**FS-STBP融合结构测试总结**

**一、说明**

**本阶段的工作基于Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes 和Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks 两篇文献，将第一篇文章的FS神经元模型和第二篇文章的STBP梯度替代算法相融合，实现了基于FS神经元的直接监督学习。**

**（1）K值和alpha值的改进情况**

**原本的FS神经元模型每一层的K值和alpha值为定值，但是实际上，各层神经元平均发射脉冲数(平均单个神经元发的脉冲数)的分布是不均匀的。某些层数发射的脉冲数较多，某些层数发射的脉冲数较少，一般呈现出前层多后层少的特点。**

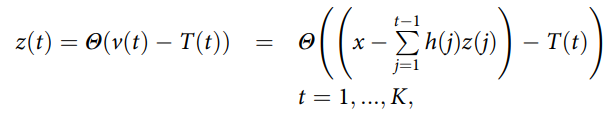
**在FS网络中K值如果过大，则平均脉冲数一般会增大，从而使SNN芯片能耗增大。同时，由于FS神经元编码的误差为alpha/2^k，K值过小也会导致误差的增大。**

K = [4, 3, 3, 3, 2]  
alpha = [1, 3, 4, 2, 1]  
  
d1 = H1 = T1 = alpha[0] \* 2 \*\* (-K[0]) \* np.array([float(2 \*\* (K[0] - i)) for i in range(1, K[0] + 1)]).astype(np.float32)  
d2 = H2 = T2 = alpha[1] \* 2 \*\* (-K[1]) \* np.array([float(2 \*\* (K[1] - i)) for i in range(1, K[1] + 1)]).astype(np.float32)  
d3 = H3 = T3 = alpha[2] \* 2 \*\* (-K[2]) \* np.array([float(2 \*\* (K[2] - i)) for i in range(1, K[2] + 1)]).astype(np.float32)  
d4 = H4 = T4 = alpha[3] \* 2 \*\* (-K[3]) \* np.array([float(2 \*\* (K[3] - i)) for i in range(1, K[3] + 1)]).astype(np.float32)  
d5 = H5 = T5 = alpha[4] \* 2 \*\* (-K[4]) \* np.array([float(2 \*\* (K[4] - i)) for i in range(1, K[4] + 1)]).astype(np.float32)

**因此我对融合后的代码进行了修改，使得每一层的K值和alpha值都可以灵活调控，从而寻找更优的超参数设置方案。**

**（2）脉冲发射函数的修改情况**

**原论文中的一个例子是这样的**

****

**K=10，T(t)=h(t)=d(t)=先以x=10为例：**

**z(1)=Θ(10-0- )=0, 🡪d(1)z(1)=0;**

**z(2)=Θ(10-0- )=0, 🡪d(2)z(2)=0;**

**……**

**z(6)=Θ(10-0- )=0, 🡪d(6)z(6)=0;**

**z(7)=Θ(10-0- )=1, 🡪d(7)z(7)=8;**

**z(8)=Θ(10-8- )=0, 🡪d(8)z(8)=0;**

**z(9)=Θ(10-8- )=1, 🡪d(9)z(9)=2;**

**z(10)=Θ(10-8- 2-)=0, 🡪d(10)z(10)=0;**

**原来的FS神经元代码中，每一个step权重都要乘0.5（原论文中T(t)=h(t)=d(t)=α，在代码中d（t）体现为每个step中weight\*0.5）**

# fs-neuron 更新膜电位  
def fs\_mem\_update(ops, x, mem): # self.fc1, h0\_spike, h1(zeros)  
 mem = mem + ops(x)  
 return mem # h1  
  
  
# fs-neuron 编码  
def fs\_coding(mem, decay, th): # h0, h[step], T[step]  
 z = act\_fun(mem - th)   
 mem = mem - decay \* z  
 return mem, z # h0, h0\_spike

for step in range(K[0]):  
 # 输入值编码  
 h0, h0\_spike = fs\_coding(h0, H1[step], T1[step])  
 # 更新c1\_mem  
 h1 = fs\_mem\_update(self.fc1, h0\_spike, h1)  
 self.fc1.weight.data = self.fc1.weight.data \* 0.5

**但是如果加入STBP算法会使反传失败，无法训练SNN，测试得到的准确率只有11%。(目前为什么这样写代码无法训练的原因还没有找到)**

**所以此次代码进行了修改：**

# fs-neuron 更新膜电位  
def fs\_mem\_update(ops, x, mem): # self.fc1, h0\_spike, h1(zeros)  
 mem = mem + ops(x)  
 return mem # h1  
  
