Machine perception project

Chengsi Yang

11.5.2020

Project Overview

The goal of this project is to train a cifar-10 classifier with Pytorch tools. And I used VGG 11 and VGG 16 VGG 19 to be the target model. The configuration of the project is based on anaconda and ran on Titan-X GPU on linux system. The final result showed that we can increase the accuracy to approximately 92 percent and test loss to be around 0.2. The experiment was successful.

Configuration

The model I built has structure below, where each 'M' represent the max pooling and number deonte

```
'VGG11': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
```

Convolution layer. And in each convolution layer we have one batch norm and one Relu activation function. As we know that the parameters of the VGG 11 is calculated as:

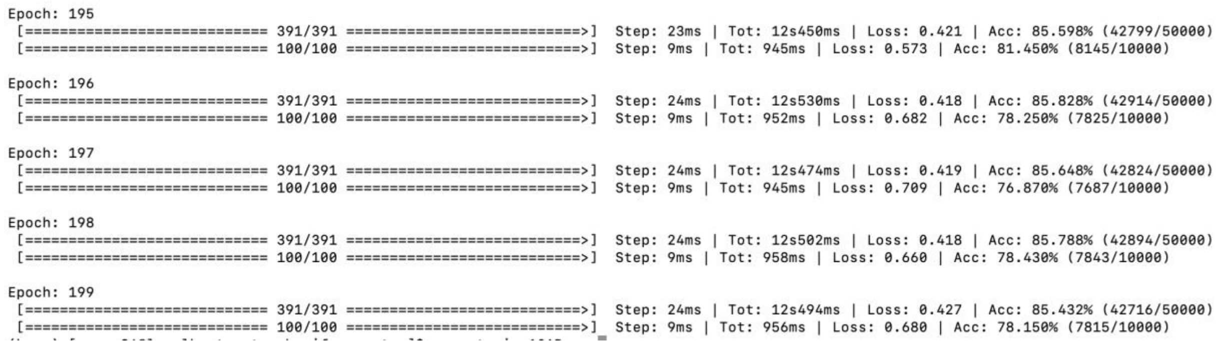
conv3-64 x 2	38,720
conv3-128 x 2	221,440
conv3-256 x 3	1,475,328
conv3-512 x 3	5,899,776
conv3-512 x 3	7,079,424
fc1	102,764,544
fc2	16,781,312
fc3	4,097,000

Which sums together will be 138,357,544 parameters. In particular for the fully-connected layers the bias is needed to be considered.

For the training process we set the learning rate to be 0.1 and total epochs to be 200. And the python file for the training is written as following logic: First, import Cifar 10 data with each batch size to be 128 and shuffled. And for the testing set we set it to be 120 batch size. Secondly, we import the model into the device and data in to the device and at the same time activate the optimizer which helps us to do the weight modification after the loss backpropagation. Third, we will train the model by each batch. We put data and label into the devices(cuda) and then apply prediction to get the loss. After that the loss will be used to do backpropagation to estimate the gradient. Finally, the gradient will be used to update the optimizer. After all of each training we will do test to see the result printing loss and accuracy to help monitor the process.

Result and Observation

As we monitored the training process, I found that during the first 30 epochs of the 200 epochs the loss is going down very quickly and accuracy of the result is rising up quickly however the accuracy does not



```
Epoch: 195
 [============================ 391/391 ===========================>]  Step: 23ms | Tot: 12s450ms | Loss: 0.421 | Acc: 85.598% (42799/50000)
 [============================ 100/100 ===========================>]  Step: 9ms | Tot: 945ms | Loss: 0.573 | Acc: 81.450% (8145/10000)

Epoch: 196
 [============================ 391/391 ===========================>]  Step: 24ms | Tot: 12s530ms | Loss: 0.418 | Acc: 85.828% (42914/50000)
 [============================ 100/100 ===========================>]  Step: 9ms | Tot: 952ms | Loss: 0.682 | Acc: 78.250% (7825/10000)

Epoch: 197
 [============================ 391/391 ===========================>]  Step: 24ms | Tot: 12s474ms | Loss: 0.419 | Acc: 85.648% (42824/50000)
 [============================ 100/100 ===========================>]  Step: 9ms | Tot: 945ms | Loss: 0.709 | Acc: 76.870% (7687/10000)

Epoch: 198
 [============================ 391/391 ===========================>]  Step: 24ms | Tot: 12s502ms | Loss: 0.418 | Acc: 85.788% (42894/50000)
 [============================ 100/100 ===========================>]  Step: 9ms | Tot: 958ms | Loss: 0.660 | Acc: 78.430% (7843/10000)

Epoch: 199
 [============================ 391/391 ===========================>]  Step: 24ms | Tot: 12s494ms | Loss: 0.427 | Acc: 85.432% (42716/50000)
 [============================ 100/100 ===========================>]  Step: 9ms | Tot: 956ms | Loss: 0.680 | Acc: 78.150% (7815/10000)
```

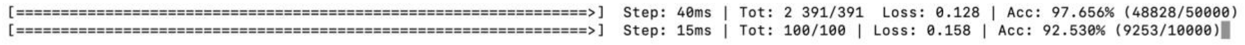
change much since then and stick to 78 - 82 percent around. And as you can see above in the final training epochs the loss is **0.680** and accuracy is **78 percent**. And this could also be the reason of overfitting because in the middle of the training process the test performance achieved **85 percent** or more.

In order to **increase** the accuracy I modified the training process by changing the net to be VGG16 and VGG19. At the same time, I modified the loss function from criterion to Cross-entropy and result seems to change dramatically.(We also choose the best accuracy rather than the latest accuracy to prevent overfitting)

```
'VGG16': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M'],
```

```
'VGG19': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512, 'M'],
```

We can see that the accuracy improves from 0 percent to 77 percent using only 10 epochs and then the network converges slowly due to the momentum process, where learning rate is automatically shrinking to get the optimize point quickly. And this slow training procedure takes another 190 epochs and this time we get the best accuracy of **92 percent** of the 10000 test sets.



```
[==============================================================>]  Step: 40ms | Tot: 2 391/391  Loss: 0.128 | Acc: 97.656% (48828/50000)
[==============================================================>]  Step: 15ms | Tot: 100/100 | Loss: 0.158 | Acc: 92.530% (9253/10000)
```

Summary

There are four experiment including VGG-11, VGG-16, VGG-16 momentum, VGG19 momentum. And the result shows that the best accuracy is **92 percent** which is VGG-16 momentum. During the training process I found overfitting happened because training loss decreases while validation loss increases. And it happened on every of the training process. There are several steps that help to reduce overfitting including using Cross-validation, train with more data, random drop off, early stopping. What I adopted is Cross-validation and early stopping. And by doing so we can get 92 percent of test result out of 80 percent or lower. What's more, if we do not use momentum, manually decreasing on learning rate is necessary because the later of the training the difficult will be for accuracy going up with high learning rate.

Reference:

Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.