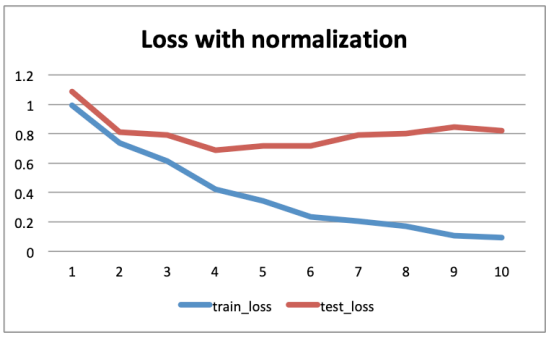
CS 535 Deep Learning

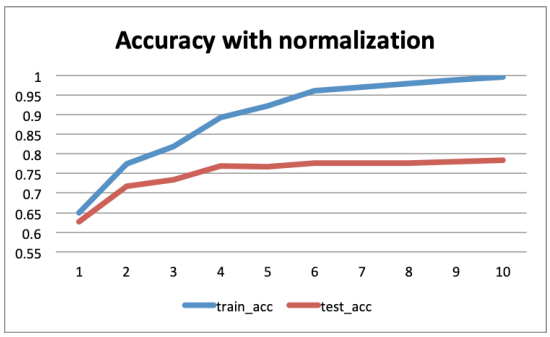
Pytorch CIFAR-10 Image Classification

2020 winter term
Oregon State of University

Instructor: Fuxin Li
Name: Yu-Wen, Tseng
ONID: 933652910
tsengyuw@oregonstate.edu

1) Add a batch normalization layer after the first fully-connected layer (fc1) Save the model after training (Checkout our tutorial on how to save your model). Be careful that batch normalization layer performs differently between training and evaluation process, make sure you understand how to convert your model between training mode and evaluation mode (you can find hints in my code). Observe the difference of final training/testing accuracy with/without batch normalization layer.

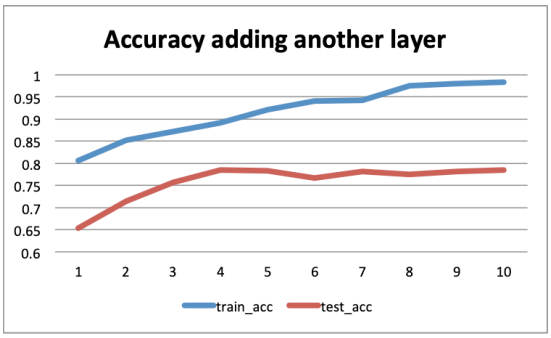




******The TensorBoard figures are in Q5******

EPOCH: 1 train_loss: 0.99352 train_acc: 0.64970 test_loss: 1.08645 test_acc 0.62630
EPOCH: 2 train_loss: 0.73916 train_acc: 0.77446 test_loss: 0.81165 test_acc 0.71790
EPOCH: 3 train_loss: 0.61324 train_acc: 0.81994 test_loss: 0.79024 test_acc 0.73350
EPOCH: 4 train_loss: 0.42192 train_acc: 0.89288 test_loss: 0.68798 test_acc 0.76940
EPOCH: 5 train_loss: 0.34164 train_acc: 0.92168 test_loss: 0.71640 test_acc 0.76800
EPOCH: 6 train_loss: 0.23399 train_acc: 0.96200 test_loss: 0.71585 test_acc 0.77640
EPOCH: 7 train_loss: 0.20726 train_acc: 0.97038 test_loss: 0.78892 test_acc 0.76640
EPOCH: 8 train_loss: 0.10292 train_acc: 0.97938 test_loss: 0.80344 test_acc 0.77650
EPOCH: 9 train_loss: 0.10783 train_acc: 0.98804 test_loss: 0.84623 test_acc 0.78050
EPOCH: 10 train_loss: 0.09385 train_acc: 0.99624 test_loss: 0.82314 test_acc 0.7834

2) Modify our model by adding another fully connected layer with 512 nodes at the second-to-last layer (before the fc2 layer). Apply the model weights you saved at step 1 to initialize to the new model (only up to fc2 layer since after that all layers are newly created) before training. Train and save the model (Hint: check the end of the assignment description to see how to partially restore weights from a pretrained weights file).