  
# fs-neuron 编码  
def fs\_coding(mem, decay, th): # h0, h[step], T[step]  
 z = act\_fun(mem - th) \* 0.5  
 mem = mem - decay \* z  
 return mem, z # h0, h0\_spike

for step in range(K[0]):  
 # 输入值编码  
 h0, h0\_spike = fs\_coding(h0, H1[step], T1[step])  
 # 更新c1\_mem  
 h1 = fs\_mem\_update(self.fc1, h0\_spike, h1)  
 # self.fc1.weight.data = self.fc1.weight.data \* 0.5

#在weight上乘0.5会使SNN无法训练，因此改为0.5乘到act\_fun后面  
 for step in range(K[1]):  
 # c1\_mem编码  
 h1, h1\_spike = fs\_coding(h1, H2[step], T2[step])  
 h1\_spikes += h1\_spike  
 # x = F.avg\_pool2d(h1\_spike, 2)  
 # 更新c2\_mem  
 h2 = fs\_mem\_update(self.fc2, h1\_spike, h2)  
 # self.fc2.weight.data = self.fc2.weight.data \* 0.5

**每个step都要调用一次act\_fun()函数来产生z(t)，因此在脉冲序列上面乘以0.5，使得脉冲序列由原来的0，1变成现在的0，0.5，相当于原本每个step用weight\*0.5代替d(t)，改为z(t) \* 0.5代替d(t)。**

**这样一来，反传成功，SNN网络可以得到训练。**

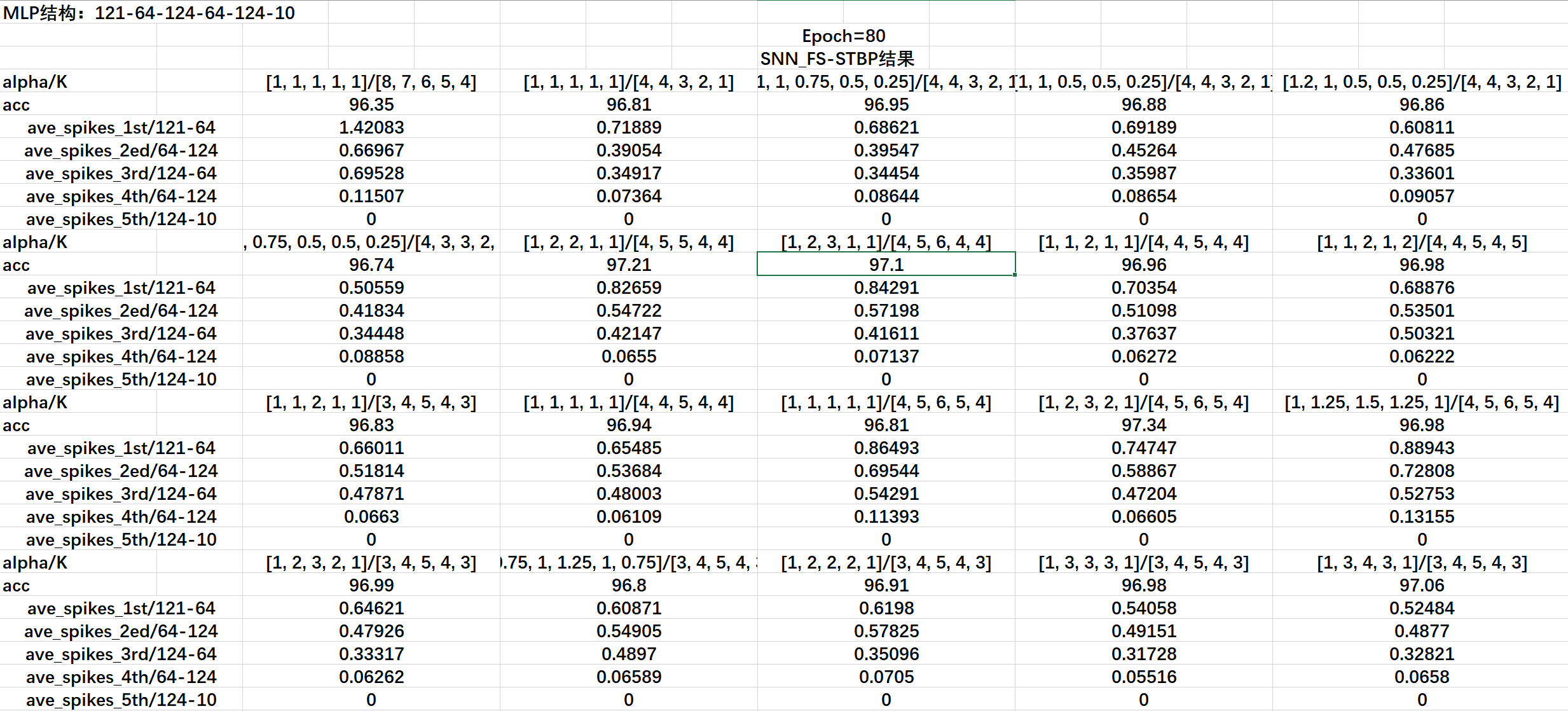
**注：本段前传代码基于convert\_wo\_train\_v3.py文件改写，个人感觉这个代码对FS神经元的前传模型表达的不是很透彻，因为传递的是膜电压而不是f(x)。以后可以参考唐林学长的Fs神经元代码进行改进。**

