

**《实验报告》**

实验题目 设计实现Hugging Face数据集用途分类

学院系别 信息学院

专业名称 人工智能

学生姓名 郑凯航

学生学号 37220222203885

任课教师 曹冬林

2024年 5 月 9 日

# 一、 实验目的

列举实验要达到的几个目的。

* 针对实验1和实验2构建的数据集信息分析以下内容：

•1. 设计实现通过数据简介进行大类分类的程序。

•2.设计实现通过数据简介进行大类+小类分类的程序。

# 二、 实验环境

列举实验机器环境，配置环境。

os: Windows

platform Windows-10-10.0.22621-SP0 version: 10.0.22621

python: 3.8.18 requests==2.27.1 bs4==0.0.2

lxml==4.9.3

# 三、 实验步骤

详细说明所做实验的基本实验步骤及流程图。

 数据处理  词嵌入

 标签编码

 相关性评估

### 数据

#### 数据预处理

空缺值处理

 对于数据简介，其中有许多无意义的换行符号，手动进行处理

 对于url,前面的[https://huggingface.co/datasets/无意义，手动删除](https://huggingface.co/datasets/%E6%97%A0%E6%84%8F%E4%B9%89%EF%BC%8C%E6%89%8B%E5%8A%A8%E5%88%A0%E9%99%A4)  获取到的网页数据有部分缺失，使用-1替换

df.iloc[:,-1] = df.iloc[:,-1].apply(lambda

x:x.replace('\\n','').replace('\\t','')) df.iloc[:,0] = df.iloc[:,0].apply(lambda x:x.replace('https://huggingface.co/datasets/',''))

df.fillna(value=-1,inplace=True)

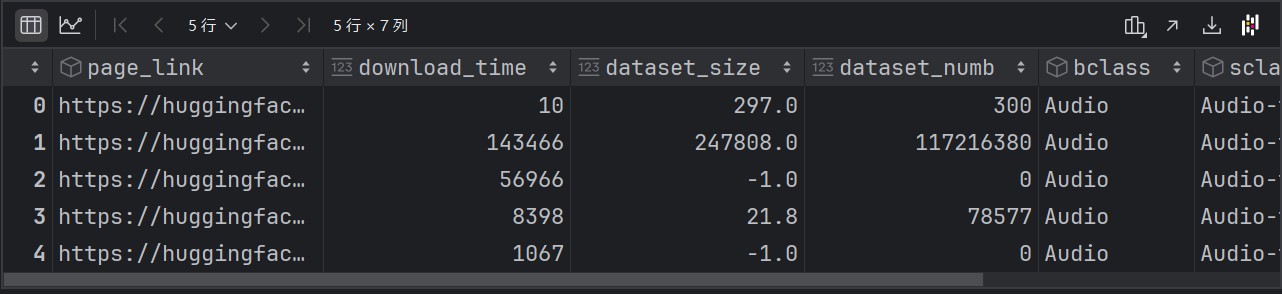
x[:,:3][x[:,:3] == -1] = np.nan

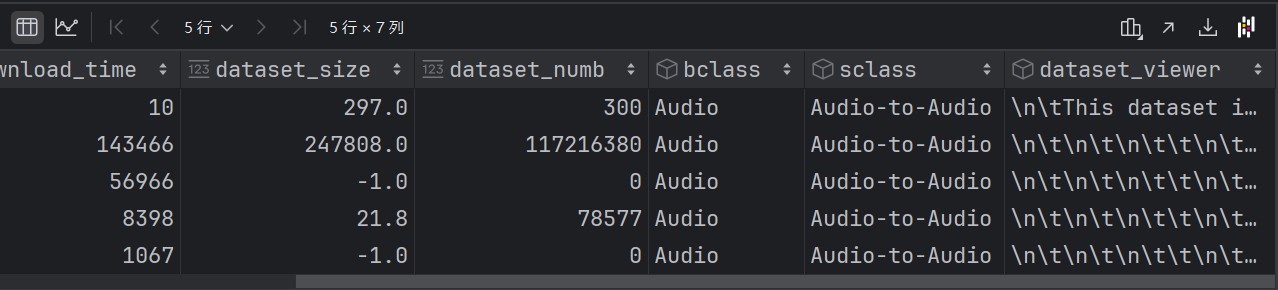
x[:,:3][x[:,:3] == 0] = np.nan

x = pd.DataFrame(x)

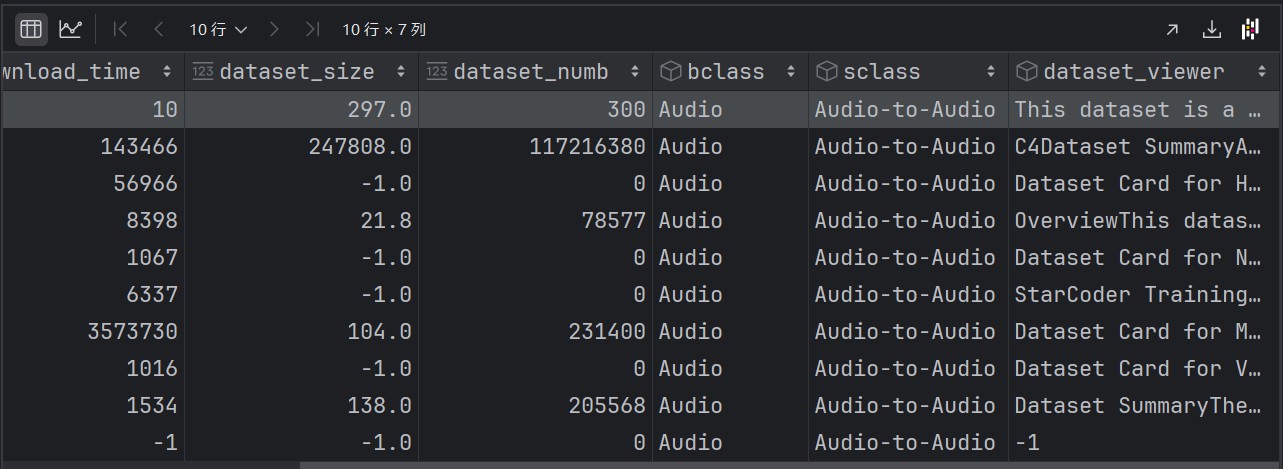
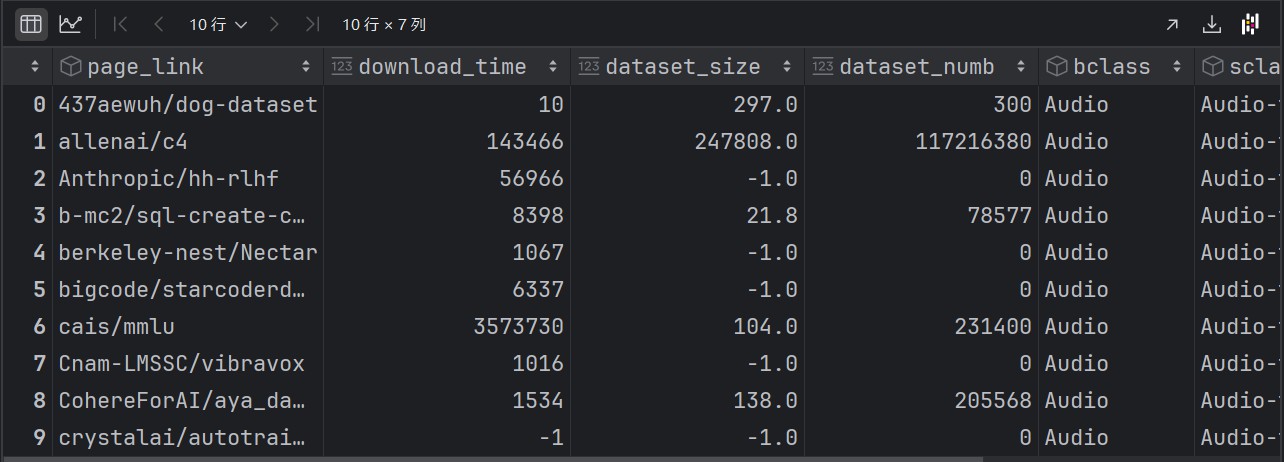
# xx == np.nan

