Assignment 2

Subhadra Mokashe

Problem A – Convolutional Neural Network

Part I: Model

See code: last layer sigmoid is only applied to test set as loss/gradient calculations already take a softmax for the data in the inbuilt function.

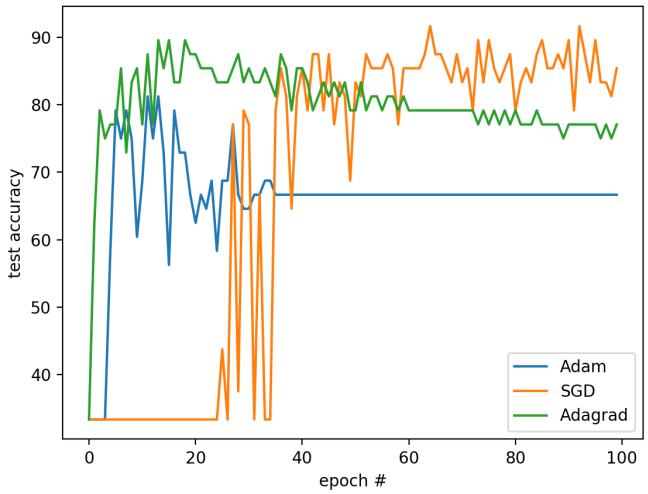
Part II: Performance

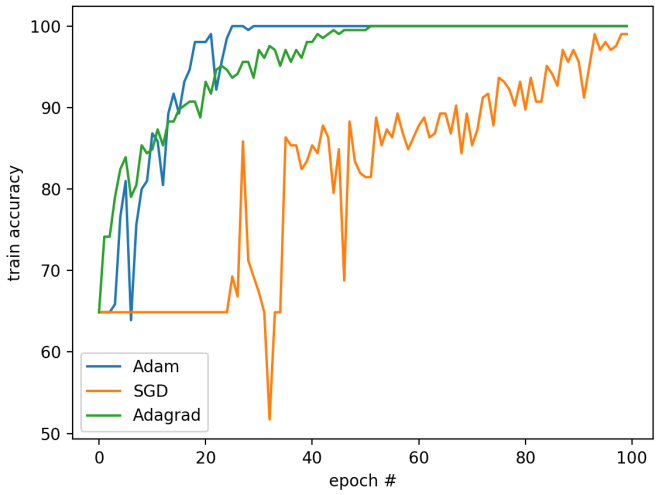
PA-II-1: Run your code and report accuracy over training data and test data (Hint: accuracy in test data is over 60%), show screenshot of your results.

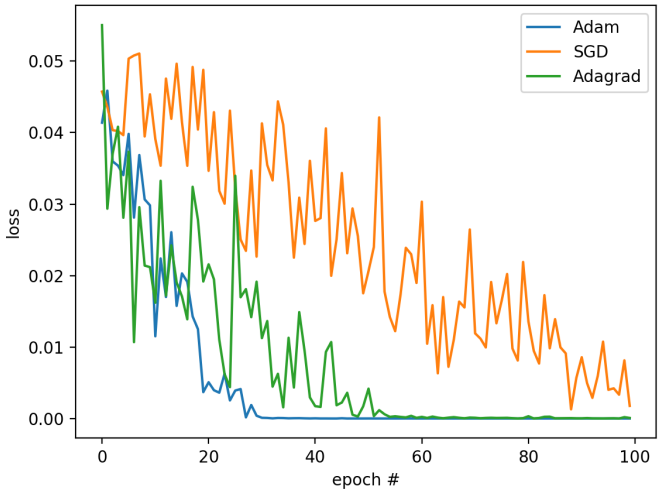
```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        outputs = torch.sigmoid(outputs)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the test images: 85 %
```

PA-II-2: Set epoch number = 100, plot learning curves (test accuracy w.r.t. epoch number) of three optimization algorithms (i.e., SGD, Adagrad, and Adam) and compare them. You can choose different learning rates to make performances of different algorithms good.