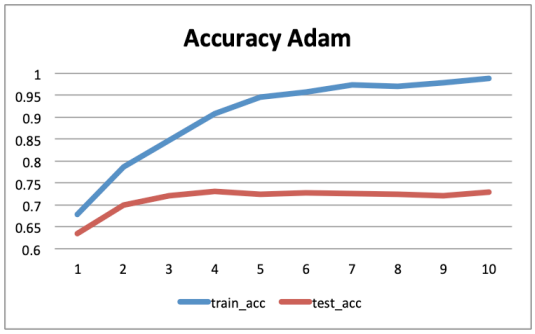


******The TensorBoard figures are in Q5******

EPOCH: 1 train_loss: 0.51642 train_acc: 0.80576 test_loss: 0.78080 test_acc 0.65300
EPOCH: 2 train_loss: 0.47736 train_acc: 0.85306 test_loss: 0.83154 test_acc 0.71500
EPOCH: 3 train_loss: 0.48982 train_acc: 0.87154 test_loss: 0.80697 test_acc 0.75750
EPOCH: 4 train_loss: 0.32691 train_acc: 0.89194 test_loss: 0.75768 test_acc 0.77550
EPOCH: 5 train_loss: 0.23907 train_acc: 0.92156 test_loss: 0.76942 test_acc 0.78510
EPOCH: 6 train_loss: 0.17860 train_acc: 0.94142 test_loss: 0.80130 test_acc 0.78410
EPOCH: 7 train_loss: 0.16312 train_acc: 0.94188 test_loss: 0.92770 test_acc 0.76750
EPOCH: 8 train_loss: 0.08374 train_acc: 0.97462 test_loss: 0.87855 test_acc 0.78090
EPOCH: 9 train_loss: 0.07253 train_acc: 0.98052 test_loss: 0.87272 test_acc 0.78190
EPOCH: 10 train_loss: 0.07845 train_acc: 0.98426 test_loss: 1.02946 test_acc 0.7848

3) Try to use an adaptive schedule to tune the learning rate, you can choose from RMSprop, Adagrad and Adam (Hint: you don't need to implement any of these, look at Pytorch documentation please).

[Adam]: When choosing Adam optimization with lr = 0.0001



******The TensorBoard figures are in Q5******

EPOCH: 1 train_loss: 0.92713 train_acc: 0.67800 test_loss: 1.03459 test_acc 0.63460

EPOCH: 2 train_loss: 0.62724 train_acc: 0.78688 test_loss: 0.84638 test_acc 0.70020

EPOCH: 3 train_loss: 0.45293 train_acc: 0.84782 test_loss: 0.79969 test_acc 0.72130

EPOCH: 4 train_loss: 0.29406 train_acc: 0.90840 test_loss: 0.80426 test_acc 0.73020

EPOCH: 5 train_loss: 0.18257 train_acc: 0.94640 test_loss: 0.86261 test_acc 0.72430

EPOCH: 6 train_loss: 0.14187 train_acc: 0.95716 test_loss: 0.95696 test_acc 0.72770

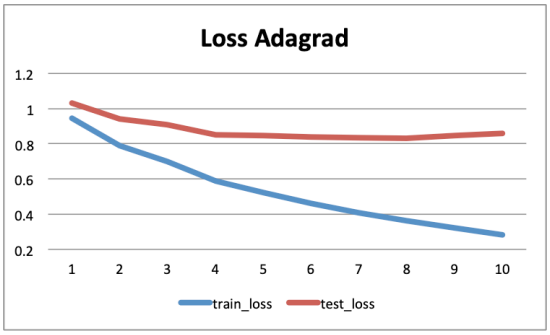
EPOCH: 7 train_loss: 0.09007 train_acc: 0.97392 test_loss: 0.99817 test_acc 0.72540

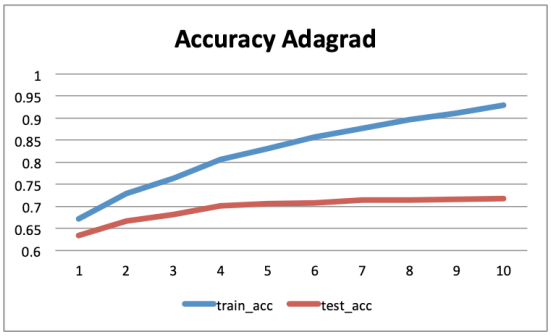
EPOCH: 8 train_loss: 0.09099 train_acc: 0.96972 test_loss: 1.13309 test_acc 0.72350

