初始输入时：



输入：x

N = 2 \* batch\_size, 一帧里面有两个人；

C = 3， channel = 3;

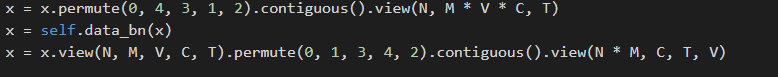
T = 150, #time\_frame = 150;

V = 25, #vertex = 25;

M = 60, #action\_class = 60;

x.size = (2, 3, 150, 25, 60)

维度变换和bn：

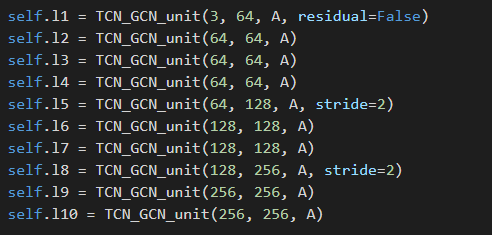


x.size = (2, 60, 25, 3, 150) 合并成 (2, 4500, 150)

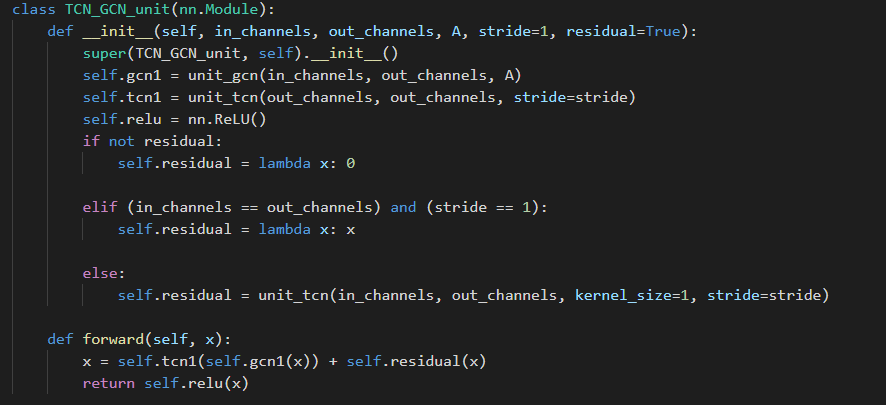
data\_bn = nn.BatchNorm1d (N \* C \* V) 即data\_bn = nn.BatchNorm1d (150) 沿着第2维的方向做bn

再次展开：x.size = (2, 60, 25, 3, 150) -> x.size() = (2, 60, 3, 150, 25) -> x.size = (120, 3, 150, 25)

GCN结构：



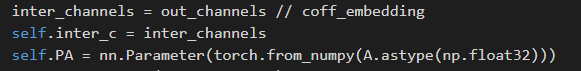
L1层运算：



无残差模块，stride = 1, in\_channel = 3, out\_channel = 64

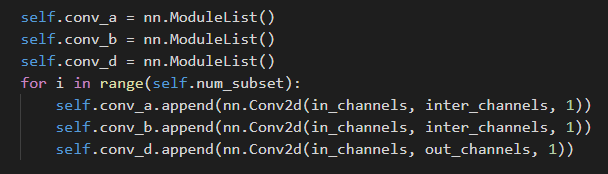
L1.gcn: in\_channel = 3, out\_channel = 64,

input\_x.size = (120, 3, 150, 25)