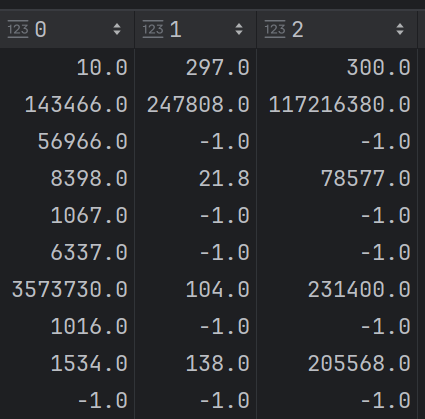
x = pd.concat([x.iloc[:,:3].isna().astype(float),x.fillna(-1)],axis=1) x[:10]





处理前





处理后

 文本向量化

借助网络上预处理好的模型将数据简介和数据名向量化，从而可以被处理在hugging face上找到可以将文本转为512维的向量的模型

jina\_embeddings\_v2\_small\_en

编码函数

def generate\_embeddings():

"""

文本转向量

""" # 返回numpy数据

TextEmbedding\_model = AutoModel.from\_pretrained('jinaai/jina- embeddings-v2-small-en', trust\_remote\_code=True)

return TextEmbedding\_model

embedding\_model = generate\_embeddings()

def m\_embedding(text):

"""

文本转向量

:param text::return: tensor

"""

embeddings = embedding\_model.encode(text,max\_length=3487)

return embeddings

选择的模型可以指定文本最大长度，所以先计算数据简介的最大长度并填入编码器

m = 0

for i in df.iloc[:,-1].to\_list():

l = i.split(). len ()

m = max(m,l) print('max\_length',m)

编码，使用applymap 处理

x\_encode = df.iloc[:,[0,-1]].applymap(lambda x:m\_embedding(x) if type(x)==str else x)

之后将文本向量化后的数据与下载信息等结合在一起

# 转为numpy对象

x = x\_encode.to\_numpy() x\_ = []

for i in range(x.shape[0]): ls = []

x\_.append(list(np.append(x[i][0],x[i][-1]))) x = np.array(x\_)

x1 = x

x\_ = df.iloc[:,1:4]

def f(x):

try:

x = float(x) except Exception:

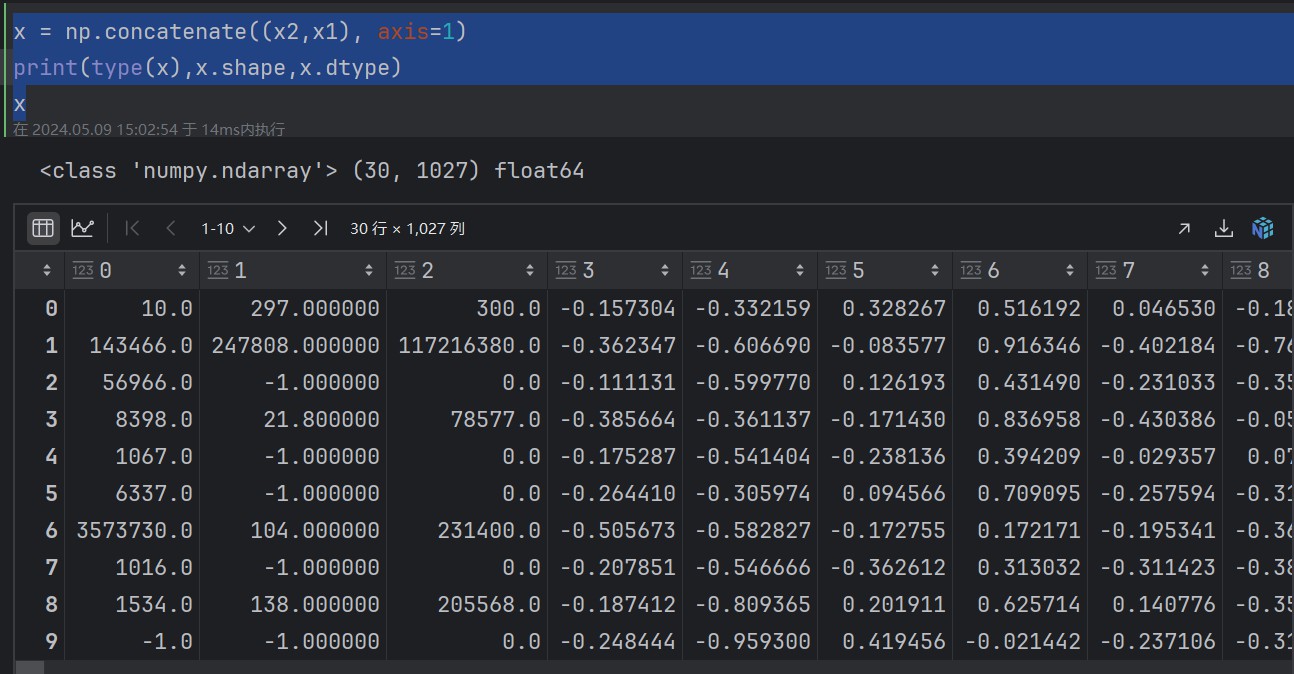
x = -1

return x

x2 = x\_.applymap(f).fillna(value=-1).to\_numpy()

x = np.concatenate((x2,x1), axis=1)