EPOCH: 9 train_loss: 0.06642 train_acc: 0.97902 test_loss: 1.15304 test_acc 0.72160

EPOCH: 10 train_loss: 0.04027 train_acc: 0.98858 test_loss: 1.16314 test_acc 0.7284

[Adagrad]: When choosing Adagrad optimization with lr=0.001





******The TensorBoard figures are in Q5******

EPOCH: 1 train_loss: 0.94486 train_acc: 0.67152 test_loss: 1.03115 test_acc 0.63400

EPOCH: 2 train_loss: 0.79080 train_acc: 0.72880 test_loss: 0.94026 test_acc 0.66720

EPOCH: 3 train_loss: 0.69910 train_acc: 0.76036 test_loss: 0.90822 test_acc 0.68080

EPOCH: 4 train_loss: 0.58679 train_acc: 0.80692 test_loss: 0.85304 test_acc 0.70160

EPOCH: 5 train_loss: 0.52470 train_acc: 0.83034 test_loss: 0.84782 test_acc 0.70610

EPOCH: 6 train_loss: 0.46335 train_acc: 0.85744 test_loss: 0.83802 test_acc 0.70730

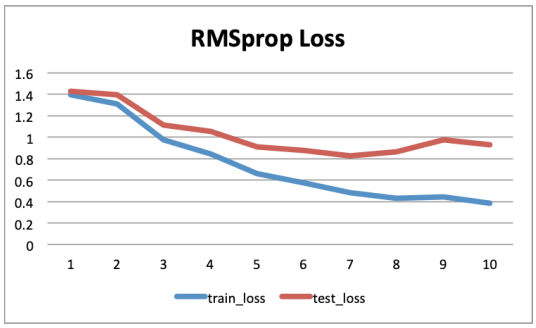
EPOCH: 7 train_loss: 0.41000 train_acc: 0.87724 test_loss: 0.83461 test_acc 0.71390

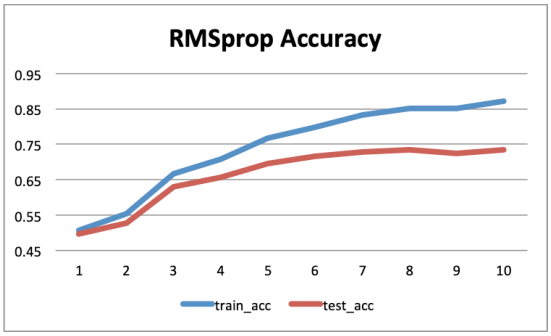
EPOCH: 8 train_loss: 0.36319 train_acc: 0.89658 test_loss: 0.83151 test_acc 0.71360

EPOCH: 9 train_loss: 0.32239 train_acc: 0.91184 test_loss: 0.83698 test_acc 0.71570

EPOCH: 10 train_loss: 0.28261 train_acc: 0.92976 test_loss: 0.83787 test_acc 0.7182

[RMSprop] When choosing RMSprop optimization with lr=0.001





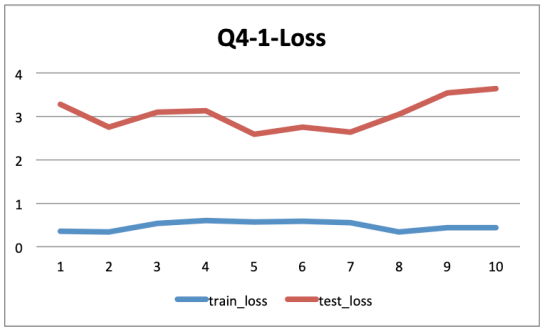
******The TensorBoard figures are in Q5******

