SNN 仿真实验报告

——李昭阳 2021013445

基于 Pytorch 实现一个纯净版 LIF-SNN

在 SNN 的仿真实验中，我整理获得了一个纯净版 LIF-SNN 模型，并通过联合优化超参数来获得最佳性能。以下是具体步骤与优化分析。

代码展示

补全缺失代码块如下

```
def forward(self,input_data):
    ############## your code ###########
    x, _ = self.lifcnn1(input_data, spikeAct = LIFactFun.apply)
    x, _ = self.lifcnn2(x, spikeAct = LIFactFun.apply)
    x, _ = self.lifcnn3(x, spikeAct = LIFactFun.apply)
    x = temporalPooling(x,kernel_size=x.shape[-2:])

    x = x.view(x.shape[0],x.shape[1],x.shape[2]).permute(0,2,1)
    x, _ = self.liffc1(x,spikeAct = LIFactFun.apply)
    x, _ = self.liffc2(x,spikeAct = LIFactFun.apply)
    x = torch.mean(x,dim=1)
    ##############    end    ###########
    return x
```

设计卷积层输出通道和全连接层输入维度一样大是为了参数连接的一致、平滑过渡。

参数优化过程

在优化的过程中，我选取了三个超参数进行分析，分别为批量大小、学习率、迭代周期数。同时用控制变量法对其优化效果进行了量化实验。

首先分析批量大小（batch_size）对训练的影响，多组超参数实验结果如下：（下文中的所有图注格式均为：卷积层大小-全连接层大小-批量大小-学习率）

```
training loss: 0.6795827514174193 % train_acc: 97.82432432432432 %
100%|██████████| 3750/3750 [04:24<00:00, 14.16it/s]
100%|██████████| 625/625 [00:22<00:00, 27.22it/s]
Saving...
epoch=2, train_loss=0.0068, train_acc=0.9783, test_loss=0.0032, test_acc=0.9822, max_test_acc=0.9822, total_time=287.737096786499
```

128-64-16-0.01

```
training loss: 0.5554079625289887 % train_acc: 98.20659722222223 %
100%|██████████| 937/937 [02:01<00:00,  7.71it/s]
100%|██████████| 157/157 [00:14<00:00, 10.55it/s]
Saving...
epoch=2, train_loss=0.0056, train_acc=0.9821, test_loss=0.0030, test_acc=0.9841, max_test_acc=0.9841, total_time=136.38543105125427
```

128-64-64-0.01

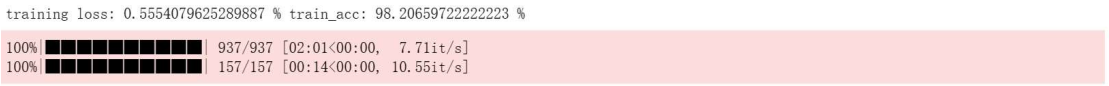
```
training loss: 0.5027948967763223 % train_acc: 98.490234375 %
100%|██████████| 468/468 [01:40<00:00,  4.63it/s]
100%|██████████| 79/79 [00:13<00:00,  5.83it/s]
Saving...
epoch=2, train_loss=0.0050, train_acc=0.9850, test_loss=0.0029, test_acc=0.9843, max_test_acc=0.9843, total_time=114.56573677062988
```

128-64-128-0.01

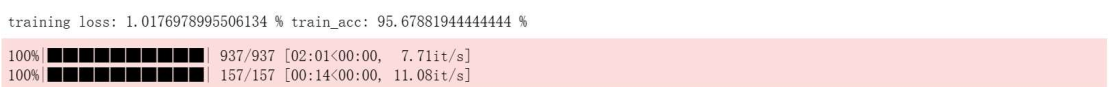
由图可知，过小的 batch_size 会使得模型预测准确率降低；同时由统计可知，训练时间随着 batch_size 的增加而减小。

再分析学习率对训练的影响，实验结果如下：

128-64-64-0.001



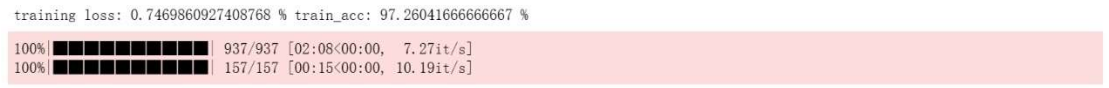
128-64-64-0.01



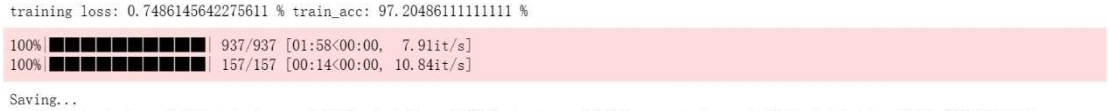
128-64-64-0.1

适当的增加学习率可以略微提升训练准确率，同时使得收敛速度更快。但是过大的学习率会引起振荡，使得训练精度不足。同时不精确的学习率会增大 CPU 占用时间，使得对设备的要求提高。