最终得到的数据



### 标签

#### one\_hot编码

1. 获取大小类，通过对网页的信息处理获得大小类标签相关信息

if os.path.exists(r'Hugging \_dataset.html'):

with open(r'Hugging \_dataset.html','r',encoding='utf-8') as f: dataset\_html = f.read()

else:

dataset\_web\_respond = requests.get(url=r'https://huggingface.co/datasets',verify=False)

dataset\_html = dataset\_web\_respond.text

soup = bs(dataset\_html,'html.parser')

tags = soup.select('div.mb-20')[0].contents btags = []

for i in tags: try:

if 'mb-3' in i.get('class'):

btags.append(i) except Exception:

pass cls = dict() scls = []

for btag in btags:

blabel = btag.div.string.rstrip('\t').rstrip('\n') temp = []

for a in btag.find\_all('a'):

slabel = a.span.string

scls.append(slabel) temp.append(slabel)

cls[blabel]=temp s\_class\_count = 0

for k,v in cls.items(): s\_class\_count += len(v) # print(k,v)

print('小类总数',s\_class\_count)

1. one\_hot编码

发现小类一共46个，则可以用维度46的向量作为标签

通过得到的映射关系构建c2h字典，将小类名映射到标签

c2h = dict()

label = np.eye(46,46,dtype=np.float64) k = 0

for \_,v in cls.items(): for c in v :

# print(v)

c2h[c.replace(' ','\_')] = label[k] k += 1

y = df.loc[:,'sclass'].apply(lambda x:np.array(c2h[x]))

y = np.vstack(y)

#### 相关性评估

由于发现直接编码后的数据用来训练网络效果很差，所以继续将46个小类标签也编码，随后计算他们相关度，将512\*2维转为46\*2维

本质是将文本信息和标签信息映射到函数空间上，然后计算两个向量的相关度

x = pd.DataFrame(x)

y = pd.DataFrame(y)

n = pd.concat([x,y],axis=1)

non\_duplicate\_indices = n.iloc[:, :1030].drop\_duplicates().index

n = n.loc[non\_duplicate\_indices]

x = torch.Tensor(n.iloc[:,:1030].values) y = torch.Tensor(n.iloc[:,1030:].values)

# 计算相关性的函数

cosine\_fn = util.cos\_sim

# 记录最后的相关度

df = pd.DataFrame(np.zeros([15347, 2\*46]))

# 小类到编码的映射

s2e = dict()

for k,v in cls.items(): s\_class\_count += len(v) for s in v:

lab = m\_embedding(k+s)

s2e[s] = lab

# 计算相关度

for i in range(15347):

for j,(k,v) in enumerate(s2e.items()):

df.iloc[i,j] = cosine\_fn(v,x[i,:512]).numpy()[0,0]

df.iloc[i,j+46] = cosine\_fn(v,x[i,512:]).numpy()[0,0]

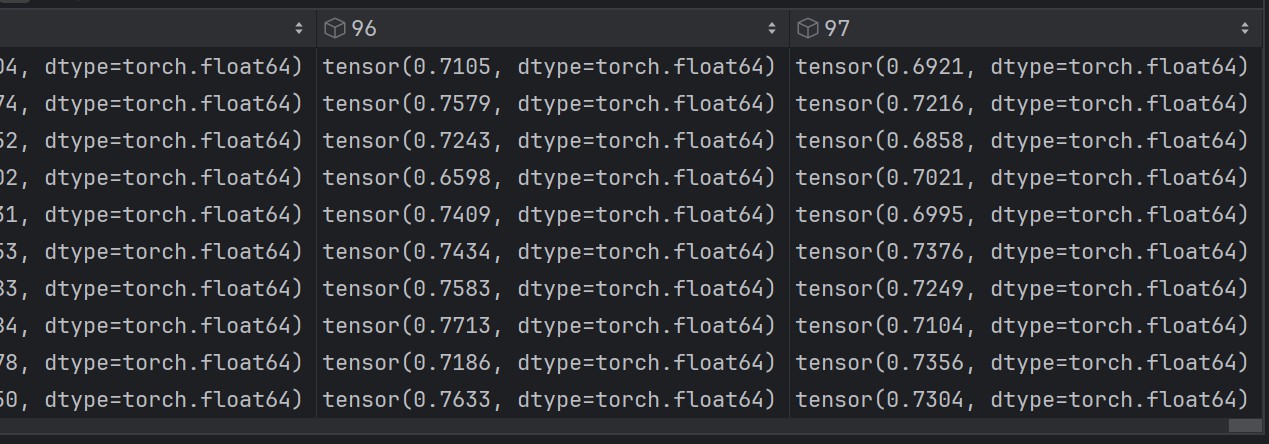
x = pd.DataFrame(x)

df = pd.concat([x.iloc[:,:6],df],axis=1).shape

my\_array = np.array(df)

my\_tensor = torch.tensor(my\_array) torch.save(my\_tensor,'data')

最后可以得到98维的向量，其中数据简介，数据标签与小类的相关性占46\*2=92维，剩下的6维表示三位下载次数，数据大小和数据条数和三位代表他们是不是缺失值的标志位



## 类型转换

由于在pytorch模型上进行计算的数据必须有一致类型和设备，因此在开始就将其基于平台定义

def try\_gpu(i=0):

if torch.cuda.device\_count() >= i + 1: return torch.device(f'cuda:{i}')

return torch.device('cpu')

device = try\_gpu()

example = True

typ = torch.float32

在数据处理完成后将其进行转换，避免不必要的麻烦

x,y = torch.load('data.data') x = x.to(typ).to(device)

y = y.to(typ).to(device)

## 网络搭建

参考网上资料，使用pytorch框架

# 组件1

def sblock(input\_size,ouput\_size): "1个全连接层"

return nn.Sequential(

nn.BatchNorm1d(input\_size), # nn.Dropout(0.1),

nn.Linear(input\_size,ouput\_size),

nn.ReLU()

)