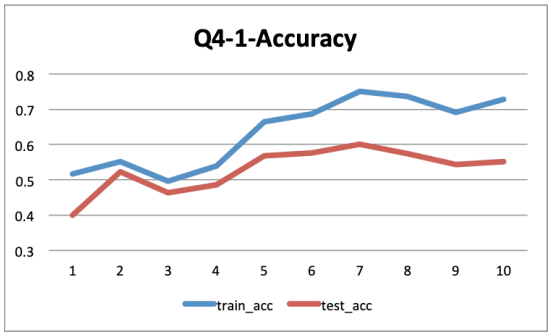
EPOCH: 1 train_loss: 1.39773 train_acc: 0.50638 test_loss: 1.43102 test_acc 0.49660
EPOCH: 2 train_loss: 1.31042 train_acc: 0.55470 test_loss: 1.39322 test_acc 0.52740
EPOCH: 3 train_loss: 0.97713 train_acc: 0.66728 test_loss: 1.11538 test_acc 0.63070
EPOCH: 4 train_loss: 0.84819 train_acc: 0.70788 test_loss: 1.05213 test_acc 0.65750
EPOCH: 5 train_loss: 0.66380 train_acc: 0.76824 test_loss: 0.91112 test_acc 0.69660
EPOCH: 6 train_loss: 0.57817 train_acc: 0.79838 test_loss: 0.87677 test_acc 0.71530
EPOCH: 7 train_loss: 0.48343 train_acc: 0.83222 test_loss: 0.82707 test_acc 0.72810
EPOCH: 8 train_loss: 0.43434 train_acc: 0.85138 test_loss: 0.86560 test_acc 0.73500
EPOCH: 9 train_loss: 0.44448 train_acc: 0.85152 test_loss: 0.97352 test_acc 0.72440
EPOCH: 10 train_loss: 0.38488 train_acc: 0.87206 test_loss: 0.92852 test_acc 0.7345

4) Try to tune your network in another way (e.g. add/remove a layer, change the activation function, add/remove regularizer, change the number of hidden units, more batch normalization layers) not described in the previous four. You can start from random initialization or previous results as you wish.

- I add one batch normal and change hidden units, and remove two convolutional neural network. Still keep the Q1 and Q2 batch and fully connected layer.

```python
#####Q4 - 1#######
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 32, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(32 * 16 * 16, 512)
        self.batchNormal = nn.BatchNorm1d(512) #Q1 Add a batch normalization layer after the first fully-connected layer(fc1)
        self.add_Fc = nn.Linear(512,512) #Q2 adding another fully connected layer with 512 nodes
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = self.batchNormal(x) #Q1
        x = F.relu(self.add_Fc(x)) #Q2
        x = self.batchNormal(x)
        x = self.fc2(x)
        return x
##################
```





******The TensorBoard figures are in Q5******

EPOCH: 1 train_loss: 0.35126 train_acc: 0.51768 test_loss: 3.27989 test_acc 0.39890

EPOCH: 2 train_loss: 0.34267 train_acc: 0.55192 test_loss: 2.74592 test_acc 0.52220

EPOCH: 3 train_loss: 0.54205 train_acc: 0.49594 test_loss: 3.09014 test_acc 0.46420

EPOCH: 4 train_loss: 0.59420 train_acc: 0.53914 test_loss: 3.12315 test_acc 0.48540

EPOCH: 5 train_loss: 0.56216 train_acc: 0.66552 test_loss: 2.58198 test_acc 0.56840

EPOCH: 6 train_loss: 0.59125 train_acc: 0.68642 test_loss: 2.74542 test_acc 0.57620

EPOCH: 7 train_loss: 0.55978 train_acc: 0.75062 test_loss: 2.63697 test_acc 0.60010

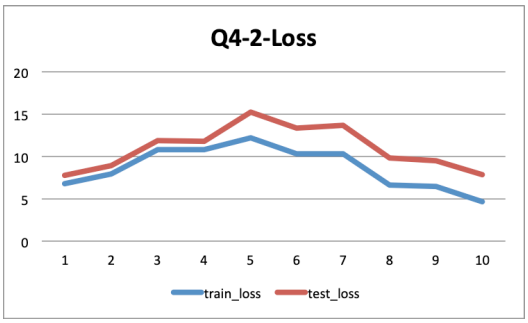
EPOCH: 8 train_loss: 0.33702 train_acc: 0.73558 test_loss: 3.04541 test_acc 0.57480

