人工神经网络 transformer 转换为脉冲神经网络 spikformer

by hzh

注：该文档主要讲述了模型在代码上是如何实现的，文档里暂时还没有完整的加入各个方法的原理，主要思想来自三篇论文：

Quantization Framework for Fast Spiking Neural Networks

SPIKFORMER WHEN SPIKING NEURAL NETWORK MEETS TRANSFORMER

Optimal ANN-SNN Conversion for High-accuracy and Ultra-low-latency Spiking Neural Networks

对应的代码放在：https://github.com/h-z-h-cell/ann2snn_transformer

一、模型概览

共有如下几个文件：

- cifar10.yml : 存放 cifar10 训练的参数
- train.py : 负责读入参数，建立模型，进行模型训练，或者测试现有模型
- model.py : 存放我们设计的 transfomer 模型
- Myneuron.py : 存放我们设计的几种神经元，包括最大池化层神经元，正负脉冲神经元
- utils.py : 存放我们设计的工具函数，包括脉冲网络初始化函数，模型模块替换函数等
- augment.py : 存放几种数据增强方法，该模块我是从 QCFS 算法的开源代码处拷贝的

-cifar10.yml 有如下参数：

action: train/test #表示进行训练还是进行测试

mode: snn/ ann #ann 表示对现有模型进行 ann 测试，snn 表示对现有 ann 模型进行转化然后再进行测试，仅当 action 为 test 的时候该参数才起作用

l: 4 #一个正整数，表示 ann 非线性层使用的函数的量化步长

t: 8 #一个正整数，表示转换成 SNN 后要模拟的时间步长

extra_t: 0 #要额外模拟的 SNN 时间步长,这段时间内 SNN 进行全 0 的输入，这是为了补齐神经元在模拟 t 时长之后多发出或者少发出的脉冲（但是实际测试似乎取 t=x,extra_t=y 的效果不如取 t=x+y,extra_t=0）

bs: 64 #训练采用的 batchsize

lr: 1e-2 #训练采用的初始学习率

wd: 1e-4 #训练采用的权重衰减

epochs: 150 #训练的轮数

id: mymodel2 #模型的名字，代码在当前位置建立一个名为 id 的文件夹，里面包括一个记录训练每轮的损失值和准确率的文件以及各个模型保存结果。

seed: 42 #设置的随机数种子，为了可重复实验

data: cifar10 #使用的数据集，目前我只测过 cifar10

dim: 384 #将每个 patch 转换出的向量的嵌入维度

num_heads: 12 #采用多头自注意力的头数

patch_size: 4 #每个 patch 的大小，embeding 时图像视作分割为 patch*patch 的大小看

mlp_ratio: 4 #每个 block 中 mlp 的隐藏维度和输入维度的比例

save: True #是否保存训练模型

save_acc: True #是否保存训练的损失值和准确率的日志

model_path: ./mymodel1/best_model.pt #测试或者继续训练时使用的模型的路径

inherit: False #训练时是否在已有的模型上训练

use_maxpool: True #是否使用最大池化，这里推荐使用最大池化而不是平均池化

use_amp: True #是否使用 amp 加速

二、人工神经网络 transformer 模型的训练

train.py 包括如下几个部分

①if __name__ == '__main__': 主函数部分，负责调用_parse_args 解析输入的参数、获取数据集加入数据增强方法并转换为 Dataloader、根据参数将初始定义的模型转换为我们想要训练的模型结构、根据 args.mode 决定选择 train 函数进行训练还是 eval 函数进行测试。

②def _parse_args(): 解析读入 cifar10.yml 配置文件中的参数

③def train(model,train_loader,eval_loader,args):

训练模型的函数，设定 optimizer,scheuler 等参数，进行多轮训练，每轮具体调用 train_one_epoch 函数在训练集上进行训练，接着调用 validate 在测试集上验证损失和准确率，之后根据需要保存相关参数

④def train_one_epoch(model, loader, optimizer, loss_fn, args,nowepoch):

计算该轮的损失并进行反向传播迭代更新，返回平均损失以及每个 batch 的平均用时

⑤def eval(model,model_path,eval_loader,args):