# 组件2

class inception(nn.Module): """

并行网络

""" def

init (self,in\_size,c1\_size,c2\_size,c3\_size,c4\_size,\*\*kwargs): super(inception,self). init (\*\*kwargs)

self.l1 = sblock(in\_size,c1\_size) self.l2 = sblock(in\_size,c2\_size) self.l3 = sblock(in\_size,c3\_size)

self.l4 = sblock(in\_size,c4\_size)

def forward(self,x):

return torch.cat((self.l1(x),self.l2(x),self.l3(x),self.l4(x)),dim=1)

# 组件3

class inception\_re(nn.Module): """

并行网络

""" def

init (self,c1\_size,c2\_size,c3\_size,out1\_size,out2\_size,out3\_size,\*\*kwarg s):

super(inception\_re,self). init (\*\*kwargs) self.c1 = c1\_size

self.c2 = c2\_size + c1\_size

self.c3 = c3\_size + c2\_size

self.l1 = sblock(c1\_size,out1\_size)

self.l2 = sblock(c2\_size,out2\_size) self.l3 = sblock(c3\_size,out3\_size)

def forward(self,x):

return torch.cat( (

self.l1(x[:,:self.c1]),

self.l2(x[:,self.c1:self.c2]),

self.l3(x[:,self.c2:])

),dim=1)

# 小类网络

class Go\_net(nn.Module): """

训练用网络

""" def init (self,input\_size=98): """

全连接+(并行+全连接)\*n+全连接

:param input\_size::param output\_size: 最后分类个数

""" super(Go\_net,self). init ()

self.inception\_re1 = inception\_re(6,46,46,8,124,124)

# self.sblock1 = sblock(input\_size,128)

# self.sblock2 = sblock(1024,1024)

self.inception1 = inception(256,256,256,256,256) self.sblock5 = sblock(1024,128)

self.sblock7 = sblock(128,46)

self.sequential = nn.Sequential(

self.inception\_re1,

self.inception1, self.sblock5, self.sblock7

)

def forward(self,x): """

前向传播

:param X::return:

"""

return self.sequential(x)

def sblock(input\_size,ouput\_size): "1个全连接层"

return nn.Sequential( nn.BatchNorm1d(input\_size),

# nn.Dropout(0.1), nn.Linear(input\_size,ouput\_size), nn.ReLU()

)

# 参数及网络初始化

batch\_size = 1024

lr = 0.0001

epoch = 20000

m\_Go\_net = Go\_net().to(torch.float32).to(device)

m\_net = m\_Go\_net m\_net\_name = 'm\_Go\_net' m\_net.apply(init\_xavier)

x = x.to(torch.float32).to(device) y = y.to(torch.float32).to(device) m\_net.to(device)

dataset = load\_array((x,y),batch\_size)

# 优化器

optimizer = torch.optim.Adam(m\_net.parameters(),lr=lr,weight\_decay=1e-4) scheduler = lr\_scheduler.StepLR(optimizer,step\_size=100,gamma=0.9)

ok\_model = train(data\_loader=dataset,lr=lr,net=m\_net,epochs=epoch,optimizer=optimizer, scheduler=scheduler,device=device,flag=m\_net\_name)

# 大类网络

# 瞎写的

class Go\_net(nn.Module): """

训练用网络

"""

def init (self,input\_size=98): """

全连接+(并行+全连接)\*n+全连接

:param input\_size:

:param output\_size: 最后分类个数

"""

super(Go\_net,self). init ()

# self.inception\_re1 = inception\_re(6,46,46,8,60,60) # self.sblock1 = sblock(input\_size,128)

# self.sblock2 = sblock(1024,1024)

self.inception1 = inception(input\_size,128,128,128,128)

# self.sblock3 = sblock(2048,1024)

# self.inception2 = inception(1024,64,64,128,256)

self.sblock5 = sblock(512,256)

# self.inception3 = inception(256,32,32,64,128) self.sblock7 = sblock(256,7)

self.sequential = nn.Sequential( # self.inception\_re1,

# self.sblock1, self.inception\_re1, self.inception1,

# self.sblock2, # self.sblock3,

# self.inception2, # self.sblock4, self.sblock5,

# self.inception3, # self.sblock6, self.sblock7

)

def forward(self,x): """

前向传播

:param X:

:return:

"""

# x = self.sblock1(x)

# x = self.sblock2(x)

# x = self.inception1(x) # # x = self.sblock3(x)

# # x = self.inception2(x) # x = self.sblock4(x)

# x = self.sblock5(x)

# return x

return self.sequential(x)

def sblock(input\_size,ouput\_size): "1个全连接层"

return nn.Sequential(

nn.BatchNorm1d(input\_size), nn.Dropout(0.4), nn.Linear(input\_size,ouput\_size), nn.ReLU()

)

# 参数及网络初始化

batch\_size = 1024

lr = 0.001

epoch = 20000

m\_Go\_net = Go\_net().to(torch.float32).to(device)

m\_net = m\_Go\_net m\_net\_name = 'm\_Go\_net' m\_net.apply(init\_xavier)

x = x.to(torch.float32).to(device) y = y.to(torch.float32).to(device) m\_net.to(device)

dataset = load\_array((x,y),batch\_size)

# 优化器

optimizer = torch.optim.Adam(m\_net.parameters(),lr=lr,weight\_decay=1e-4) scheduler = lr\_scheduler.StepLR(optimizer,step\_size=100,gamma=0.9)

ok\_model =

train(data\_loader=dataset,lr=lr,net=m\_net,epochs=epoch,optimizer=optimizer, scheduler=scheduler,device=device,flag=m\_net\_name)

初始化参数，避免梯度丢失，爆炸

# 初始化参数

def init\_xavier(m): """

xavier初始化可以避免梯度爆炸、梯度消失

:param m::return:

"""

if isinstance(m, nn.Linear): nn.init.xavier\_uniform\_(m.weight)

## 训练

在进行调参后找到比较好的模型参数

# 参数及网络初始化

batch\_size = 1024

lr = 0.0001

epoch = 20000

m\_Go\_net = Go\_net().to(torch.float32).to(device)

m\_net = m\_Go\_net

m\_net\_name = 'm\_Go\_net' m\_net.apply(init\_xavier)

x = x.to(torch.float32).to(device)

y = y.to(torch.float32).to(device) m\_net.to(device)

dataset = load\_array((x,y),batch\_size)

# 优化器

optimizer = torch.optim.Adam(m\_net.parameters(),lr=lr,weight\_decay=1e-4)

scheduler = lr\_scheduler.StepLR(optimizer,step\_size=100,gamma=0.9)