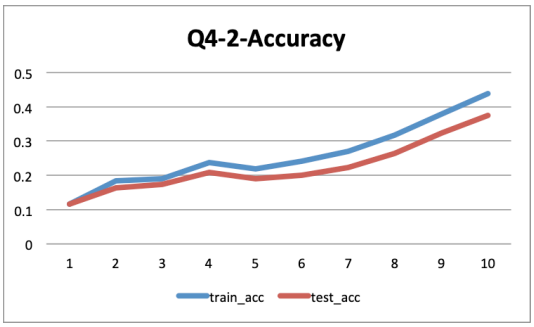
EPOCH: 9 train_loss: 0.44475 train_acc: 0.69170 test_loss: 3.53752 test_acc 0.54450

EPOCH: 10 train_loss: 0.43957 train_acc: 0.72874 test_loss: 3.63074 test_acc 0.5518

- Adding two convolutions neural network layers and one batch normal, and change hidden unit. Then changing activation function to Relu. Still keep the Q1 and Q2 batch and fully connected layer.

```python
##########Q4-2##########
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv4 = nn.Conv2d(64, 64, 3, padding=1)
        self.conv5 = nn.Conv2d(64, 128, 3, padding=1)
        self.conv6 = nn.Conv2d(128, 128, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(128 * 4 * 4, 512)
        self.batchNormal = nn.BatchNorm1d(512) #Q1 Add a batch normalization layer after the first fully-connected layer(fc1)
        self.add_Fc = nn.Linear(512,512) #Q2 adding another fully connected layer with 512 nodes
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.pool(x)
        x = F.relu(self.conv5(x))
        x = F.relu(self.conv6(x))
        x = self.pool(x)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = self.batchNormal(x) #Q1
        x = F.relu(self.add_Fc(x)) #Q2
        x = self.batchNormal(x)
        x = self.fc2(x)
        return x
######################

    def num_flat_features(self, x):
```





******The TensorBoard figures are in Q5******

EPOCH: 1 train_loss: 6.76159 train_acc: 0.11574 test_loss: 7.74452 test_acc 0.11620

EPOCH: 2 train_loss: 7.91830 train_acc: 0.18474 test_loss: 8.90453 test_acc 0.16430

EPOCH: 3 train_loss: 10.82407 train_acc: 0.19028 test_loss: 11.8618 test_acc 0.1742

EPOCH: 4 train_loss: 10.77770 train_acc: 0.23748 test_loss: 11.8325 test_acc 0.2079

EPOCH: 5 train_loss: 12.16222 train_acc: 0.21824 test_loss: 15.27015 test_acc 0.191

EPOCH: 6 train_loss: 10.33041 train_acc: 0.24232 test_loss: 13.39213test_acc 0.2011

EPOCH: 7 train_loss: 10.33559 train_acc: 0.26950 test_loss: 13.7152 test_acc 0.2231

EPOCH: 8 train_loss: 6.62749 train_acc: 0.31660 test_loss: 9.86063 test_acc 0.26320

EPOCH: 9 train_loss: 6.48812 train_acc: 0.37990 test_loss: 9.51572 test_acc 0.32390

EPOCH: 10 train_loss: 4.65737 train_acc: 0.43906 test_loss: 7.85516 test_acc 0.3758

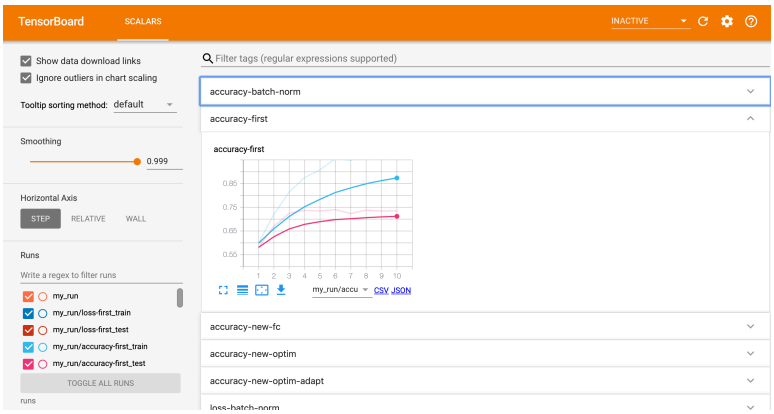
5) Try to use the visualization toolkit, tensorboard, for tracking and visualizing your training process and include them in your report: show the loss anuracy, visualize the model graph, and display images or other tensors as they change over time.

