cifar10_Target_Propagation

兰欣泽 2018012036

一、整体结构

根据已有代码，正向传播采用4个全连接层，且在最后加入LogSoftmax，反向计算target采用3个全连接层：

```
DTPNet(
  (logsoft): LogSoftmax(dim=1)
  (wf): ModuleList(
    (0): Linear(in_features=3072, out_features=1000, bias=True)
    (1): Linear(in_features=1000, out_features=1000, bias=True)
    (2): Linear(in_features=1000, out_features=1000, bias=True)
    (3): Linear(in_features=1000, out_features=10, bias=True)
  )
  (wb): ModuleList(
    (0): Linear(in_features=10, out_features=1000, bias=True)
    (1): Linear(in_features=1000, out_features=1000, bias=True)
    (2): Linear(in_features=1000, out_features=1000, bias=True)
  )
)
```

compute_target部分根据根据论文复现：

$$\hat{\mathbf{h}}_{M-1} \leftarrow \mathbf{h}_{M-1} - \hat{\eta}\frac{\partial L}{\partial \mathbf{h}_{M-1}}$$

其中η选择了0.5。

为方便计算与理解，将backward放在主函数中。

为方便计算，为reconstruct添加了一个参数i，方便分别计算每层的reconstruction 。

对于损失函数，最后一层选择nlll，之前的层选择mse。

对每层的梯度计算，采用了autograd.grad()和optimizer同时使用的方法，原因会在之后讨论。

整个训练过程依据论文的流程：

Algorithm 1. Training deep neural networks via difference target propagation

Compute unit values for all layers:
for $i = 1$ to M **do**
  $\mathbf{h}_i \leftarrow f_i(\mathbf{h}_{i-1})$
end for
Making the first target: $\hat{\mathbf{h}}_{M-1} \leftarrow \mathbf{h}_{M-1} - \hat{\eta}\frac{\partial L}{\partial \mathbf{h}_{M-1}}$, ($L$ is the global loss)
Compute targets for lower layers:
for $i = M-1$ to 2 **do**
  $\hat{\mathbf{h}}_{i-1} \leftarrow \mathbf{h}_{i-1} - g_i(\mathbf{h}_i) + g_i(\hat{\mathbf{h}}_i)$
end for
Training feedback (inverse) mapping:
for $i = M-1$ to 2 **do**
  Update parameters for g_i using SGD with following a layer-local loss L_i^{inv}
  $L_i^{inv} = ||g_i(f_i(\mathbf{h}_{i-1} + \epsilon)) - (\mathbf{h}_{i-1} + \epsilon)||_2^2, \quad \epsilon \sim N(0, \sigma)$
end for
Training feedforward mapping:
for $i = 1$ to M **do**
  Update parameters for f_i using SGD with following a layer-local loss L_i
  $L_i = ||f_i(\mathbf{h}_{i-1}) - \hat{\mathbf{h}}_i||_2^2$ if $i < M$, $L_i = L$ (the global loss) if $i = M$.
end for

二、遇到的困难与Debug过程

最开始感觉任务较为简单，根据论文中的流程写了一个网络，同时将各种循环写的很好看，也能适合各种参数输入的网络，之后就出现了如下报错：

```
RuntimeError: one of the variables needed for gradient computation has been modified by an
inplace operation.
```

发现是inplace的问题。

之后找遍整个网络，发现没有明显的inplace操作，考虑是否可能是在构建一些层时采用循环时出现问题，于是将所有循环展开（也导致了现在的代码不美观的问题），发现还是一样的报错，偶尔网络能够运行，global_loss和acc却一直不变，说明网络根本没有成功更新梯度。

经过查阅资料，发现可能是网络结构导致在计算单层梯度时，由于backward会自动计算之前的梯度，导致前面梯度被更新，所以再一次计算的时候就会有inpalce问题，且这样计算的梯度并不正确。

在弄清楚backward()和optimizer的实现机制后，发现backward并不能使用，因为每次会计算之前层的所有梯度，如果retain_grad=False会报错，如果retain_grad=True还会对梯度进行累加求和。optimizer是没有问题的。

考虑用.detach()阻断网络计算单层梯度时backward反向传播的问题，但尝试之后未能成功实现。

之后决定使用autograd.grad()和optimizer结合：

```
pp0=torch.autograd.grad(loss_wb0,model.wb[0].weight)[0].detach()
model.wb[0].weight.grad=pp0
optimizer_wb0.step()
```

每次计算出单层梯度，用detach()阻断，再将weight.grad手动等于算得梯度，最后使用optimizer更新。

最后成功使得网络收敛。

三、结果

在训练了200多个epoch后，对训练集的准确率已经达到了99%

```
Average Training Error Rate: 1.01% (504.0/50000)

[257,  1000] loss: 0.111
[257,  2000] loss: 0.108

Average Training Error Rate: 1.00% (499.0/50000)

[258,  1000] loss: 0.109
[258,  2000] loss: 0.111

Average Training Error Rate: 1.01% (507.0/50000)

[259,  1000] loss: 0.106
[259,  2000] loss: 0.108

Average Training Error Rate: 0.95% (477.0/50000)

[260,  1000] loss: 0.106
[260,  2000] loss: 0.107

Average Training Error Rate: 0.99% (493.0/50000)

[261,  1000] loss: 0.103
[261,  2000] loss: 0.109
```

但之后对测试集计算发现准确率只有48%！

每个类别的准确率如下：

```
Accuracy of plane : 47 %
Accuracy of   car : 57 %
Accuracy of  bird : 31 %
Accuracy of   cat : 28 %
Accuracy of  deer : 46 %
Accuracy of   dog : 45 %
Accuracy of  frog : 54 %
Accuracy of horse : 53 %
Accuracy of  ship : 65 %
Accuracy of truck : 58 %
```

四、总结

首次尝试复现论文中的神经网络，其实整体结构还是较简单的，但由于对pytorch中一些函数的机理不够十分明确，导致Debug用了较长时间。

对于最终结果，可以发现训练集的准确率可以达到很高，但是测试集的准确率始终在50%左右，每个类别中，动物类别的表现也较差。可以看出，改网络的设计结构导致过拟合现象明显，这是可以预料的，因为对于每层分别进行梯度计算导致特异性过大。

DTP网络宣称是对BP与SGD的一次改进，使其更接近人脑的实际结构，但是实际调试过程中发现，首先其计算速度较慢，在训练了数百个epoch后，结果也不如普通的CNN好，且过拟合现象应该较为严重。当然，网络本身结构导致了出现这个问题是必然的。

同时，令我较为疑惑的一点是既然文章认为反向传播在实际神经中存在性存疑，但其计算target时采用的也是反向计算，可能有些自相矛盾。查阅资料后发现，可能实际脑神经中反向传播可以通过神经元反向循环连接实现，传递一定的权重是可能存在的，作者应该赞同这点，但认为反向传播无法像普通BP中那样携带精确的参数，所以该网络还是有一定实际意义的，提醒人们需要更注重神经网络的“局部”。或者在网络某层不可导时，DTP也可以发挥一定作用。

总的来说，本次构建网络与调试的过程加深了我对pytorch和神经网络的了解，收获较大。