# 训练

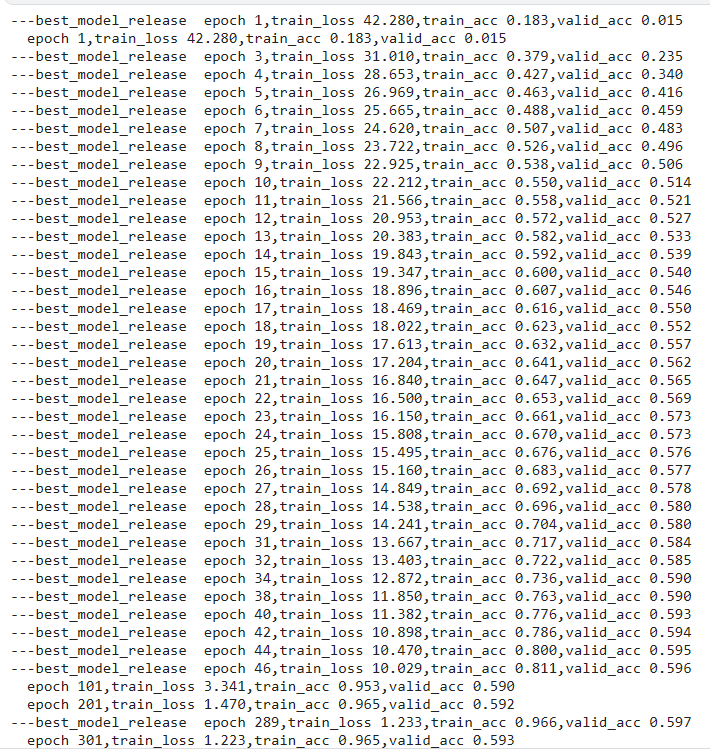
ok\_model = train(data\_loader=dataset,lr=lr,net=m\_net,epochs=epoch,optimizer=optimizer,

scheduler=scheduler,device=device,flag=m\_net\_name)

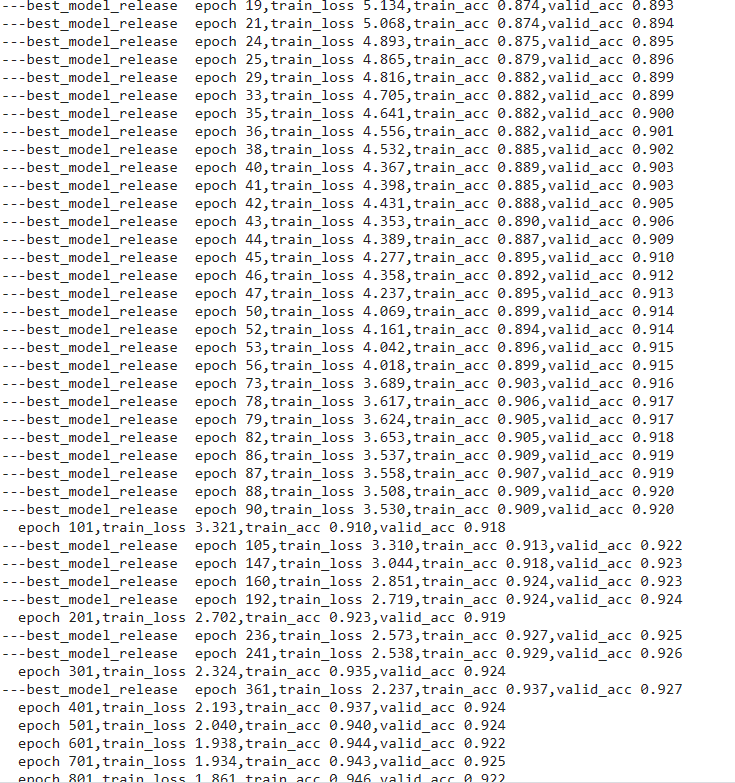
可以在在线平台上训练，免费使用TPU

## 测试

小类



大类



测试结果

---best\_model\_release epoch 1,train\_loss 29.535,train\_acc 0.361,valid\_acc

0.048

epoch 1,train\_loss 29.535,train\_acc 0.361,valid\_acc 0.048

---best\_model\_release

0.068

---best\_model\_release 0.157

---best\_model\_release

0.427

epoch 2,train\_loss 21.268,train\_acc 0.518,valid\_acc

epoch 3,train\_loss 18.519,train\_acc 0.567,valid\_acc

epoch 4,train\_loss 16.645,train\_acc 0.606,valid\_acc

---best\_model\_release epoch 5,train\_loss 15.016,train\_acc 0.640,valid\_acc 0.488

---best\_model\_release epoch 6,train\_loss 13.634,train\_acc 0.678,valid\_acc

0.559

---best\_model\_release epoch 7,train\_loss 12.264,train\_acc 0.716,valid\_acc 0.573

---best\_model\_release epoch 8,train\_loss 10.830,train\_acc 0.751,valid\_acc

0.581

---best\_model\_release epoch 9,train\_loss 9.445,train\_acc 0.790,valid\_acc 0.582

---best\_model\_release epoch 12,train\_loss 6.134,train\_acc 0.880,valid\_acc

0.592

---best\_model\_release epoch 18,train\_loss 2.966,train\_acc 0.942,valid\_acc 0.593

---best\_model\_release epoch 19,train\_loss 2.522,train\_acc 0.953,valid\_acc

0.594

---best\_model\_release epoch 21,train\_loss 1.932,train\_acc 0.964,valid\_acc 0.595

---best\_model\_release epoch 24,train\_loss 1.581,train\_acc 0.969,valid\_acc

0.598

---best\_model\_release epoch 37,train\_loss 0.901,train\_acc 0.978,valid\_acc 0.601

---best\_model\_release epoch 60,train\_loss 0.738,train\_acc 0.978,valid\_acc

0.601

---best\_model\_release epoch 80,train\_loss 0.712,train\_acc 0.979,valid\_acc 0.603

---best\_model\_release epoch 81,train\_loss 0.703,train\_acc 0.980,valid\_acc

0.604

---best\_model\_release epoch 82,train\_loss 0.648,train\_acc 0.979,valid\_acc 0.605

---best\_model\_release epoch 86,train\_loss 0.601,train\_acc 0.980,valid\_acc

0.607

---best\_model\_release epoch 94,train\_loss 0.592,train\_acc 0.978,valid\_acc 0.608

epoch 101,train\_loss 0.588,train\_acc 0.980,valid\_acc 0.601

---best\_model\_release epoch 175,train\_loss 0.625,train\_acc 0.979,valid\_acc 0.608