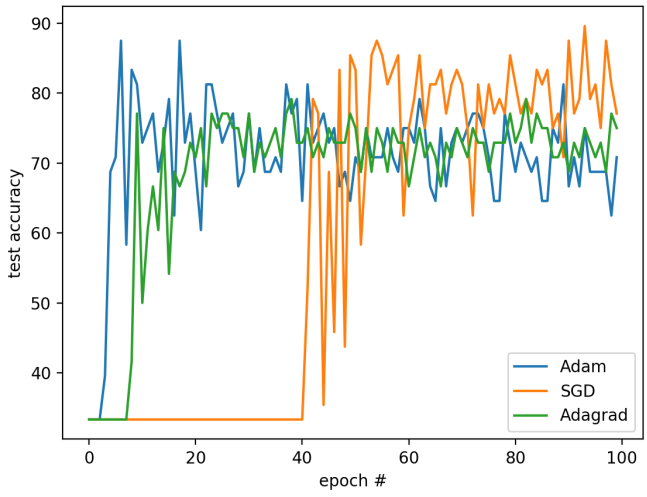
Learning rates: [ADAM,SGD,ADAGRAD] = [0.001,0.01,0.01]. We see that SGD does the best(but the result is not stable) in test accuracy as the loss does not converge quickly as due to the stochastic nature it takes longer to reach high training accuracy. Adagrad does better than Adam optimization and takes longer to converge. Thus longer convergence times gives the scope to better generalize and not overfit.

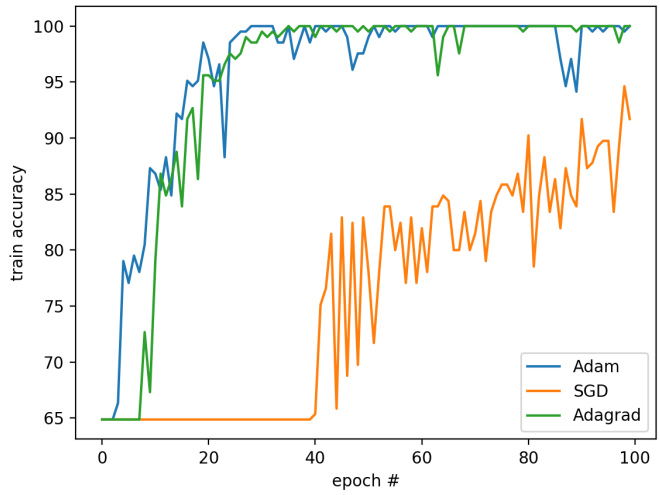
PA-II-3 Use dropout strategy after the second convolution layer and the second fully connected layer (dropout ratio = 0.2). Show your revised code of Net class. Then redo PA-II-2.

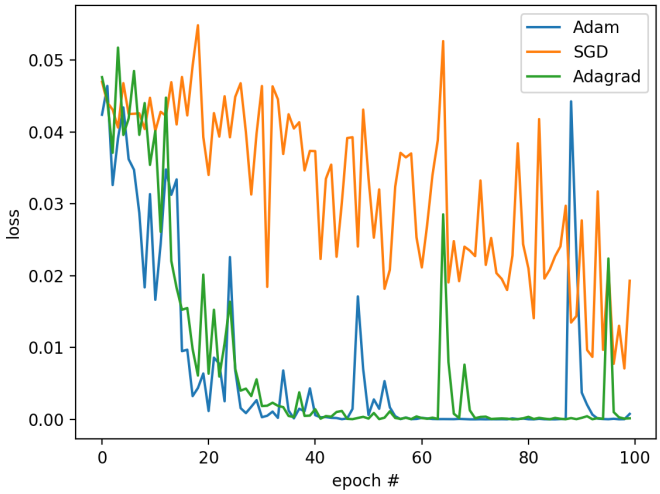
```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6,[5,5],stride=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 12, [5,5],stride=1)
        self.fc1 = nn.Linear(12 * 13 * 13, 120)
        self.fc2 = nn.Linear(120, 64)
        self.dropout = nn.Dropout(0.2)
        self.fc3 = nn.Linear(64, 2)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.dropout(x)
        #print(x.size())
        x = x.view(-1, 12 * 13 * 13)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x
```

```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the test images: 81 %
```