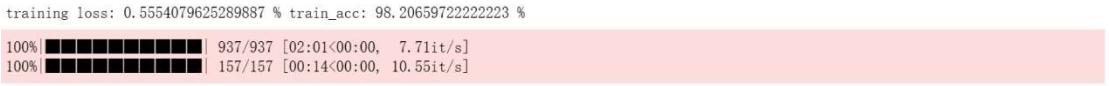
分析卷积层神经元数量对训练的影响，实验结果如下：



32-64-64-0.01



64-64-64-0.01



128-64-64-0.01

共三个卷积层，设计每层神经元都以 2 倍迭代。可以看到，随着卷积层神经元数目增加，训练准确率也在提高。究其原因是卷积层神经元数量增多带来更复杂的特征提取导致的。

分析全连接层神经元数量对训练的影响，实验结果如下：

```
training loss: 0.5398715039611691 % train_acc: 98.3125 %
100%|██████████| 937/937 [02:02<00:00,  7.64it/s]
100%|██████████| 157/157 [00:15<00:00, 10.38it/s]
Saving...
epoch=2, train_loss=0.0054, train_acc=0.9831, test_loss=0.0028, test_acc=0.9846, max_test_acc=0.9846, total_time=137.81674933433533
```

128-32-64-0.01

```
training loss: 0.5554079625289887 % train_acc: 98.20659722222223 %
100%|██████████| 937/937 [02:01<00:00,  7.71it/s]
100%|██████████| 157/157 [00:14<00:00, 10.55it/s]
Saving...
epoch=2, train_loss=0.0056, train_acc=0.9821, test_loss=0.0030, test_acc=0.9841, max_test_acc=0.9841, total_time=136.38543105125427
```

128-64-64-0.01

```
training loss: 0.46270310058025643 % train_acc: 98.712890625 %
100%|██████████| 468/468 [02:01<00:00,  3.84it/s]
100%|██████████| 79/79 [00:15<00:00,  5.07it/s]
Saving...
epoch=2, train_loss=0.0046, train_acc=0.9872, test_loss=0.0025, test_acc=0.9875, max_test_acc=0.9875, total_time=137.57465195655823
```

128-256-64-0.01

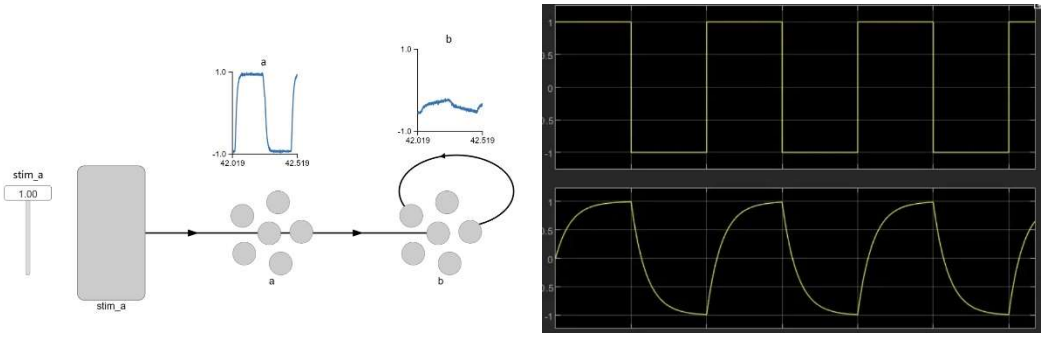
共两个全连接层。可以看到，随着隐藏层神经元数目增加，训练准确率也在提高。全连接层神经元数量增多可以带来更细节的内部结构。

综上分析了各类超参数对模型的影响，但是由于初始参数十分鲁棒，故可认为不需要额外调整即可完成实验任务，且效果较好。

自由探索

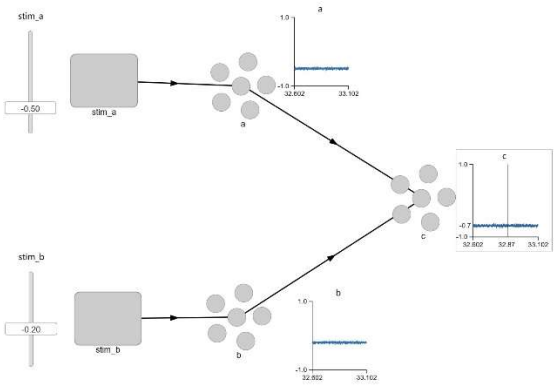
我选取了 Nengo 的 SNN 框架，复现了 memory、adding、calculate、multiply 四种内置 example。