---best\_model\_release epoch 176,train\_loss 0.575,train\_acc 0.980,valid\_acc 0.609

---best\_model\_release epoch 178,train\_loss 0.548,train\_acc 0.981,valid\_acc 0.610

---best\_model\_release epoch 179,train\_loss 0.513,train\_acc 0.979,valid\_acc 0.611

---best\_model\_release epoch 191,train\_loss 0.489,train\_acc 0.981,valid\_acc

0.612

---best\_model\_release epoch 192,train\_loss 0.478,train\_acc 0.980,valid\_acc 0.612

epoch 201,train\_loss 0.463,train\_acc 0.980,valid\_acc 0.612

---best\_model\_release epoch 209,train\_loss 0.482,train\_acc 0.979,valid\_acc 0.613

---best\_model\_release epoch 217,train\_loss 0.456,train\_acc 0.981,valid\_acc 0.613

---best\_model\_release epoch 221,train\_loss 0.472,train\_acc 0.980,valid\_acc 0.614

---best\_model\_release epoch 224,train\_loss 0.475,train\_acc 0.980,valid\_acc 0.614

epoch 301,train\_loss 1.205,train\_acc 0.967,valid\_acc 0.601

---best\_model\_release epoch 314,train\_loss 0.412,train\_acc 0.981,valid\_acc 0.616

---best\_model\_release epoch 320,train\_loss 0.420,train\_acc 0.981,valid\_acc

0.617

---best\_model\_release epoch 354,train\_loss 0.399,train\_acc 0.981,valid\_acc 0.618

---best\_model\_release epoch 377,train\_loss 0.396,train\_acc 0.982,valid\_acc

0.618

epoch 401,train\_loss 1.337,train\_acc 0.960,valid\_acc 0.585

---best\_model\_release epoch 461,train\_loss 0.386,train\_acc 0.981,valid\_acc 0.619

epoch 501,train\_loss 0.358,train\_acc 0.982,valid\_acc 0.615 epoch 601,train\_loss 0.331,train\_acc 0.983,valid\_acc 0.616

---best\_model\_release epoch 627,train\_loss 0.323,train\_acc 0.984,valid\_acc 0.620

---best\_model\_release epoch 656,train\_loss 0.321,train\_acc 0.983,valid\_acc 0.620

epoch 701,train\_loss 0.332,train\_acc 0.982,valid\_acc 0.615 epoch 801,train\_loss 0.296,train\_acc 0.983,valid\_acc 0.614 epoch 901,train\_loss 0.496,train\_acc 0.982,valid\_acc 0.600 epoch 1001,train\_loss 0.273,train\_acc 0.984,valid\_acc 0.607 epoch 1101,train\_loss 0.316,train\_acc 0.983,valid\_acc 0.613 epoch 1201,train\_loss 0.256,train\_acc 0.985,valid\_acc 0.611

---best\_model\_release epoch 1,train\_loss 16.894,train\_acc 0.547,valid\_acc 0.752

epoch 1,train\_loss 16.894,train\_acc 0.547,valid\_acc 0.752

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ---best\_model\_release  0.820 | epoch | 2,train\_loss | 9.617,train\_acc | 0.769,valid\_acc |
| ---best\_model\_release  0.835 | epoch | 3,train\_loss | 7.924,train\_acc | 0.821,valid\_acc |
| ---best\_model\_release  0.852 | epoch | 4,train\_loss | 7.115,train\_acc | 0.835,valid\_acc |
| ---best\_model\_release  0.869 | epoch | 5,train\_loss | 6.803,train\_acc | 0.843,valid\_acc |
| ---best\_model\_release  0.882 | epoch | 6,train\_loss | 6.462,train\_acc | 0.848,valid\_acc |
| ---best\_model\_release  0.883 | epoch | 8,train\_loss | 6.158,train\_acc | 0.853,valid\_acc |
| ---best\_model\_release  0.884 | epoch | 10,train\_loss | 5.898,train\_acc | 0.859,valid\_acc |
| ---best\_model\_release  0.885 | epoch | 11,train\_loss | 5.609,train\_acc | 0.865,valid\_acc |
| ---best\_model\_release  0.885 | epoch | 12,train\_loss | 5.590,train\_acc | 0.865,valid\_acc |
| ---best\_model\_release  0.889 | epoch | 14,train\_loss | 5.436,train\_acc | 0.868,valid\_acc |
| ---best\_model\_release  0.889 | epoch | 16,train\_loss | 5.251,train\_acc | 0.871,valid\_acc |
| ---best\_model\_release  0.890 | epoch | 17,train\_loss | 5.170,train\_acc | 0.876,valid\_acc |
| ---best\_model\_release  0.891 | epoch | 18,train\_loss | 5.165,train\_acc | 0.871,valid\_acc |
| ---best\_model\_release  0.893 | epoch | 19,train\_loss | 5.134,train\_acc | 0.874,valid\_acc |
| ---best\_model\_release  0.894 | epoch | 21,train\_loss | 5.068,train\_acc | 0.874,valid\_acc |
| ---best\_model\_release  0.895 | epoch | 24,train\_loss | 4.893,train\_acc | 0.875,valid\_acc |
| ---best\_model\_release  0.896 | epoch | 25,train\_loss | 4.865,train\_acc | 0.879,valid\_acc |
| ---best\_model\_release  0.899 | epoch | 29,train\_loss | 4.816,train\_acc | 0.882,valid\_acc |
| ---best\_model\_release | epoch | 33,train\_loss | 4.705,train\_acc | 0.882,valid\_acc |

0.899

---best\_model\_release epoch 35,train\_loss 4.641,train\_acc 0.882,valid\_acc 0.900

---best\_model\_release epoch 36,train\_loss 4.556,train\_acc 0.882,valid\_acc 0.901

---best\_model\_release epoch 38,train\_loss 4.532,train\_acc 0.885,valid\_acc 0.902

---best\_model\_release epoch 40,train\_loss 4.367,train\_acc 0.889,valid\_acc 0.903

---best\_model\_release epoch 41,train\_loss 4.398,train\_acc 0.885,valid\_acc 0.903

---best\_model\_release epoch 42,train\_loss 4.431,train\_acc 0.888,valid\_acc 0.905

---best\_model\_release epoch 43,train\_loss 4.353,train\_acc 0.890,valid\_acc 0.906