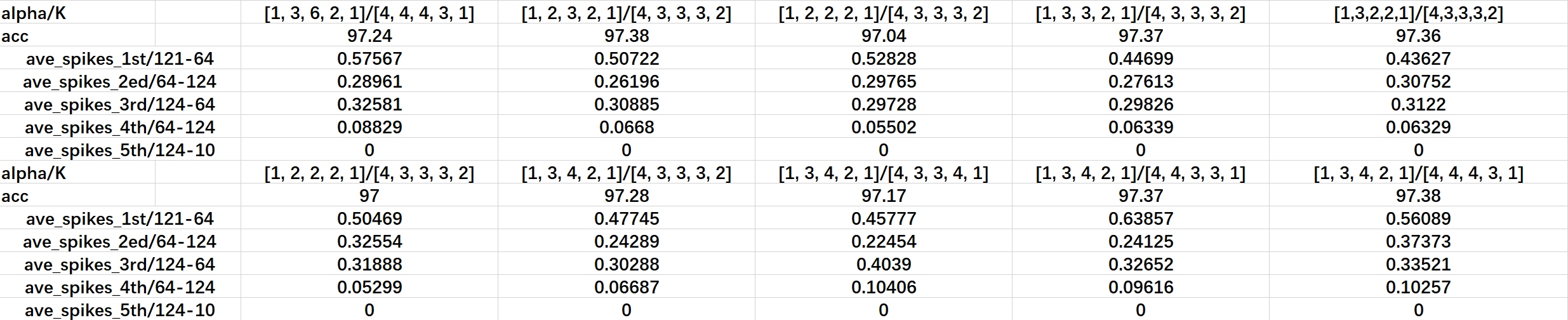
**（3）训练方式的改进情况**

**与常用的SNN训练方法STDP、ANN转化法不同的是，FS-STBP融合结构采用的是STBP的反向传播方法，使用脉冲发射函数导数的近似处理函数进行梯度近似，解决了脉冲发射函数发射处不可导的问题。**

class ActFun(torch.autograd.Function):  
 *'''  
 Approaximation function of spike firing rate function  
 '''* @staticmethod  
 def forward(ctx, input):  
 ctx.save\_for\_backward(input) # ctx：类似于self，这里保存下来input供反传用  
 return input.gt(0.).float() # input：膜电位，gt比较是否超过阈值thresh，return 脉冲0/1  
  
 @staticmethod  
 def backward(ctx, grad\_output):  
 input, = ctx.saved\_tensors  
 grad\_input = grad\_output.clone()  
 temp = abs(input) < lens# 膜电压在门限一定范围内，置1，发射脉冲，temp置1，该时间步的反传梯度为1  
 return grad\_input \* temp.float()  
  
  
act\_fun = ActFun.apply

**可以在不使用ANN进行预训练的前提下，实现 SNN网络的直接监督学习。同时，与大多数采用LIF-BP算法的SNN相比，FS-STBP融合结构采用的FS神经元具有更加优良的脉冲稀疏性和准确性，(此处说法不是太严谨，需要进一步搭建相同网络结构的LIF-STBP，用控制变量法进行验证)有望进一步改进SNN芯片性能。**

**二、测试结果**

**每一层的K值和alpha值设置测试数据如上图。**

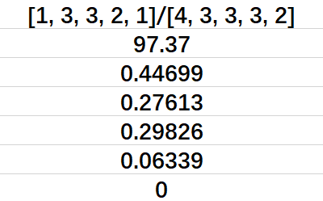
**经过测试可以看到：**

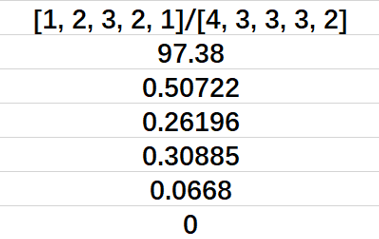
**第一层K值设为4，后层K值非严格单调递减； （例如4，3，3，3，1）**

**第一层alpha值设为1，alpha按照“小大小”排列； （例如1，3，3，2，1）**

**这样的训练效果较好，acc可以达到97.3%左右，而且每层每个神经元的平均脉冲数降至0.5以下，呈现出由前到后平均脉冲数越来越小的趋势。**

**两个比较好的结果的参数配置如下：**

**alpha/K=[1,3,3,2,1]/[4,3,3,3,2]时准确率为97.37%，alpha/K=[1,2,3,2,1]/[4,3,3,3,2]时准确率为97.38%。**

****