Advanced Computer Vision: Assignment 5

Jean-Bernard Uwineza

May 27, 2019

1. Defining a Deep Convolutional Neural Network

First, we define a 4-layer convolutional neural network that takes as input a 3-channel RGB color image and outputs probabilities 10 classes.
Each of the convolutional layers has a non-linear ReLU activation function and Batch Normalization. The fist two convolutional layers have a Max Pooling operation.
After the fourth convolutional layer, a dropout operation is applied to prevent over-fitting.
The first fully-connected layer is passed through a ReLU activation function as well.

```
        def __init__(self, init_weights=False):
            super(ConvNet, self).__init__()
            self.conv1 = nn.Sequential(
                nn.Conv2d(3, 12, kernel_size=4, stride=1, padding=2),
                nn.BatchNorm2d(12),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=4, stride=1)
            )
            self.conv2 = nn.Sequential(
                nn.Conv2d(12, 48, kernel_size=4, stride=1, padding=2),
                nn.BatchNorm2d(48),
                nn.ReLU(),
                nn.MaxPool2d(2, 1)
            )
            self.conv3 = nn.Sequential(
                nn.Conv2d(48, 48, kernel_size=4, stride=2, padding=1),
                nn.BatchNorm2d(48),
                nn.ReLU()
            )
            self.conv4 = nn.Sequential(
                nn.Conv2d(48, 12, kernel_size=1, stride=1, padding=1),
                nn.BatchNorm2d(12),
                nn.ReLU()
            )
            self.drop_out = nn.Dropout()
            self.fc1 = nn.Linear(17 * 17 * 12, 100)
            self.fc2 = nn.Linear(100, 10)
```

Listing 1: Deep CNN model definition

The forward propagation of the network is defined as follows:

```
    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.conv3(out)
        out = self.conv4(out)
        out = out.reshape(out.size(0), -1)

        out = self.drop_out(out)
        out = F.relu(self.fc1(out))
        out = self.fc2(out)
        return out
```

Listing 2: Forward Propagation

2. Training and Testing the Deep CNN

The defined model was trained and tested on the CIFAR-10 dataset. The batch size throughout was 128, and the training rate was 0.001. Before training, the images were passed through a transform for routine pre-processing. The dataset was deliberately not augmented for reasons mentioned later. The transforms were defined as:

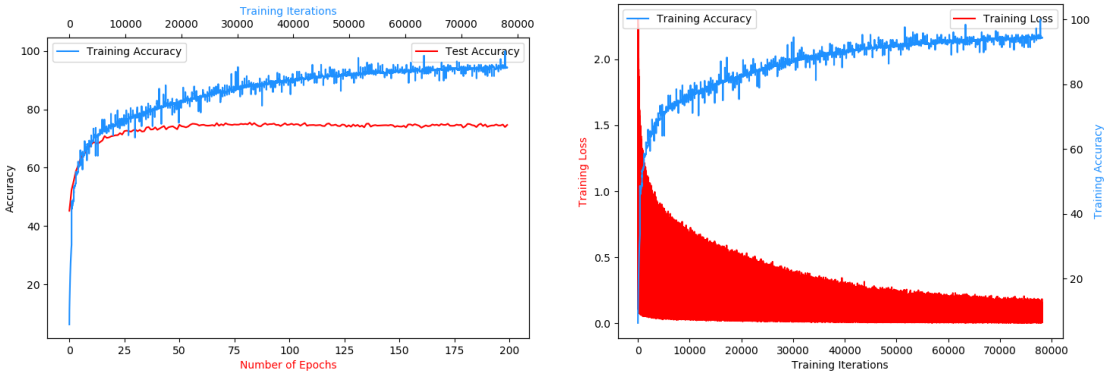
```
transform = transforms.Compose(
    [transforms.Resize(32),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

Listing 3: Transforms from image pre-processing

After this, the dataset was properly loaded using PyTorch's own data loading routines. (It was automatically divided into training, validation, and testing sets. Back in the old days, one had to do that from scratch.) The network was then trained and tested at the end of each epoch.

3. Results

The network was trained for 200 epochs with a learning rate of 0.001, a batch size of 128 (on a modest NVIDIA GPU, of course), and a dropout rate of 0.5. Using these parameters, we achieved the best test accuracy of **75.46%**.
Figure 1a shows both the training and test accuracies, while figure 1b shows the training loss versus the training accuracy.
It is clear from figure 1a that this network is over-fitted, and it increasingly becomes over-fitted as it is trained, since the training and test accuracy diverge as the model is further trained. The training accuracy increases despite the network not learning any new information, as shown by the test accuracy.
The network was trained for 200 epochs to show the exaggerated effect of training a deep network for long with no over-fitting preventative measures in place. This can be fixed by either augmenting the data or refining the model definition to avoid many unnecessary hyper-parameters. (Our model has too many hyper-parameters and it could use a max-pooling operation in third or fourth layer.)



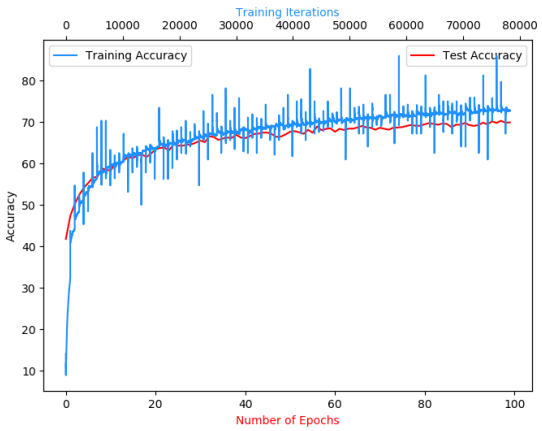
(a) Training and test accuracy

(b) Training loss

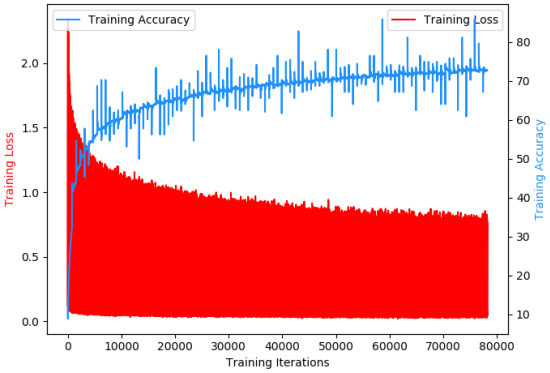
Figure 1: Accuracy and Loss over 200 epochs of training with a dropout rate of 0.5

If we increase the dropout rate to 0.9 (i.e. 90% of parameters will be deactivated), the network becomes less over-fitted as shown in figure 2. However, as shown in figure 2a, the drawback is the loss of accuracy. The accuracy after 100 epochs is around 70 %, compared to 75% that

was previously obtained. No doubt that experimenting with other aspects of the model would increase accuracy while preventing over-fitting.



(a) Training and test accuracy



(b) Training loss

Figure 2: Accuracy and Loss over 100 epochs of training with a dropout rate of 0.9