The below code is to use for tensorboard and display images:

```
from torch.utils.tensorboard import SummaryWriter


writer = SummaryWriter(log_dir='./log')

        writer.add_scalars('Loss', {'train_loss':train_loss, 'test_loss':test_loss}, epoch+1)
        writer.add_scalars('Accuracy', {'train_acc':train_acc, 'test_acc':test_acc}, epoch+1)

writer.close()
```

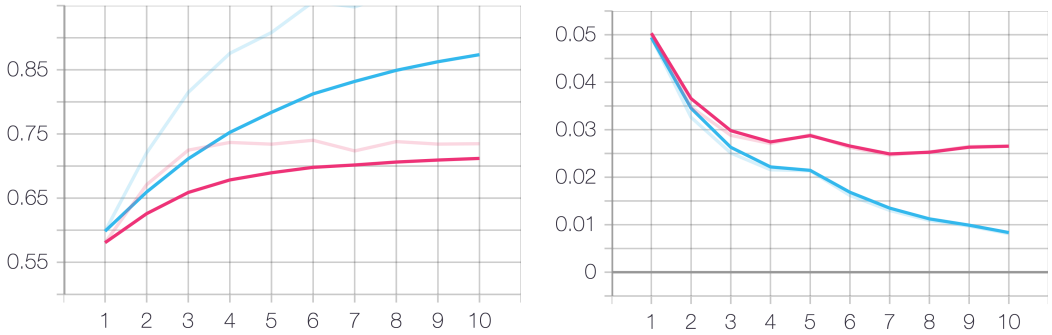
After completely training the data, entering “tensorboard --logdir=log” in command line. And I will get the localhost number before get into the tensorboard website. Then the below figures will display.

The figures for question#1

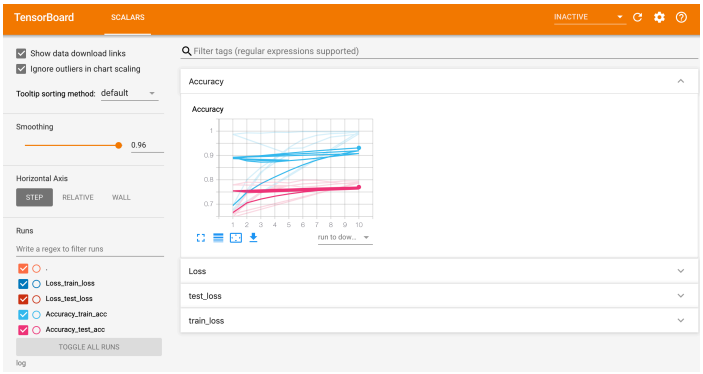


Because the file I download is .svg file, which cannot insert to the PDF, I use the screen shot.

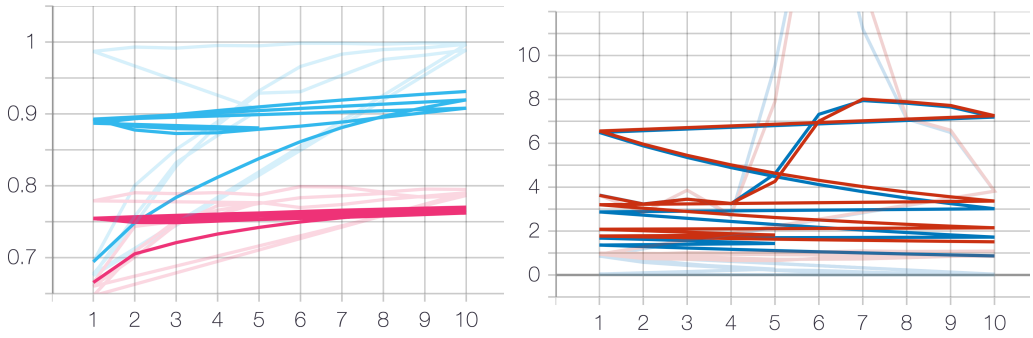
[Accuracy] and [Loss]