Inter\_c = 64 // 4 = 16

PA = B, 对应原论文中f\_out = w \* f\_in \* (A + B + C), A为原始邻接矩阵



Conv\_a: conv2d(in\_c = 3, out\_c = 16, kernel = 1, stride = 1), 连续三次

Conv\_b: conv2d(in\_c = 3, out\_c = 16, kernel = 1, stride = 1), 连续三次

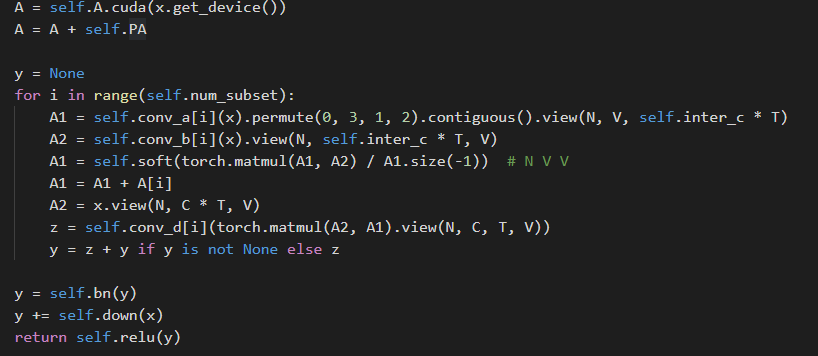
Conv\_d: conv2d(in\_c = 3, out\_c = 64, kernel = 1, stride = 1), 连续三次

Conv\_a的卷积参数size：(1, 3, 1, 1) \* 16 filter \* 3 loops

Conv\_b的卷积参数size：(1, 3, 1, 1) \* 16 filter \* 3 loops

Conv\_c的卷积参数size：(1, 3, 1, 1) \* 64 filter \* 3 loops

Gcn\_forward:



A = A + self.PA: 执行A+ B

Conv\_a[0](x).size = (120, 16, 150, 25) -> (120, 25, 16, 150) -> 60 x (2, 25, 2400) ???

卷积核尺寸：(1, 3, 1, 1), 16个filter, 120\*25\*150\*16 = 7.2M次向量乘加, 21.6M乘&14.4M加, 向量尺寸1x3

Conv\_b[0](x).size = (120, 16, 150, 25) -> (120, 25, 16, 150) -> 60 x (2, 2400, 25) ???

卷积核尺寸：(1, 3, 1, 1), 16个filter, 120\*25\*150\*16 = 7.2M次向量乘加, 21.6M乘& 14.4M加, 向量尺寸1x3

Matmul(A1, A2): 4次 (25, 2400) \* (2400, 25) 的矩阵乘法，2500次向量乘加，6M乘&6M加，向量尺寸1x2400，输出尺寸(4, 25, 25) ???

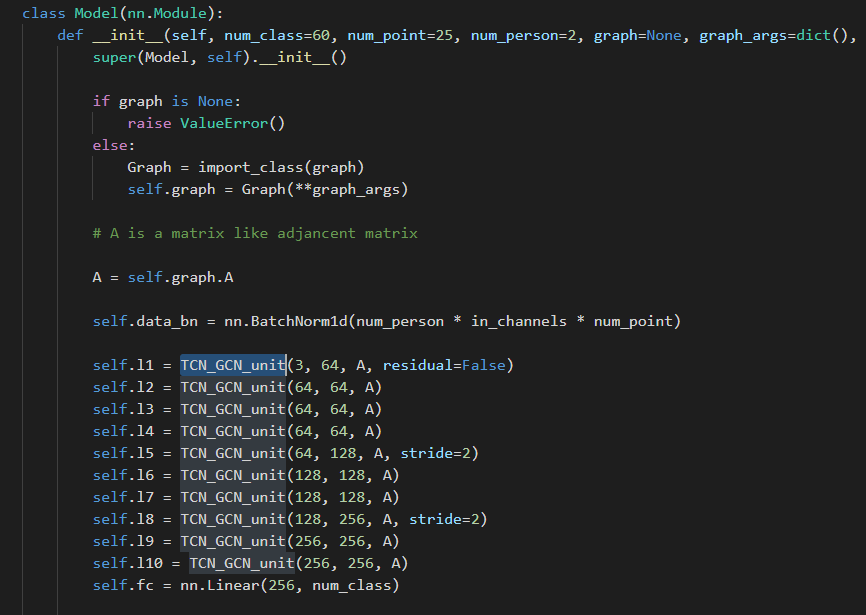
A1 = Softmax(Matmul(A1, A2)): 得到C矩阵

A1 = A1 +A[i]: 执行A + B + C的操作

A2.size() = 60 x (2, 450, 25) ???