---best\_model\_release epoch 44,train\_loss 4.389,train\_acc 0.887,valid\_acc 0.909

---best\_model\_release epoch 45,train\_loss 4.277,train\_acc 0.895,valid\_acc 0.910

---best\_model\_release epoch 46,train\_loss 4.358,train\_acc 0.892,valid\_acc 0.912

---best\_model\_release epoch 47,train\_loss 4.237,train\_acc 0.895,valid\_acc 0.913

---best\_model\_release epoch 50,train\_loss 4.069,train\_acc 0.899,valid\_acc 0.914

---best\_model\_release epoch 52,train\_loss 4.161,train\_acc 0.894,valid\_acc 0.914

---best\_model\_release epoch 53,train\_loss 4.042,train\_acc 0.896,valid\_acc 0.915

---best\_model\_release epoch 56,train\_loss 4.018,train\_acc 0.899,valid\_acc 0.915

---best\_model\_release epoch 73,train\_loss 3.689,train\_acc 0.903,valid\_acc 0.916

---best\_model\_release epoch 78,train\_loss 3.617,train\_acc 0.906,valid\_acc 0.917

---best\_model\_release epoch 79,train\_loss 3.624,train\_acc 0.905,valid\_acc 0.917

---best\_model\_release epoch 82,train\_loss 3.653,train\_acc 0.905,valid\_acc 0.918

---best\_model\_release epoch 86,train\_loss 3.537,train\_acc 0.909,valid\_acc 0.919

---best\_model\_release epoch 87,train\_loss 3.558,train\_acc 0.907,valid\_acc 0.919

---best\_model\_release epoch 88,train\_loss 3.508,train\_acc 0.909,valid\_acc

0.920

---best\_model\_release epoch 90,train\_loss 3.530,train\_acc 0.909,valid\_acc 0.920

epoch 101,train\_loss 3.321,train\_acc 0.910,valid\_acc 0.918

---best\_model\_release epoch 105,train\_loss 3.310,train\_acc 0.913,valid\_acc 0.922

---best\_model\_release epoch 147,train\_loss 3.044,train\_acc 0.918,valid\_acc 0.923

---best\_model\_release epoch 160,train\_loss 2.851,train\_acc 0.924,valid\_acc 0.923

---best\_model\_release epoch 192,train\_loss 2.719,train\_acc 0.924,valid\_acc 0.924

epoch 201,train\_loss 2.702,train\_acc 0.923,valid\_acc 0.919

---best\_model\_release epoch 236,train\_loss 2.573,train\_acc 0.927,valid\_acc 0.925

---best\_model\_release epoch 241,train\_loss 2.538,train\_acc 0.929,valid\_acc

0.926

epoch 301,train\_loss 2.324,train\_acc 0.935,valid\_acc 0.924

---best\_model\_release epoch 361,train\_loss 2.237,train\_acc 0.937,valid\_acc 0.927

epoch 401,train\_loss 2.193,train\_acc 0.937,valid\_acc 0.924 epoch 501,train\_loss 2.040,train\_acc 0.940,valid\_acc 0.924 epoch 601,train\_loss 1.938,train\_acc 0.944,valid\_acc 0.922 epoch 701,train\_loss 1.934,train\_acc 0.943,valid\_acc 0.925 epoch 801,train\_loss 1.861,train\_acc 0.946,valid\_acc 0.922 epoch 901,train\_loss 1.887,train\_acc 0.944,valid\_acc 0.923

# 四、 实验分析

详细分析实验结果，包括程序中使用数据的基本情况、程序的正确性验证、实验的效果对比。

## 数据的基本情况

使用上次实验的两万条数据，经过清洗后用于训练模型

其中文本数据在编码和计算相关性后转为向量用于训练，缺失值用0补全并且设置额外的3个维度用于表示缺失

## 程序的正确性验证

模型在训练集，测试集上的准确率

---best\_model\_release epoch 656,train\_loss 0.321,train\_acc 0.983,valid\_acc

0.620

---best\_model\_release epoch 361,train\_loss 2.237,train\_acc 0.937,valid\_acc

0.927

## 实验结果检测

测试代码

m\_Go\_net\_test = Go\_net()

m\_Go\_net\_test.load\_state\_dict(torch.load('go\_re/m\_Go\_netbest\_model.pkl',map

\_location=device)) m\_Go\_net\_test.to(torch.bfloat16).to(device)

val\_data,val\_label = torch.load('data.data')

dataset = data.TensorDataset(val\_data,val\_label)

val\_loader = data.DataLoader(dataset, batch\_size=1, shuffle=True)

def test(data):

with torch.no\_grad(): m\_Go\_net\_test.eval()

# 确保数据转换为BFloat16

data = data.to(torch.bfloat16)

return m\_Go\_net\_test(data)

t = 0

acc = 0

for x,y in val\_loader: if t == 100:

break else:

t+=1

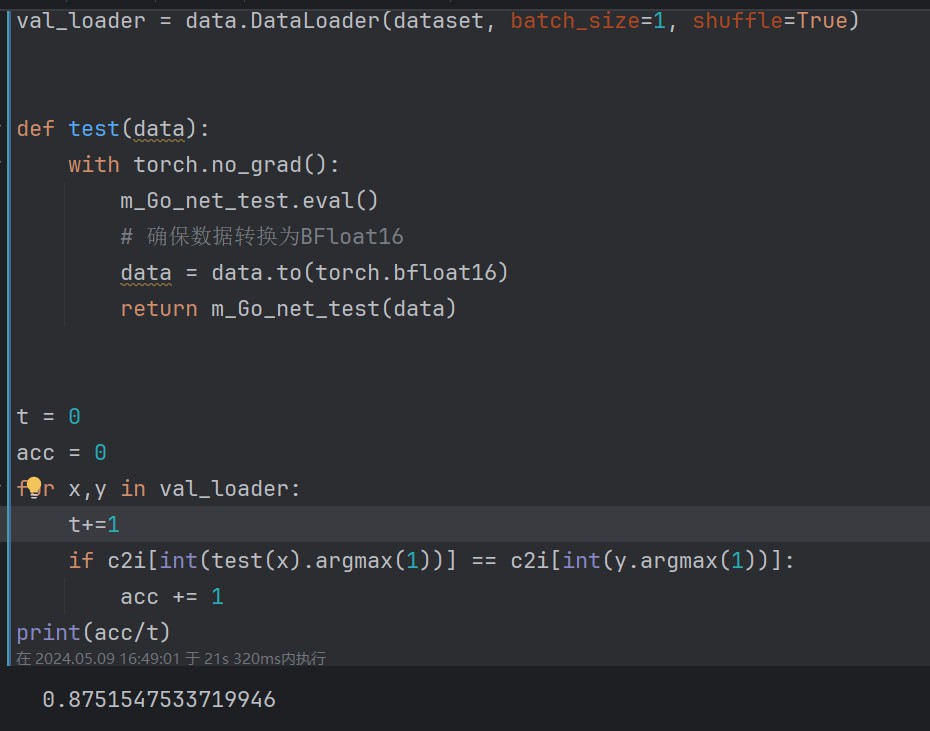
print(c2i[int(test(x).argmax(1))],' ',c2i[int(y.argmax(1))])