测试模型的函数首先载入模型

若 args.mode 为 ann 则调用 validate 函数测试该模型的 ann 准确率和损失

若 args.mode 为 snn 首先将所有最大池化层替换为最大池化神经元、将所有非线性层替换为对应的可以发放正负脉冲的神经元，接着调用 validate_snn 测试转换后的 SNN 的准确度。

⑥def validate(model, loader, loss_fn, args):

计算该模型在测试集上的损失和准确率，返回平均损失，每个 batch 的平均用时，top1 准确率以及 top5 准确率(前五个里面有一个对就算对)。

⑦def validate_snn(model,loader,loss_fn,args):

计算该模型在测试集上的损失和准确率，返回平均损失，每个 batch 的平均用时，top1 准确率以及 top5 准确率。Snn 的执行方法为，保持输入不变，对转换后的模型重复 args.t 次执行，将输出 output 累加计算输出准确率。若存在 args.extra_t 则修改输入为 0，对转换后的模型重复 args.extra_t 次执行，将输出累加 output 中（为了延长时间抵消多发或者少发的脉冲，但是实际测试似乎取 t=x,extra_t=y 的效果不如取 t=x+y,extra_t=0）。最后测试完重置模型膜电位等参数。

model.py : 存放我们设计的 transfomer 模型，包括如下模块

①class Transformer(nn.Module):

由一个 Embedding 和 depths 个 Encoder Block 组成（depths 默认为 4）

②class Embedding(nn.Module):

一开始这一块我想简单了直接就只定义了一个卷积+BN+激活函数层，导致在 CIFAR10 上用优化器 SGD 跑了 150 轮只有 84%，500 轮也只有 89%准确率。后来将优化器改为 AdamW（这么改是因为 transformer 收敛比较慢，为了加快收敛），并借鉴了 spikefomer 论文中的做法，把 D 个特征维度通过四次卷积中间按 D//8,D//4,D//2,D 的顺序得到（主要为了增加网络深度减少信息的丢失），在 CIFAR10 上跑了 200/300 轮便有 93.38%，最终最优解准确率为 94.06%。该模块最后还进行了位置嵌入，通过 x=x+x_feat 实现，x_feat 通过有 D 维特征的 x 卷积变换得到。

③class Block(nn.Module):

该类定义了 Encoder Block 由一个 VSA 和一个 MLP 组成

④class VSA(nn.Module):

VSA 中先将输入并行经过三个 全连接层-BN-激活函数三层 组合，分别得到 Q,K,V 三个矩阵，

然后先求 x=softmax(Q@K.T)@V 再经过一个 BN 和激活函数层。这个地方不知道是否后续需要进行修改，因为在转换为 SNN 网络后 softmax 的操作感觉不符合脉冲的硬件的规则，感觉可以就像最大池化神经元一样，考虑设计一种神经元以及对应的 ann 中的激活函数来替代 softmax 层。

⑤class MLP(nn.Module):

由两组全连接-BN-激活函数层组成，中间的隐藏维度为 input 的维度*arg.mlp_ratio(默认为 4)

三、转换为脉冲神经网络 spikformer

转换总共工作包括四个部分：

该部分主要是复现论文：Quantization Framework for Fast Spiking Neural Networks 中的思想

1. ann 在训练的时候使用 QCFS 函数替代 Relu 函数，需要设置参数 args.l 作为量化步长，详细见 utils.py 中的 replace_activation_by_floor 以及 MyFloor 函数
2. 训练完成后将模型中的所有卷积层替换为脉冲神经元层，详见 Myneuron.py 中的 ScaledNeuron，以及 utils.py 中的 replace_activation_by_neuron
3. 训练完成后将模型中的所有最大池化层替换为最大池化神经元，详见 Myneuron.py 中的 MaxpoolNeuron，以及 utils.py 中的 replace_maxpool2d_by_MaxpoolNeuron
4. 在训练时保持输入不变，进行测试，详见 train.py 中的 validate_snn，每次测试完一轮数据要重置模型详见 utils.py 中的 reset_net

Myneuron.py

①class TwotypeSpikeIFNode

该神经元继承自 spikingjelly.clock_driven.neuron.IFNode 关键步骤是实现了对 def neuronal_fire(self)函数的覆盖。这是一种可以发出正脉冲和负脉冲的神经元，该神经元默认阈值电压为 1，膜电位为 0，每次初始话后归 0(所以 ScaledNeuron 在使用前都会进行一次 0.5 的充电)。该神经元只有一个参数 Tmax，表示最大可以发出的脉冲数，即可以控制总脉冲数在[0， Tmax]之间。