The figures for question#2

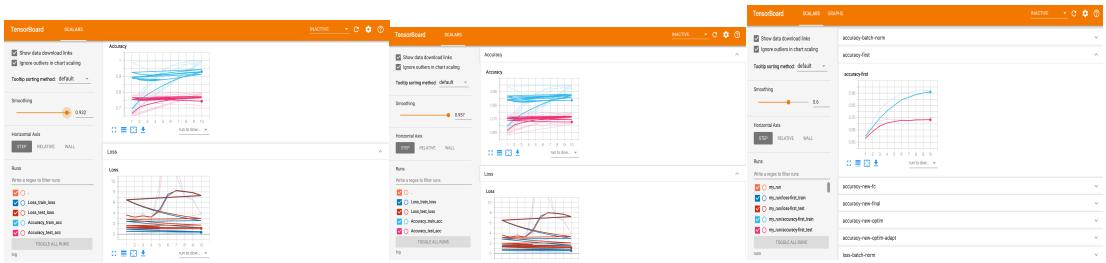


[Accuracy] and [Loss]



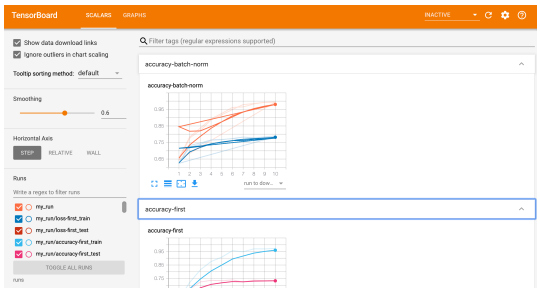
The figures for question#3

[Adam] [Adagrad] [RMSprop]

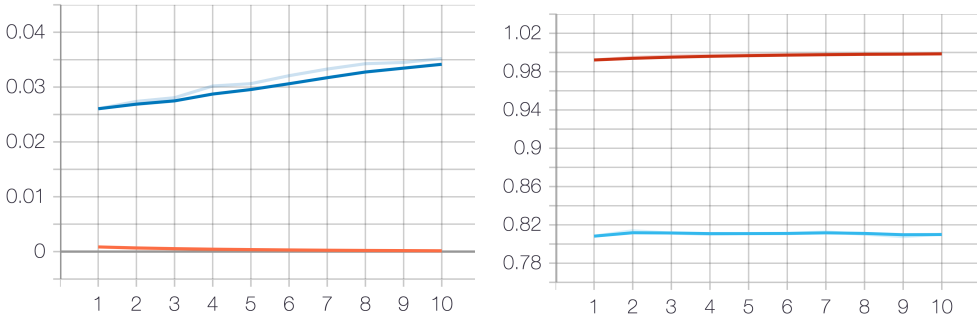


The figures for question#4

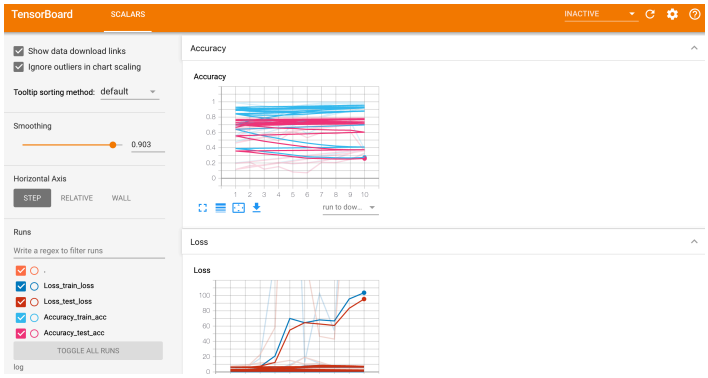
[Method 1]



[Loss] and [Accuracy]


0.04
0.03
0.02
0.01
0
1 2 3 4 5 6 7 8 9 10
1.02
0.98
0.94
0.9
0.86
0.82
0.78
1 2 3 4 5 6 7 8 9 10

[Method 2]


TensorBoard
SCALARS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.903
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
Loss_train_loss
Loss_test_loss
Accuracy_train_acc
Accuracy_test_acc
TOGGLE ALL RUNS
log
Accuracy
Accuracy
run to dow...
Loss
Loss