if c2i[int(test(x).argmax(1))] == c2i[int(y.argmax(1))]: acc += 1

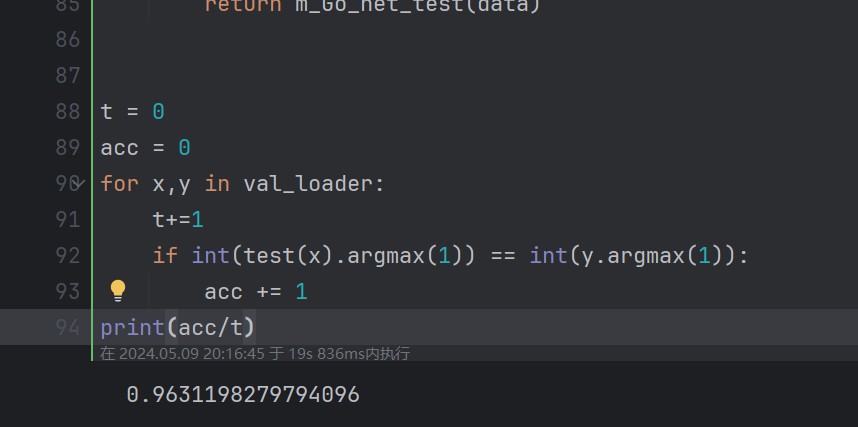
print(acc/t)

在全体数据上的准确率：

小类



大类：



# 五、 问题和解决方案

详细列举实验中出现的问题以及自己所给出的解决方法。

## 数据清洗

### 文本

上次实验获得的数据中有文本信息，对此先使用网络上的模型将其转为向量，但是在训练过程中发现效果较差，推测可能是网络难以拟合词嵌入后的向量

解决：

手动将词嵌入后的向量与类别进行计算，然后计算他们之间的相关性在这样处理后，模型的测试集准确率由0.4提升到0.6

## 网络搭建

参考：《动手学深度学习》搭建svm和并行网络

## 训练调参

尝试使用svm和并行网络，初始时svm表现较好，但是在手动计算相关性后并行网络的效果优于svm

最终在svm上调出了小类0.604的测试集准确率，并行网络上调出了小类0.620，大类

0.927的测试集准确率

且svm更容易出现过拟合

svm -- train\_acc 0.983,valid\_acc 0.620

并行 -- train\_acc 0.702,valid\_acc 0.604

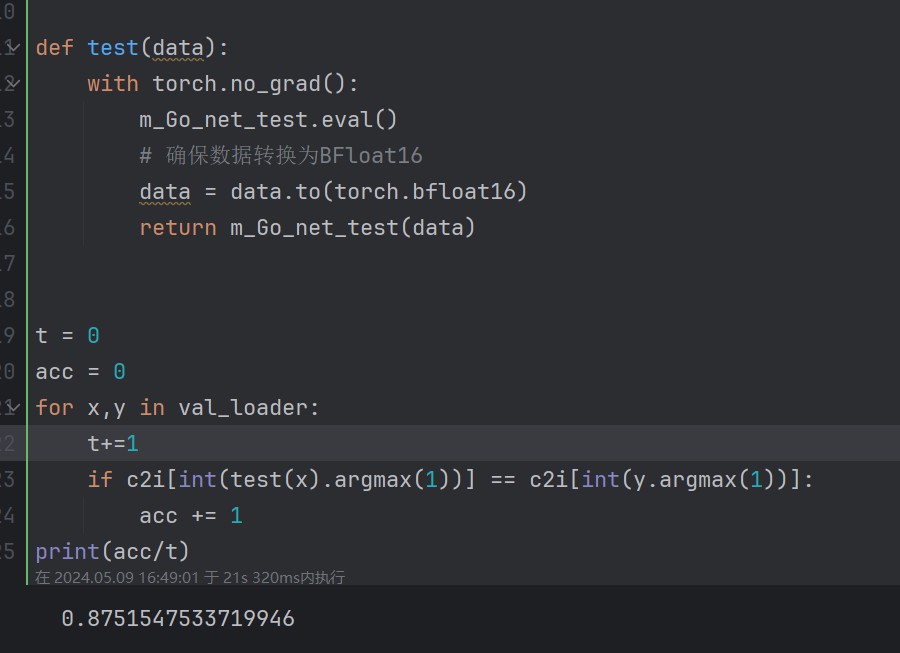
调参时网络层数过多容易导致过拟合，过少会导致欠拟合

初始的网络参数设置过少，导致训练集，测试集准确率都偏低，且dropout 层参数设置过大影响最终结果

多次改变网络层架构和lr ， batch\_size ， weight\_decay=1e-4 等参数后发现对于svm，三层网络可以得到比较好的结果，而对并行神经网络4层较好，还发现并行神经网络的每个输出大小最好不要低于输入大小

## 结果验证

编写代码在全部数据上进行了验证，准确率约为0.88



# 六、 实验总结

总结实验效果，还有哪些地方可以继续改进。

## 实验效果

训练的模型在验证集上分别达到了0.62，0.927的准确率

### 数据

还可以基于网页提取更多特征信息，仅靠过去收集的五个信息对46类结果预测较为困难例如可以记录网页中的图片信息并基于此获得新的输入

基于文本信息获得更多维度的特征，例如可以计算数据简介中关键字出现的次数或者图片数量，特定标签数量等

此外，从网站上抽取的信息中包含上个月下载次数等与时间相关的信息，在检查时发现这一数据变化很大，可以考虑以此建立时序或者采用其平均值来进行计算

网站上还有很多数据没有大小类标签，可以考虑将他们也加入训练集

## 网络

可以尝试构建结构更好的网络，尽管使用了正则化， dropout 等手段，但实验中所选的网络在训练后期均出现了严重的过拟合现象，