**关于TPN模型精度对齐的测试报告**

1. 测试环境

Paddlepaddle-gpu：1.8.3

Pytorch：1.1.0

Torchvision：0.4.0

Python：3.5

1. 测试方法及步骤

1.建立两个工程：包含作者源码的torch工程和自己新写的paddle模型工程

2.选定一个配置（如r50f32s2）将原torch工程中生成的模型保存为tpn.pkl的文件；

3.在自己新建的paddle工程中载入模型文件，生成paddle和torch两个模型，使用torch模型的参数来初始化paddle的模型（需要进行参数转换同时保证两个模型的配置相同）；

4.随机生成一组输入数据，分别送入paddle和torch的两个模型中进行正向运算，获得运行结果，比较两个模型的输出，统计输出结果的差值的绝对值的平均值。比较的计算结果包括两个loss（aux\_loss,cls\_loss）和cls\_head模块最后一个全连接层的输出（softmax层的输入，维度为（-1,400））；

5.重复步骤4，观察输出的平均误差的变化并记录

6.更换配置文件重复步骤2-5

1. 测试结果
2. 使用r50f32s2的配置文件进行测试，随机输入数据的维度为（2,1,96,224,224），即模拟batchsize为2的情况，多次测试后其平均绝对差值变化很小，约为2.8×10-9

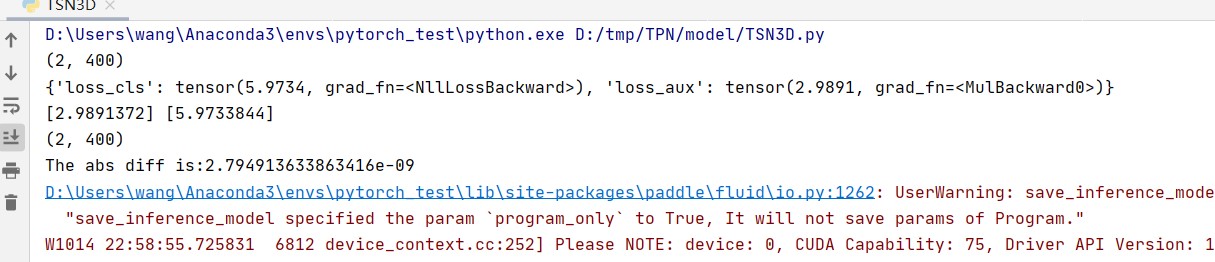


图1 r50f32s2配置下的精度对齐结果截图

1. 使用r50f8s8的配置文件进行测试，随机输入的数据维度为（2,1,24,224,224），多次测试后其平均绝对差值约为2.6×10-9

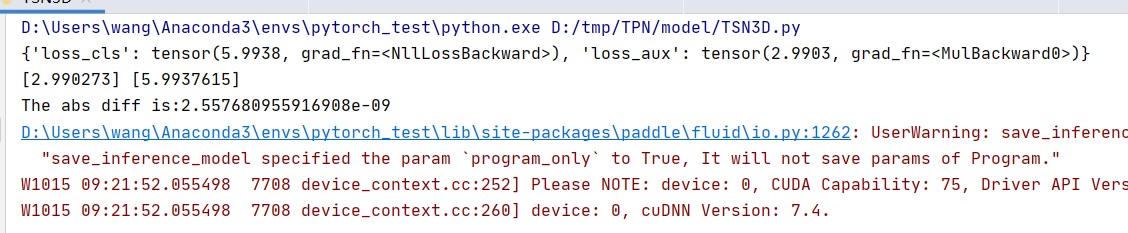


图2 r50f8s8配置下的精度对齐结果截图

1. 结论

通过测试可以看出paddle模型在正向运算时与torch模型的输出误差约在10-9的数量

级，可以忽略不计。证明paddle模型与原torch模型计算结构相同，精度已对齐。

1. 遇到的问题与解决办法
2. Dropout层的数据对齐

Paddle中dropout层的默认实现方法与torch中不一样，torch默认是在训练时将未丢

弃的参数等比例放大，在测试时保持参数不变；而paddle中默认是在训练时保存参数不变，在测试时将参数等比例减小，这样导致输出精度无法对齐。需要在paddle中调用fluid.layers.dropout接口时设置参数dropout\_implementation='upscale\_in\_train'。经测试如果使用默认参数，误差的绝对均值会增大到10-3

1. 上采样（upsampling）模块的实现

原始工程中调用了torch中的插值函数（原工程/models/necks/tpn.py line 105）：

torch.nn.functional. interpolate(x, scale\_factor=self.scale, mode='nearest')

不同paddle版本下转换该函数的方法不一样，经过测试都是可以达到精度对齐的目的，实际工程中由于平台版本限制使用了第二种方法。

（1）1.8.3版本

paddle中对应的函数为：fluid.layers.interpolate（），但是当输入为5维张量，插值类型参数设置resample=‘nearest’时，paddle运行会报错如下：

in interpolate "'BILINEAR', 'BICUBIC' and 'NEAREST' only support 4-D tensor."

