络。网络结构包含 Convlution block、Convlution tower 和 Dilation residual block 及一个全连接层。此模型结构 (Basenji) 来源于文献：
https://genome.cshlp.org/content/28/5/739.full?sid=3ecf5a07-edd6-4fe8-b183-10b7d1e269f4

Geneid	sequence	TPM	dataset
Zm00001d027230	TGGAGGAAGCGAGCACATGTCGGGCGG	34.621006	train
Zm00001d027235	TTTGCTCCTGCCATTCCCCAAAGCTTA	1.673876	valid
Zm00001d027250	TAATGTGTCCAAAAAACGGTTCCTCTA	4.509122	train
Zm00001d027253	AAATAGACACAAAAGCAAGTCTACTTG	5.616972	train
Zm00001d027254	TCAGCTTCAGTCTAGTCTCTTCTTGTG	3.272217	valid
Zm00001d027256	TAGGAAATTAACCCACCGTCCCTGGTT	12.515516	valid
Zm00001d027258	CTTTAAAAAGTGGGGAATTTTATGCTA	291.908051	train
Zm00001d027265	GTTCTCCTTTGAGGGCGGAGGTACTAG	27.537571	test
Zm00001d027266	GCGGCTTCGAGCACTGCACCGGCAAGC	152.018448	valid
Zm00001d027267	TCATCATCTGCTGTCAAAATTTGGTGG	140.49733	train
Zm00001d027269	GAACCGGTATGGAATCTTAGCCGTTGG	0.882457	train
Zm00001d027273	TCTTACCGAATGGGTGGGACTCGTCTC	5.569419	train
Zm00001d027275	TCAGGTCGGTCATCACCTAAACCACCT	6.147862	train
Zm00001d027278	TTGGGTAAACATTTATATAAGTTGTAT	93.774559	train
Zm00001d027279	GAAATGTGAAGTTGTGACTAAATCATC	2.54835	train
Zm00001d027283	GATGATCCAGACTTACCTTTATGAGTG	196.878952	train
Zm00001d027285	TTATCTAAAAAGGTTAAAAAGTCTTAT	6.53042	test
Zm00001d027286	CGGTCGCGCCCTACTGGGCCTAGGAGC	10.948885	valid
Zm00001d027289	TTGAATGCCAGGTCATGAAAAAAAAGT	2.951355	test
Zm00001d027290	GCAAGTGTCACATTCGTCATTTTAAAA	3768.132568	train
Zm00001d027292	CTTCACATCATCAACTTCATAATAATT	21.867928	test
Zm00001d027295	TTCCAGAAGACATAAAGTCTGTGTTAA	2193.95166	train
Zm00001d027304	CATCGACGACGTGGGGAGGAGGGTGGG	100.480644	test
Zm00001d027307	GCACTGGACAGTGTCCGGTGCGCCAAC	73.999771	train
Zm00001d027311	ACTTGCAATAAGCATGTCTTTTTAATT	9.518116	train
Zm00001d027312	CTGATATACGAACCCCGTGTTCAAGCA	20.452984	train
Zm00001d027314	TACGACCCAAACACCGCACAACGCTCG	187.788651	train
Zm00001d027317	TCATACACGCATGACACACATGTGCTT	191.618698	test
Zm00001d027319	TGTCCACACATCAAATAATCTGGCAGA	52.433243	train
Zm00001d027321	TAGCCTCCAATCATCCCTCAACTCTAT	163.306503	train
Zm00001d027322	TGTTTGGAGCTGATGGGAATCGTCGTA	96.274788	train
Zm00001d027323	GTGAGACCTCTCTTGGCCATTCCTTGC	382.299866	train
Zm00001d027324	AGTGGGGCGCGAAAATTCATTTGATGA	89.222183	train
Zm00001d027325	GAGCTGCCGCTCTCCCACATGCCCGCC	37.097458	train
Zm00001d027329	AGCGAAAGAGGGCAAGCAAGCAGAGAA	91.757378	train
Zm00001d027330	TCATCTTTTTCTTAGAAGGGCTATTCA	260.086792	train

图 2: 数据集示意图



Input : DNA sequence
3K-bp proximal regulatory genomic sequences (s)
Promoter
5'UTR
3'UTR
Terminator
TSS
T T A G C A
A G T C G
One-hot encoding
GELU
Conv (Cm/2, 15, 1)
MaxPool
BatchNorm
GELU
Conv (C, W, D)
ConvLayer (Cm/2, 5, Di)
MaxPool ()
ConvBlock
xM
Model architecture
ConvLayer (Cm/2, 3, Di)
ConvLayer (Cm, 1, 1)
Dropout ()
Dilated residual blocks
xN
Constants:
Cm : Maximum number of filters
C : Number of filters
D : Dilation rate
W : Size of filters
Width : Size of the max pooling window
Out: Number of output neurons
ConvLayer (Cm, 1, 1)
Dropout ()
GELU
Output : Gene expression prediction
Conv (1, 1, 1)
Flatten
Dense (Out = 1)

图 3: 网络结构示意图

3. 实验代码

3.1 数据预处理代码

```python
# %%
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
# %%
def one_hot_encode_along_channel_axis(sequence):
    to_return = np.zeros((4, len(sequence)), dtype=np.int8)
    seq_to_one_hot_fill_in_array(zeros_array=to_return,sequence=sequence,
        one_hot_axis=0)

    return to_return

def seq_to_one_hot_fill_in_array(zeros_array, sequence, one_hot_axis):