Learning rates: [ADAM,SGD,ADAGRAD] = [0.001,0.01,0.01]. We see that SGD does the best(but the result is not stable as can be seen by a drastic rise in accuracy) in test accuracy as the loss does not converge quickly Adagrad does similar to Adam optimization when drop out is used. The performance in most cases is lower than no droup-out results from the previous sections

Problem B – Residual Neural Network

Part I: Model

PB-I-1 See code:

Part II: Performance

PB-II-1 Here the accuracy was higher with Adagrad than Adam (75%) and the results were not stable. Another implementation was tried as is submitted as a google colab notebook.

```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        outputs = torch.sigmoid(outputs)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the test images: 85 %
```

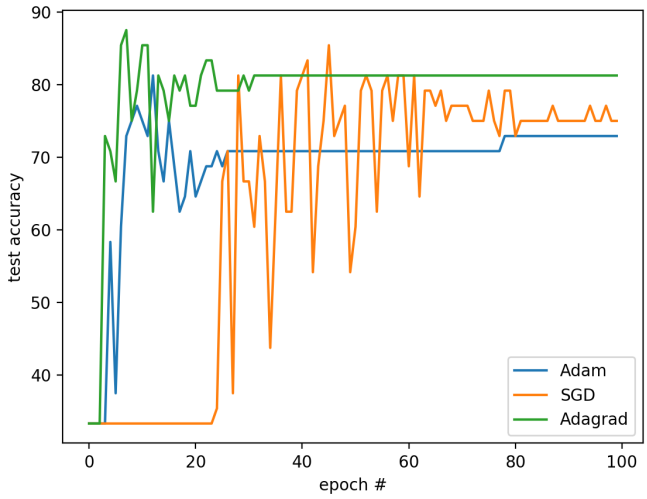
Figure 1: adagrad

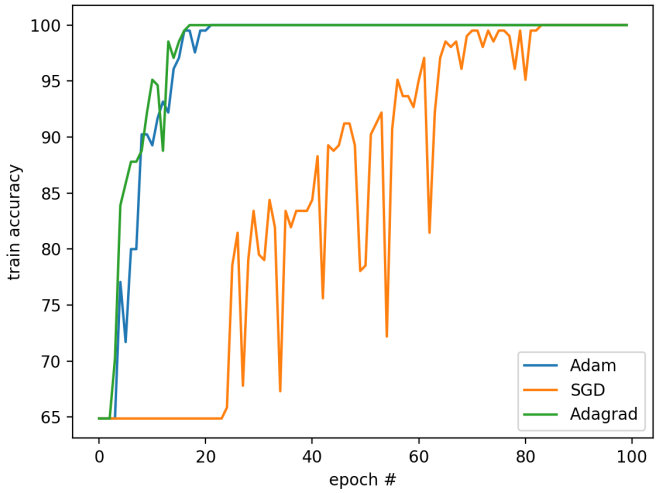
```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        outputs = torch.sigmoid(outputs)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the test images: %d %%' % (
    100 * correct / total))
```

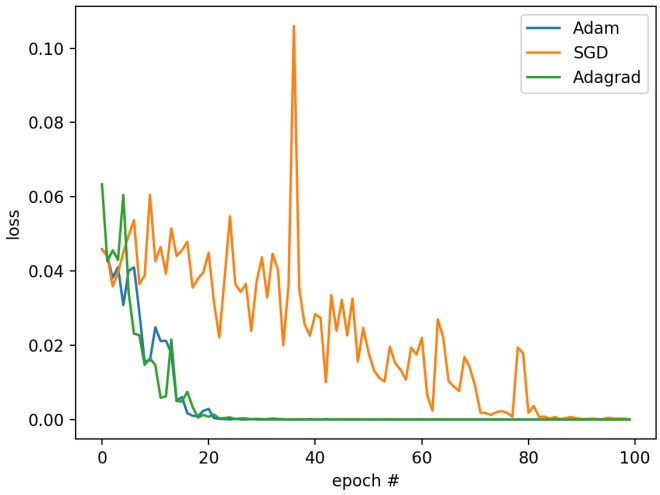
```
Accuracy of the network on the test images: 75 %
```