只能将插值类型参数设置为‘TRALINEAR’才能正常运行，由于目前用到的配置中该工程的上采样缩放系数均为1，所以两种插值方法的结果是一样的。

（2）1.7.2 版本

由于平台多卡环境只支持1.7.2的版本，发现该版本下paddle没有对应的插值函数，为此自己写了一个简单的插值函数如下：

def Upsamping(self, input, scale=2):  
 if(scale==1):  
 output = input  
 elif(scale==2):  
 output = fluid.layers.expand(input,[1,1,2,1,1])  
 else:  
 output=None  
 return output

当上采样缩放系数为1时输出等于输入，当缩放系数为2时输出等于输入在时间维上的复制，等同于最近邻插值。实际上本工程配置文件的上采样缩放系数均为1，该模块等同于恒等模块。

1. 原工程中存在的bug

实际调试时发现原工程代码存在一个逻辑bug，按照论文中的实现方法，作者在进行多

层数据融合时采用了parallel flow的并行融合策略，但是在代码实现时，作者错误的使用了列表赋值操作（原工程/models/necks/tpn.py line 319）：

temporal\_modulation\_outs = outs

此处outs是一个张量列表，这种复制方式导致temporal\_modulation\_outs仅仅成为了outs的一个别名（即浅拷贝），无法起到保存outs副本的作用。当对outs进行操作时，temporal\_modulation\_outs指向的内容也发生了变化。最终造成的影响就是作者计划使用并行的融合策略，结果变成了cascade flow与topdown flow的结果进行了融合。这个bug在原github工程的issue17上也有人提到了。

虽然此处存在bug，但是根据作者在issue的回答来看对最终精度影响不大，因而在paddle实现中依然保持与作者相同。

**附录**

1. 原torch工程中保存模型的代码

在tools/train\_recognizer.py 的main函数中增加以下代码：

model = build\_recognizer(  
 cfg.model, train\_cfg=cfg.train\_cfg, test\_cfg=cfg.test\_cfg)  
  
print(model)  
torch.save(model,**'./tpn.pkl'**)

1. 在新建的paddle工程中精度对齐的测试代码

在 model/TSN3D.py 中增加以下测试代码：

if \_\_name\_\_ == **"\_\_main\_\_"**:  
 startup = fluid.Program()  
 train\_prog = fluid.Program()  
 with fluid.program\_guard(train\_prog, startup):  
 with fluid.unique\_name.guard():  
 *# input = fluid.data(name='image', shape=(2,1,96,224,224), dtype='float32')* input = fluid.data(name=**'image'**, shape=(2, 1, 24, 224, 224), dtype=**'float32'**)  
 label = fluid.data(name=**'label'**, shape=(2,1), dtype=**'int64'**)  
 model = TSN3D(config=r50f32s2,is\_training=False)  
 fetch\_list = model.net(input,label)  
  
 place = fluid.CUDAPlace(0)exe = fluid.Executor(place)  
 exe.run(startup)  
 para\_list = train\_prog.all\_parameters()  
 f = open(**'para\_p.txt'**,**'w'**)  
 for i in range(len(para\_list)):  
 f.write(para\_list[i].name+**'**\n**'**)  
 f.close()  
  
#读取torch的模型参数并转换成paddle的模型参数  
 pretrain\_para,model = get\_pretrain\_para()  
 para\_list = train\_prog.all\_parameters()  
 pretrain\_para\_new = OrderedDict()  
 i = 0  
 for key in pretrain\_para:  
 pretrain\_para\_new[para\_list[i].name] = pretrain\_para[key]  
 i = i+1  
 fluid.io.set\_program\_state(train\_prog,pretrain\_para\_new)  
 *# data = np.random.randn(2,1,96,224,224).astype('float32')* data = np.random.randn(2, 1, 24, 224, 224).astype(**'float32'**)  
 *# data = np.ones(shape=(2,1,24,224,224),dtype='float32')* label = np.array([[10],[20]]).astype(**'int64'**)  
 outs,\_,aux\_loss,cls\_loss = exe.run(train\_prog, fetch\_list=fetch\_list,feed={**'image'**:data,**'label'**:label})  
 *#根据paddle的数据格式在torch中做对应变换* x = torch.from\_numpy(data)  
 l = torch.from\_numpy(label)  
 *# x = torch.reshape(x,(2,1,32,3,224,224))* x = torch.reshape(x, (2, 1, 8, 3, 224, 224))  
 x = torch.Tensor.permute(x,(0,1,3,2,4,5))  
  