    assert one_hot_axis==0 or one_hot_axis==1
    if (one_hot_axis==0):
        assert zeros_array.shape[1] == len(sequence)
    elif (one_hot_axis==1):
        assert zeros_array.shape[0] == len(sequence)
    #will mutate zeros_array
    for (i,char) in enumerate(sequence):
        if (char=="A" or char=="a"):
            char_idx = 0
        elif (char=="C" or char=="c"):
            char_idx = 1
        elif (char=="G" or char=="g"):
            char_idx = 2
        elif (char=="T" or char=="t"):
            char_idx = 3
        elif (char=="N" or char=="n"):
            continue #leave that pos as all 0's
        else:
            raise RuntimeError("Unsupported character: "+str(char))
        if (one_hot_axis==0):
            zeros_array[char_idx,i] = 1
        elif (one_hot_axis==1):
            zeros_array[i,char_idx] = 1
# %%
# X = range(len(y)) # 37979
# x_train, x_test, y_train, y_test = train_test_split(X,
```

```
#                                                   y,
#                                                   train_size=0.8,
#                                                   random_state=2023)
# x_train, x_valid, y_train, y_valid = train_test_split(x_train,
#                                                   y_train,
#                                                   train_size=0.9,
#                                                   random_state=2023)
# x_train:27344, x_valid:3039, x_test:7596
#%%
#为了统一，这里提前为大家划分好了训练集，验证集和测试集，见df['dataset']列
#导入划分好的数据集,并对DNA序列one-hot编码处理
df = pd.read_excel('/mnt/cgshare/dataset.xlsx')
print('genes number:',df.shape[0])
df_train = df[df['dataset']=='train']
y_train = np.log2(df_train['TPM'].values + 1)
train_data = np.array([one_hot_encode_along_channel_axis(i.strip()) for i in df_train
    ['sequence'].values])
df_valid = df[df['dataset']=='valid']
y_valid = np.log2(df_valid['TPM'].values + 1)
valid_data = np.array([one_hot_encode_along_channel_axis(i.strip()) for i in df_valid
    ['sequence'].values])
df_test = df[df['dataset']=='test']
y_test = np.log2(df_test['TPM'].values + 1)
test_data = np.array([one_hot_encode_along_channel_axis(i.strip()) for i in df_test['
    sequence'].values])
```

3.2 搭建模型及训练、测试代码

```
# %%
import numpy as np
import os
import torch
import torch.nn as nn
from torch import Tensor
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, ConcatDataset
import torch.nn.functional as F
#from torch.cuda.amp.grad_scaler import GradScaler
#from torch.cuda.amp import autocast
#from torchsummary import summary
from tqdm import tqdm
from scipy.stats import pearsonr
import matplotlib.pyplot as plt
from einops.layers.torch import Rearrange
```

```python
from torchsummary import summary

# define blocks
class upd_GELU(nn.Module):
    def __init__(self):
        super(upd_GELU, self).__init__()
        self.constant_param = nn.Parameter(torch.Tensor([1.702]))
        self.sig = nn.Sigmoid()

    def forward(self, input: Tensor) -> Tensor:
        outval = torch.mul(self.sig(torch.mul(self.constant_param, input)), input)
        return outval

# 由于torch的polling操作没有padding='same',因此重新定义了maxpooling类
class KerasMaxPool1d(nn.Module):
    def __init__(
        self,
        pool_size=2,
        padding="valid",
        dilation=1,
        return_indices=False,
        ceil_mode=False,
    ):
        """Hard coded to only be compatible with kernel size and stride of 2."""