2020.10.26 1935 补充

Model中由10个TCN\_GCN\_unit模块组成，对应网络中的10层（block）



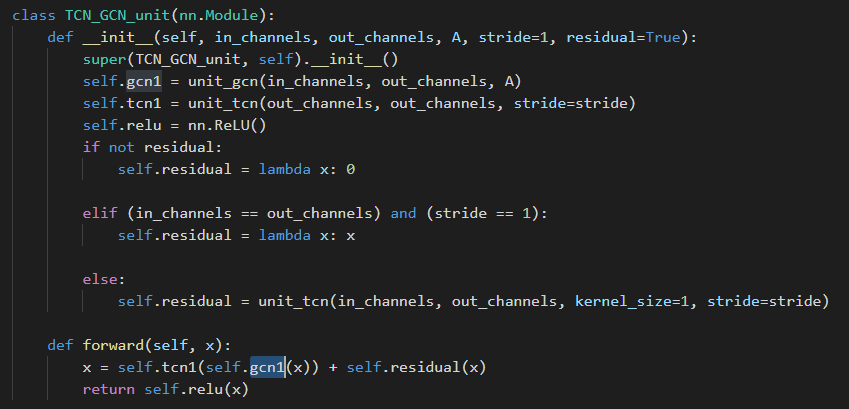
TCN\_GCN\_unit模块：

先做gcn1()

再做tcn1()

融合残差+residual()

最后relu()



Unit\_gcn()模块：

PA和A的size是一样，不过PA的值全被初始化为e-6.

class unit\_gcn(nn.Module):

    def \_\_init\_\_(self, in\_channels, out\_channels, A, coff\_embedding=4, num\_subset=3):

        super(unit\_gcn, self).\_\_init\_\_()

        inter\_channels = out\_channels // coff\_embedding

        self.inter\_c = inter\_channels

        self.PA = nn.Parameter(torch.from\_numpy(A.astype(np.float32)))

        nn.init.constant(self.PA, 1e-6)

        self.A = Variable(torch.from\_numpy(A.astype(np.float32)), requires\_grad=False)

        self.num\_subset = num\_subset

        self.conv\_a = nn.ModuleList()

        self.conv\_b = nn.ModuleList()

        self.conv\_d = nn.ModuleList()

        for i in range(self.num\_subset):

            self.conv\_a.append(nn.Conv2d(in\_channels, inter\_channels, 1))

            self.conv\_b.append(nn.Conv2d(in\_channels, inter\_channels, 1))

            self.conv\_d.append(nn.Conv2d(in\_channels, out\_channels, 1))

        if in\_channels != out\_channels:

            self.down = nn.Sequential(

                nn.Conv2d(in\_channels, out\_channels, 1),

                nn.BatchNorm2d(out\_channels)

            )

        else:

            self.down = lambda x: x

        self.bn = nn.BatchNorm2d(out\_channels)

        self.soft = nn.Softmax(-2)

        self.relu = nn.ReLU()

        for m in self.modules():

            if isinstance(m, nn.Conv2d):

                conv\_init(m)

            elif isinstance(m, nn.BatchNorm2d):

                bn\_init(m, 1)

        bn\_init(self.bn, 1e-6)

        for i in range(self.num\_subset):

            conv\_branch\_init(self.conv\_d[i], self.num\_subset)

    def forward(self, x):

        N, C, T, V = x.size()

        A = self.A.cuda(x.get\_device())

        A = A + self.PA

        y = None

        for i in range(self.num\_subset):

            A1 = self.conv\_a[i](x).permute(0, 3, 1, 2).contiguous().view(N, V, self.inter\_c \* T)

            A2 = self.conv\_b[i](x).view(N, self.inter\_c \* T, V)

            A1 = self.soft(torch.matmul(A1, A2) / A1.size(-1))  # N V V

            A1 = A1 + A[i]

            A2 = x.view(N, C \* T, V)

            z = self.conv\_d[i](torch.matmul(A2, A1).view(N, C, T, V))

            y = z + y if y is not None else z

        y = self.bn(y)

        y += self.down(x)

        return self.relu(y)