Memory 模板可以实现神经元的记忆功能，可以使得输入对输出的影响得以减缓。其效果图如下。



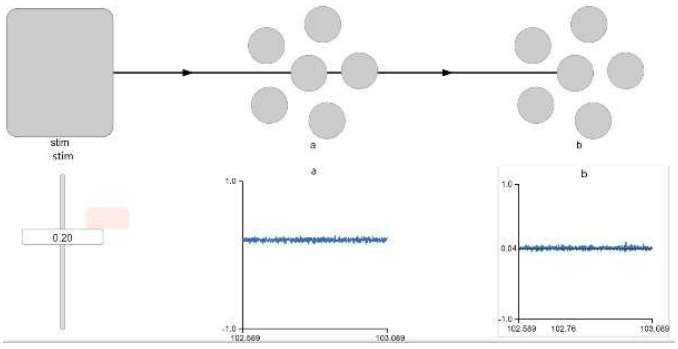
观察方波输入时，其输出波形图像和电容电路的输出图像相似。事实上也是如此，电容储存的电荷可以减缓输入的变化；电容也有记忆作用。这个模块用于模仿电路或者人体均有很好的意义。

Adding 模块可以实现神经元输出相加的功能。如图，stim_a + stim_b = 0.7。

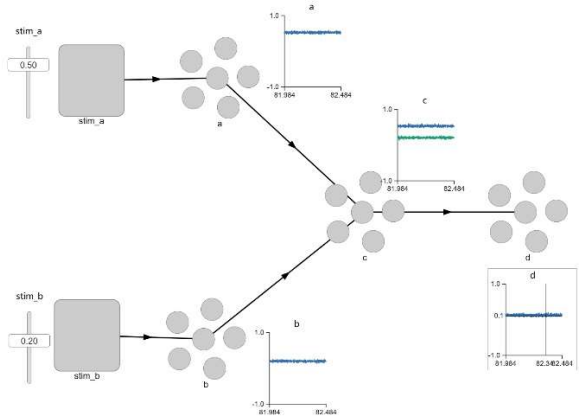


此处的相加是实现加权输入的最简单形式。

Calculate 模块可以实现模块间更复杂的运算，这部分操作很简单，支持直接输入传递方程的运算，大大降低了编程难度。如下图，$stim^2 = 0.04$



Multiply 模块可以实现神经元输出值间相乘的关系，使用了之前 Calculate 方法，传递方程为前两个神经元输出值之积。如图，stim_a × stim_b = 0.1。



同时我还探索了不同脉冲编码的效果和优劣。我采用了自创的“反 onehot 编码”、二进制编码、格雷码描述不同脉冲，并在课上所给的《纯净版 LIF-SNN》项目中，与“onehot 编码”的效果进行对比。

“onehot 编码”的训练效果如下。可知在 3 个 epoch 下，测例准确率达到 98.41%，效果较好。

```
training loss: 0.5554079625289887 % train_acc: 98.20659722222223 %
100%|██████████| 937/937 [02:01<00:00,  7.71it/s]
100%|██████████| 157/157 [00:14<00:00, 10.55it/s]
Saving...
epoch=2, train_loss=0.0056, train_acc=0.9821, test_loss=0.0030, test_acc=0.9841, max_test_acc=0.9841, total_time=136.38543105125427
```

“反 onehot 编码”是对所有“onehot 编码”取反得到的，即若“onehot 编码”是[0, 0, 0, 0, 1, 0]，则“反 onehot 编码”是[1, 1, 1, 1, 0, 1]。理论推导，此种编码形式应当取得和“onehot 编码”类似的效果，但是在现实脑科学中，过多的高电压会增加耗能，生物体不应进化出此类编码方式，同时长时间的高电压可能会对神经元造成损害，所以在此处仅做一次仿真尝试。

```
training loss: 2.4752670499599643 % train_acc: 98.32465277777777 %
100%|██████████| 937/937 [01:56<00:00,  8.05it/s]
100%|██████████| 157/157 [00:14<00:00, 10.97it/s]
Saving...
epoch=2, train_loss=0.0247, train_acc=0.9833, test_loss=0.0048, test_acc=0.9835, max_test_acc=0.9835, total_time=130.789537191391
```