        super().__init__()
        self.padding = padding
        _padding = 0
        if pool_size != 2:
            raise NotImplementedError("MaxPool1D with kernel size other than 2.")
        self.pool = nn.MaxPool1d(
            kernel_size=pool_size,
            padding=_padding,
            dilation=dilation,
            return_indices=return_indices,
            ceil_mode=ceil_mode,
        )

    def forward(self, x):
        # (b c h)
        if self.padding == "same" and x.shape[-1] % 2 == 1:
            x = F.pad(x, (0, 1), value=-float("inf"))
        return self.pool(x)

```

```
class Residual(nn.Module):
    def __init__(self, module):
        super().__init__()
        self.module = module

    def forward(self, x):
        return self.module(x) + x

class ConvBlock(nn.Module):
    def __init__(self, in_channels, filters, kernel_size=1, padding='same',
                 stride=1, dilation_rate=1, pool_size=1 , dropout=0, bn_momentum
                    =0.1):
        super().__init__()
        block = nn.ModuleList()
        block.append(upd_GELU())
        block.append(nn.Conv1d(in_channels=in_channels,
                     out_channels=filters,
                     kernel_size=kernel_size,
                     stride=stride,
                     padding=padding,
                     dilation=int(round(dilation_rate)),
                     bias=False))
        block.append(nn.BatchNorm1d(filters, momentum=bn_momentum, affine=True))
        if dropout > 0:
            block.append(nn.Dropout(p=dropout))
        if pool_size>1:
            block.append(KerasMaxPool1d(pool_size=pool_size, padding=padding))
        self.block = nn.Sequential(*block)
        self.out_channels = filters

    def forward(self, x):
        return self.block(x)

class ConvTower(nn.Module):
    def __init__(
        self,
        in_channels,
        filters_init,
        filters_end=None,
        filters_mult=None,
        divisible_by=1,
        repeat=2,
        **kwargs,
```

```
    ):
        super().__init__()

        def _round(x):
            return int(np.round(x / divisible_by) * divisible_by)

        # determine multiplier
        if filters_mult is None:
            assert filters_end is not None
            filters_mult = np.exp(np.log(filters_end / filters_init) / (repeat - 1))

        rep_filters = filters_init
        in_channels = in_channels
        tower = nn.ModuleList()
        for _ in range(repeat):
            tower.append(
                ConvBlock(
                    in_channels=in_channels, filters=_round(rep_filters), **kwargs
                )
            )
            in_channels = _round(rep_filters)
            rep_filters *= filters_mult

        self.tower = nn.Sequential(*tower)
        self.out_channels = in_channels

    def forward(self, x):
        return self.tower(x)

class DilatedResidual(nn.Module):
    def __init__(
        self,
        in_channels,
        filters,
        kernel_size=3,
        rate_mult=2,
        dropout=0,
        repeat=1,
        **kwargs,
    ):
        super().__init__()
        dilation_rate = 1 # 初始化为1, 后面累乘
        in_channels = in_channels
```

```
        block = nn.ModuleList()
        for _ in range(repeat):
            inner_block = nn.ModuleList()

            inner_block.append(
                ConvBlock(
                    in_channels=in_channels,
                    filters=filters,
                    kernel_size=kernel_size,
                    dilation_rate=int(np.round(dilation_rate)),
                    **kwargs,
                )
            )

            inner_block.append(
                ConvBlock(
                    in_channels=filters,
                    filters=in_channels,
                    dropout=dropout,
                    **kwargs,
                )
            )

            block.append(Residual(nn.Sequential(*inner_block)))

            dilation_rate *= rate_mult
            dilation_rate = np.round(dilation_rate)
        self.block = nn.Sequential(*block)
        self.out_channels = in_channels

    def forward(self, x):
        return self.block(x)

class BasenjiFinal(nn.Module):
    def __init__(
        self, in_features, units=1, activation='linear', **kwargs):
        super().__init__()
        block = nn.ModuleList()
        #if flatten:
        block.append(Rearrange('b ... -> b (...)'))