 model.eval()  
 loss ,cls\_score= model([1], True, img\_group\_0=x, gt\_label=l)  
 cls\_score = torch.detach(cls\_score).numpy()  
 *# test2 = torch.detach(test2).numpy()* print(loss)  
 print(aux\_loss,cls\_loss)  
 *# print(cls\_score.shape)* diff = np.absolute(outs-cls\_score)  
 mean = np.mean(diff)  
 print(**"The abs diff is:{}"**.format(mean))

def get\_pretrain\_para(path=**'../pretrain/tpn.pkl'**):  
 model = torch.load(path)  
 state\_dict = model.state\_dict()  
 f = open(**'para\_pretrain.txt'**, **'w'**)  
 for key in state\_dict.keys():  
 if not(**'tracked'** in key):  
 f.write(key + **'**\n**'**)  
 f.close  
 sub\_key = []  
 *# 取出backbone的参数* for key in state\_dict.keys():  
 if **'backbone'** in key:  
 sub\_key.append(key)  
 *# 查找conv2的位置* conv2\_loc = []  
 for i in range(len(sub\_key)):  
 if **'conv2'** in sub\_key[i]:  
 conv2\_loc.append(i)  
 *# 将conv2的位置往后移动5个* for i in range(len(conv2\_loc)):  
 index = conv2\_loc[i]  
 item = sub\_key[index]  
 sub\_key.pop(index)  
 sub\_key.insert(index + 5, item)  
 *# 构建新的backbone字典* backbone\_para = OrderedDict()  
 for i in range(len(sub\_key)):  
 if not (**'tracked'** in sub\_key[i]):  
 backbone\_para[sub\_key[i]] = state\_dict[sub\_key[i]].numpy()  
 del state\_dict[sub\_key[i]]  
  
 *#构建TPN字典  
 #Spatialmodule 字段* TPN\_para = OrderedDict()  
 for key in state\_dict.keys():  
 if **'spatial\_modulation'** in key:  
 if not(**'tracked'** in key):  
 TPN\_para[key] = state\_dict[key].numpy()  
  
 *#构建Temopralmodule字段* for key in state\_dict.keys():  
 if **'temporal\_modulation'** in key:  
 if not (**'tracked'** in key):  
 TPN\_para[key] = state\_dict[key].numpy()  
  
 *#构建top\_down字段* for key in state\_dict.keys():  
 if **'level\_fusion\_op2'** in key:  
 if not (**'tracked'** in key):  
 TPN\_para[key] = state\_dict[key].numpy()  
  
 *#构建downsamping字段* for key in state\_dict.keys():  
 if **'downsampling\_ops'** in key:  
 if not (**'tracked'** in key):  
 TPN\_para[key] = state\_dict[key].numpy()  
  
 *#构建down\_top字段* for key in state\_dict.keys():  
 if **'level\_fusion\_op'** in key:  
 if not (**'tracked'** in key):  
 TPN\_para[key] = state\_dict[key].numpy()  
  
 *#构建pyramid\_fusion字段* for key in state\_dict.keys():  
 if **'pyramid\_fusion\_op'** in key:  
 if not (**'tracked'** in key):  
 TPN\_para[key] = state\_dict[key].numpy()  
  
 *#构建aux\_head字段* for key in state\_dict.keys():  
 if **'aux\_head'** in key:  
 if not (**'tracked'** in key):  
 if (**'fc.weight'** in key):  
 TPN\_para[key] = state\_dict[key].numpy().T  
 else:  
 TPN\_para[key] = state\_dict[key].numpy()  
  
 *#构建cls\_head字段* for key in state\_dict.keys():  
 if **'cls\_head'** in key:  
 if (**'weight'** in key):  
 TPN\_para[key] = state\_dict[key].numpy().T  
 else:  
 TPN\_para[key] = state\_dict[key].numpy()  
  
  
 para\_dict = OrderedDict()  
 para\_dict.update(backbone\_para)  
 para\_dict.update(TPN\_para)  
  
  
 f = open(**'para\_pretrain'**, **'w'**)  
 for key in para\_dict.keys():  
 f.write(key + **'**\n**'**)  
 f.close  
 return para\_dict,model