可以看到“反 onehot 编码”可以略微提高准确率，同时缩减训练时间。

二进制编码是常用的编码方式，由于充分利用了编码位数，可以缩减每个数字的编码长度，节约储存空间与计算时间。但是也存在许多高电压，增加了耗能问题。以下是二进制编码的训练效果。

```
training loss: 0.6821839922987338 % train_acc: 97.28298611111111 %
100%|██████████| 937/937 [03:01<00:00,  5.17it/s]
100%|██████████| 157/157 [00:25<00:00,  6.23it/s]
Saving...
epoch=2, train_loss=0.0068, train_acc=0.9730, test_loss=0.0042, test_acc=0.9790, max_test_acc=0.9790, total_time=206.46420693397522
```

由于二进制编码的编码与解码工作需要时间，故训练时间有所延长，但是训练准确率依旧较高，是一个较好的编码方式。

理论上，格雷码相邻的数值仅有一个位元的差异。相比于二进制码，当信号从一个数值变化到另一个数值时，格雷码只有一个位元会发生改变，降低了出错

的可能性。同时由于信号的快速变化可能引起震荡问题。格雷码的特性可以减少这种震荡，有助于提高信号的稳定性。

以下是格雷码的训练情况。

```
training loss: 0.689588493357102 % train_acc: 81.92534722222223 %
100%|██████████| 937/937 [03:28<00:00,  4.49it/s]
100%|██████████| 157/157 [00:32<00:00,  4.87it/s]
Saving...
epoch=2, train_loss=0.0069, train_acc=0.8197, test_loss=0.0032, test_acc=0.8790, max_test_acc=0.8790, total_time=240.76259565353394
```

由于代码设计问题，格雷码的编码与解码工作需要更多时间，故训练时间更长。同时采用格雷码，训练效果远远不如其他方法，最初认为是由于模型初始化的问题，此后将训练 epoch 数量增加至 8、15 回，如下所示。

```
training loss: 0.49411279467555386 % train_acc: 86.58159722222221 %
100%|██████████| 937/937 [03:28<00:00,  4.48it/s]
100%|██████████| 157/157 [00:32<00:00,  4.84it/s]
Saving...
epoch=7, train_loss=0.0049, train_acc=0.8661, test_loss=0.0024, test_acc=0.9058, max_test_acc=0.9058, total_time=241.3685212135315
training loss: 0.4234808493509061 % train_acc: 89.93402777777779 %
100%|██████████| 937/937 [03:28<00:00,  4.50it/s]
100%|██████████| 157/157 [00:29<00:00,  5.34it/s]
epoch=14, train_loss=0.0042, train_acc=0.8994, test_loss=0.0023, test_acc=0.9216, max_test_acc=0.9271, total_time=237.7848994731903
```

可以看到，随着 epoch 数量增加，训练准确率一直在增加，但是始终无法达到其他编码方式的训练效果。

SNN 的设计灵感来自生物神经系统，而生物神经系统中的神经元之间的通信方式并不是使用格雷码。在追求生物可解释性的角度考虑，使用与生物系统更为相似的编码方式可能更为合适。

同为非生物可解释性编码，二进制编码的方法准确率较高，但是与二进制编码相比，格雷码可能没有更高的编码效率，因为它主要是为了降低误码而设计的，而不是为了有效地表示脉冲的信息。

总的来说，尽管格雷码在传统数字系统中有其优势，但在与生物神经系统相似的脉冲神经网络中，更适合使用其他与时序相关的编码方式，以更好地模拟生物神经系统的工作原理。

综上所述，虽然“反 onehot 编码”在本次仿真表现上略微胜过“onehot 编码”，但是不源自于生物可解释性，对 SNN 略有背离，所以认为“onehot 编码”依旧是当前测试下最好的编码方式。

总结

这次实验为我积累了脉冲神经网络领域的宝贵经验，我有很多心得体会。

首先我认为基于仿生角度的思考是很宝贵的，SNN 源自于对人类大脑真实的思考，并且它的创始人有信心和能力用数学化的方式将其建模成一个可优化的神经网络问题。这对我以后的学习生活有很大的指导，拓宽了我思考问题解决方案的视野。

其次，我了解了 nengo 编程框架，让我对“简单编程”的认识更上一层楼，在之后我也希望有机会尝试开发一个带有可视化的编程平台，直观地表述我的想法。

同时，在探究脉冲编码对 SNN 的效果时，虽然多次碰壁，并且没有得到一个比“onehot 编码”更好的编码方案，但是我在尝试过程中，对二进制编码、格雷码的局限性理解更为透彻，也在不断探索中收获了很多快乐和满足。

总的来说，学习脉冲神经网络是一次充满挑战和新奇体验的过程，我收获颇

丰。通过不断实践，我真正不断提高了自己的学习技能，应对着不断变化的挑战。谢谢老师和助教的耐心讲解与准备！