        # Rearrange('b ... -> b (...)') 代表将batch size维度保留，剩下的维度变成一维
        # 这里相当于flatten
```

```
            # block.append(nn.Conv1d(in_features, units, kernel_size=1, stride=1, padding
                =0))

            block.append(nn.Linear(in_features=in_features, out_features=units))
            self.block = nn.Sequential(*block)

        def forward(self, x):
            return self.block(x)

class BasenjiModel(nn.Module):
        def __init__(self, conv1_filters=8, conv1_ks=15,
                    # 第一个conv block参数
                    conv1_pad=7, conv1_pool=2, conv1_pdrop=0.4, conv1_bn_momentum=0.1,
                    # conv tower参数
                    convt_filters_init=16, filters_end=32, convt_repeat=2, convt_ks=5,
                        convt_pool=2,
                    # dil res block参数
                    dil_in_channels=32, dil_filters=16, dil_ks=3, rate_mult=2, dil_pdrop=0.3,
                         dil_repeat=2,
                    conv2_in_channels=32, conv2_filters=32, # 第二个conv block 参数
                    conv3_in_channels=32, conv3_filters=1, # 第三个conv block (1*1 conv) 参数
                    final_in_features=int(3000/(2**3)) # final block参数, 2**3表示经过3次
                        pooling
                          ):
            super().__init__()
            block = nn.ModuleList()
            block.append(ConvBlock(in_channels=4,
                                  filters=conv1_filters,
                                  kernel_size=conv1_ks,
                                  padding=conv1_pad,
                                  pool_size=conv1_pool,
                                  dropout=conv1_pdrop,
                                  bn_momentum=conv1_bn_momentum))

            block.append(ConvTower(in_channels=conv1_filters,
                                  filters_init=convt_filters_init,
                                  # filters_mult=convt_filters_mult,
                                  filters_end=filters_end,
                                  divisible_by=1,
                                  repeat=convt_repeat,
                                  kernel_size=convt_ks,
                                  pool_size=convt_pool,
                                  ))
```

```
        block.append(DilatedResidual(
                        in_channels=dil_in_channels,
                        filters=dil_filters,
                        kernel_size=dil_ks,
                        rate_mult=rate_mult,
                        dropout=dil_pdrop,
                        repeat=dil_repeat,))

        block.append(ConvBlock(in_channels=conv2_in_channels,
                        filters=conv2_filters,
                        kernel_size=1))
        block.append(ConvBlock(in_channels=conv3_in_channels,
                        filters=conv3_filters,
                        kernel_size=1))
        block.append(BasenjiFinal(final_in_features))
        self.block = nn.Sequential(*block)

    def forward(self, x):
        return self.block(x)
# %%
# define dataset class
class MyDataset(Dataset):
    def __init__(self, input, label):
        inputs = torch.tensor(input,dtype=torch.int8)
        labels = torch.tensor(label,dtype=torch.float32)
        self.inputs = inputs
        self.labels = labels

    def __getitem__(self, index):
        return self.inputs[index], self.labels[index]

    def __len__(self):
        return len(self.labels)

# %%
BATCH_SIZE = 32
EPOCHS = 10
lr=1e-3

# valid data可以用于早停，这里没有加入早停机制，你们可以自己尝试加入
train_dataset = MyDataset(train_data,y_train)