②class ScaledNeuron

该神经元继承自 pytorch.nn.Module，由两个参数 T 和 scale。

由于该神经元会调用 TwotypeSpikeIFNode，参数 T 用于传递给 TwotypeSpikeIFNode 作为 Tmax。该神经元的作用是将当前膜电位做标准处理(除以 scale)后传递给双脉冲神经元得到输出后再恢复电位(乘以 scale)。scale 值通过 MyFloor 的参数 up 传递得到。

③class MaxpoolNeuron

该神经元无训练参数，仅仅是为了解决脉冲神经网络采用最大池化法出现脉冲过多的情况。（方案来自论文 Quantization Framework for Fast Spiking Neural Networks）

方案如下：其中 Z_t 是每次神经元得到的输入，z_t 是每次神经元的输出

令$M_t = max_pooling(Z_t), z_t = M_t - M_{t-1}$

那么会有：

$\sum_{i=1}^{T} z_t = \sum_{i=1}^{T}(M_t - M_{t-1}) = \ldots = M_T - M_{-1}$

只要我们令$M_{-1} = 0$

那么就有$\sum_{i=1}^{T} z_t = M_T$

于是这样就可以做到在每个时间进行最大池化的同时不损失精度。

Input: The accumulated spike counts of spiking neurons before the max pooling layer Z
Parameter: Max pooling *max_pooling*, the output of max pooling M, Current time-step t, Time window T
Output: Generated spike z

1: LET $z_{-1} = 0, M_{-1} = 0$
2: **for** $t \in [0, T]$ **do**
3: $M_t = max\_pooling(Z_t)$
4: $z_t = M_t - M_{t-1}$
5: **end for**

utils.py

def seed_all(seed=42,rank=0):#设置随机数种子使得实验可重复
class MyFloor(nn.Module):#utils 中 replace_activation_by_floor 调用，是 QCFS 算法中用于替代激活函数的函数
def isActivation(name):#根据名字判断某一个模块是不是激活函数模块
def replace_activation_by_floor(model,t=8,up=8.):#替换模块中的激活函数为 QFCS 中的离散函数，用于 ANN 的训练和测试
def replace_activation_by_neuron(model,T):#将模型中 ann 的激活函数为 snn 神经元的计算方式，用于 SNN 输出测试
def replace_maxpool2d_by_avgpool2d(model):#将模型中的平均池化层全部替换为最大池化层（一般不打算用）
def replace_maxpool2d_by_MaxpoolNeuron (model):#将模型中最大池化层替换为最大池化神经元层
def reset_net(model):#将模型中的所以神经元初始化，用于 SNN 重新初始化
class AverageMeter:#用来计算平均值的类
def accuracy(output, target, topk=(1,)):#用来计算 top-k 准确率的的函数

四、目前的训练结果

采用 AdamW 进行训练 300 轮,lr=1e-2,wd=1e-4（实际没必要这么多轮感觉到两百一二十的时候就差不多了）测试结果如下：

ANN : top1-acc:94.06 top5-acc:99.83

SNN: top1-acc:（实际执行轮数为 arg.t+arg.extra_t）

arg.t/arg.t+arg.extra_t	1	2	3	4	5	6	7	8	9	10	11	12
1	73.7	68.59	52.06	25.08	15.4							
2		85.09	86.47	83.65	67.76	36.25						
3			90.09	91.34	90.3	86.19	73.7					
4				92.32	92.31	92.35	90.48	85.48				
5					93.13	93.35	93.03	91.59	89.37			
6						93.74	93.77	93.53	92.59	90.92		
7							93.95	93.93	93.61	92.95	91.9	
8								94.13	93.88	93.79	93.44	92.53
9									94.14	93.97	93.84	93.54
10										94.17	94.04	93.89
11											94.15	94.11
12												94.07

现象 1：arg.extra_t 感觉不设置反而比设置了来得好，后续会思考这个问题

现象 2：SNN 最终准确率比 ANN 还高

现象 3：由于 ANN 的准确率为 94.06 感觉在这个模型在 arg.t+arg.extra_t 为 5-8 的时候就已经可以在 SNN 上非常充分地还原模型了。