

1 SNN 模型描述

本次实验 SNN 网络的实现基于 `snnTorch`.

1.1 神经元

所使用的脉冲神经元是 `snnTorch` 中的 LIF 神经元, 即 Leaky Integrate-and-Fire 神经元。

Leaky 用于模拟离子渗透的过程, 由于细胞膜内外的电位是靠不同的离子浓度维持的, 细胞膜不断进行膜内外离子的交换, 神经元受到的刺激会改变细胞内外离子的分布, 而当只有一次输入时, leaky 就能够模拟电荷泄漏, 神经元回落到静息状态的过程。Integrate 的概念和实现则类似于传统的 ANN 结构, 即每个脉冲神经元都会接受到直接与其相连的其他神经元的脉冲, 这也是较为简化而便于实现大规模集成运算的一种方式, 同时也贴合人脑的神经元连接方式。Fire 则模拟了神经元的脉冲输出, 即当神经元的电位达到阈值时, 神经元会发出脉冲, 并且电位会回落到静息电位。

1.2 拓扑结构

为了便于比较 CNN 和 SNN 的性能, 本次实验所实现的两种网络有着一样的宏观拓扑结构, 而差别仅仅在于微观上神经元的差异和神经元之间的连接方式、激活方式等。SNN 和 CNN 均内嵌了两层卷积层, 两层池化层, 一个全连接层。SNN 不同的地方在于, 在每个池化层或者全连接层后, SNN 还内嵌了一个 Leaky 层, 以实现上述对 LIF 神经元前馈计算和反向传播的模拟。

```
1  SNN(  
2  (0): Conv2d(1, 12, kernel_size=(5, 5), stride=(1, 1))  
3  (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
4  (2): Leaky()  
5  (3): Conv2d(12, 64, kernel_size=(5, 5), stride=(1, 1))  
6  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
7  (5): Leaky()  
8  (6): Linear(in_features=1024, out_features=10, bias=True)  
9  (7): Leaky()  
10 )  
11 ANN(  
12 (conv1): Conv2d(1, 12, kernel_size=(5, 5), stride=(1, 1))  
13 (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
14 (conv2): Conv2d(12, 64, kernel_size=(5, 5), stride=(1, 1))  
15 (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
16 (fc): Linear(in_features=1024, out_features=10, bias=True)  
17 )
```

可以看到, SNN 和 CNN 的各层网络参数是完全相同的, 唯一的区别是 SNN 内嵌了 Leaky 层。

1.3 输入输出

由于 SNN 和 CNN 的入口是 channel, kernel_size 完全相同的卷积层, 其输入是一致的, 两个网络可以复用一个 dataset, 即 torch 内置的 MNIST 数据集。神经网络每次接受形如 `batchsize*channel*height*width`

的输入，其中 `batchsize` 我设定为 128，而 `channel`、`height`、`width` 分别为 1、28、28，是 MNIST 数据集本身的参数。

SNN 和 CNN 在输出上略有一些差别。CNN 部分我们已经很熟悉，在上面的网络结构中也可以看到，对于 MNIST 这样的分类任务，CNN 的输出接口是 `out_features=10` 的全连接层，即一个能够表征各个方向上的概率的 10 维向量。而 SNN 的输出接口是 Leaky 层，故其输出是一个形如 `num_step*batchsize*feature` 的张量，其中 `num_step` 是模拟的时间步数，我设定为 50，`batchsize` 仍为 128，`feature` 为分类的所有可能类别数，也为 10。

1.4 学习方法

两种神经网络使用的优化器均为 torch 内置的 Adam 优化器，参数选择上，我将学习率设定为 0.01，betas 设定为 (0.9, 0.999)，其余采用默认设置。

两种网络的学习方法均为前馈计算和反向传播，其中 CNN 的方法我们已经较为熟悉了，SNN 的方法则集成在了 `snnTorch.backprop` 的 BPTT 函数中。BPTT 函数实现的是依时间反向传播的方法，具体来说，在损失累积的同时，对每个时间步应用前向传播，而仅在每个时间步序列的末尾应用反向传播和参数更新。我们需要为其传入一个损失函数，我按照教程的建议采取了 `snnTorch` 库中内置的 `ce_rate_loss` 函数，即 `CrossEntropySpikeRateLoss`。

此外，由于 MNIST 数据集本身是时间无关数据集，在调用 BPTT 函数时，需要将 `time_var` 参数设定为 `False`，以避免在每个时间步都进行数据集的采样。

2 实验结果与对比

2.1 效率对比

2.2 精度对比