```

```
trainDataLoader = torch.utils.data.DataLoader(dataset = train_dataset, batch_size =
    BATCH_SIZE, shuffle = True)
valid_dataset = MyDataset(valid_data, y_valid)
validDataLoader = torch.utils.data.DataLoader(dataset = valid_dataset, batch_size =
    BATCH_SIZE)
test_dataset = MyDataset(test_data, y_test)
testDataLoader = torch.utils.data.DataLoader(dataset = test_dataset, batch_size =
    BATCH_SIZE)
# %%
device = "cuda:0" if torch.cuda.is_available() else "cpu"

net = BasenjiModel()
summary(net, input_size=[(4, 3000)], batch_size=BATCH_SIZE, device="cpu")
optimizer = torch.optim.Adam(params= net.parameters(), lr=lr)
lossF = nn.MSELoss()
print(net.to(device))
#存储训练过程
history = {'Valid Loss':[], 'Valid pcc':[]}
for epoch in range(1,EPOCHS + 1):
    processBar = tqdm(trainDataLoader,unit = 'step')
    net.train(True)
    epoch_loss_all = 0
    for step,(inputs,labels) in enumerate(processBar):
        #将序列和标签传输进device中
        inputs = inputs.to(device)
        labels = labels.to(device)
        #清空模型的梯度
        net.zero_grad()
        #对模型进行前向推理
        outputs = net(inputs)
        #计算本轮推理的Loss值
        loss = lossF(outputs.reshape(labels.shape), labels)
        epoch_loss_all += loss*len(labels)
        #进行反向传播求出模型参数的梯度
        loss.backward()
        #使用迭代器更新模型权重
        optimizer.step()
        #将本step结果进行可视化处理
        processBar.set_description("[%d/%d] Loss: %.4f" %
                                   (epoch,EPOCHS,loss.item()))
        if step == len(processBar)-1:
            valid_totalLoss = 0
            y_valid_pred_all = []
```

```
            y_valid_true_all = []
            train_loss = epoch_loss_all / len(train_data)
            #关闭模型的训练状态
            net.train(False)
            with torch.no_grad():
                #对验证集的DataLoader进行迭代
                for x_valid, y_valid in validDataLoader:
                    x_valid = x_valid.to(device)
                    y_valid = y_valid.to(device)
                    y_valid_pred = net(x_valid)
                    y_valid_pred_all.extend(y_valid_pred.flatten().tolist())
                    y_valid_true_all.extend(y_valid.tolist())
                    valid_batch_loss = lossF(y_valid_pred.reshape(y_valid.shape),
                        y_valid)
                    valid_totalLoss += valid_batch_loss * len(y_valid)
                validLoss = valid_totalLoss/len(valid_data)
                pcc_valid, _ = pearsonr(y_valid_pred_all, y_valid_true_all)
                history['Valid Loss'].append(validLoss.item())
                history['Valid pcc'].append(pcc_valid.item())
                processBar.set_description("[%d/%d] Train Loss: %.4f, Valid Loss: %.4
                    f, Valid pcc: %.4f"%
                                        (epoch, EPOCHS, train_loss.item(), validLoss.item
                                            (), pcc_valid))
    processBar.close()


# %%
# calculate test pcc
# calculate test pcc and R2
net.train(False)
y_test_pred_all = []
y_test_true_all = []
with torch.no_grad():
    #对测试集的DataLoader进行迭代
    for x_test, y_test in testDataLoader:
        x_test = x_test.to(device)
        y_test = y_test.to(device)
        y_test_pred = net(x_test)
        y_test_pred_all.extend(y_test_pred.flatten().tolist())
        y_test_true_all.extend(y_test.tolist())
# 计算测试集上的pcc和R2, 并画测试集上真实值和预测值之间的散点图
pcc, _ = pearsonr(y_test_pred_all, y_test_true_all)
print('pcc: ',pcc)
```

```
plt.xlabel('y test predict')
plt.ylabel('y test true')
plt.scatter(y_test_pred_all, y_test_true_all, s=0.1)
```