Figure 2: adam

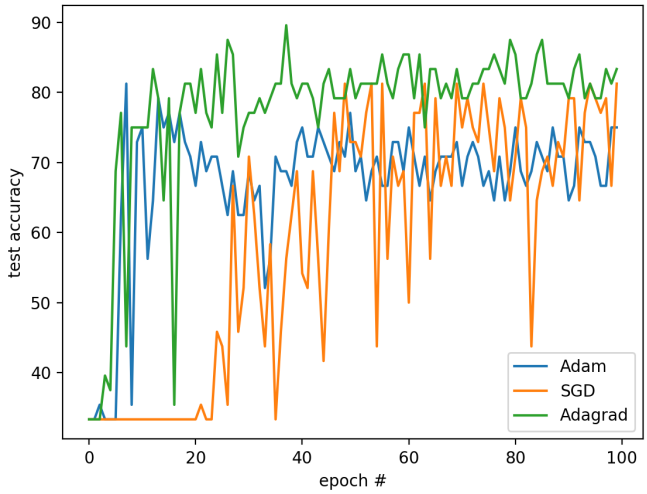
PB-II-2 Set epoch number = 100, plot learning curves (test accuracy w.r.t. epoch number) of three optimization algorithms (i.e., SGD, Adagrad, and Adam) and compare them. You can choose different learning rates to make performances of different algorithms good.

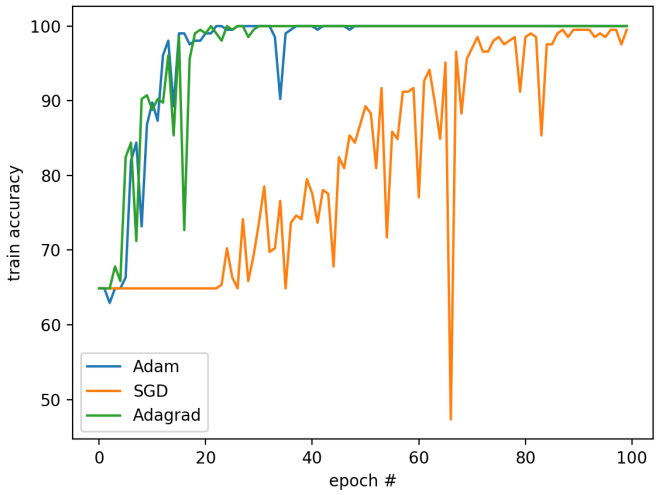

test accuracy
epoch #
Adam
SGD
Adagrad

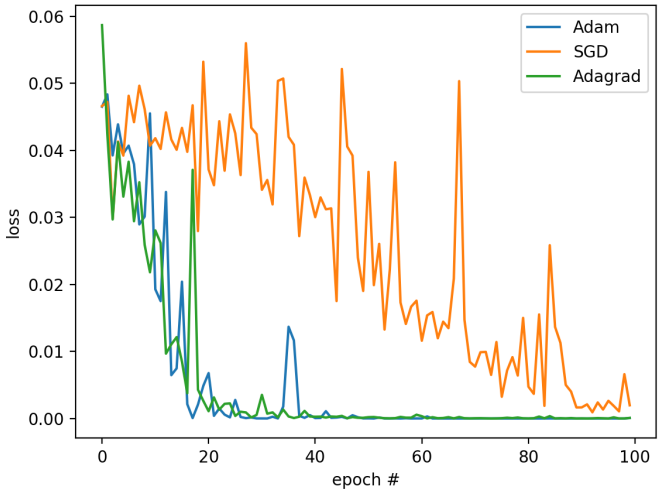

train accuracy
epoch #
Adam
SGD
Adagrad



PB-II-3 I tried dropout strategy after the second convolution layer and the second fully connected layer (dropout ratio = 0.2). It gives better accuracy in most cases.







Learning rates: [ADAM,SGD,ADAGRAD] = [0.001,0.01,0.01]. We see that Adagrad does the best(but the result is not stable). Here longer convergence times don't help much to increase the accuracy in SGD. Results are better when droupout is used.

Disclaimer: I referred to the following public codes: RNN CNN