TTIC31230 - ProblemSet2

Name: Alex Enze Liu

Date: January 17 2017

1. Problem 1

The upper bound of the equation is:

$$||\nabla_w l_{train}(w) - \nabla_w l_{generalize}(w)|| <= \frac{\gamma}{\sqrt{N}} \tag{1}$$

$$\text{where } \gamma = 2b * (1 + \sqrt{2ln\delta})$$

Proof.

First we know that,

$$||\nabla_w l(w, x, y)|| < b \tag{2}$$

Then, based on (2) we have,

$$||\nabla_w l_{generalize}(w)|| = ||E[\nabla_w l(w, x, y)]|| \leq ||E[\nabla_w l(w, x, y)]|| \leq b \tag{3}$$

Based on (3), we can have,

$$||\nabla_w l_(w, x_n, y_n) - \nabla_w l_{generalized}(w)|| \leq b + b = 2b \tag{4}$$

Given the inequality formula from the slides, we have,

$$||\nabla_w l_{train}(w) - \nabla_w l_{generalized}(w)|| \leq \frac{\gamma}{\sqrt{N}} \qquad w.p. \quad 1 - \delta \tag{5}$$

$$\text{where } \gamma = 2b * (1 + \sqrt{2ln\delta})$$

□

2. Problem 2

Note: I think the provided code is probably wrong in function "eval", I changed the code a little bit and all the graphs generated in my ipydb files are based on the modified code. Figure 1 is the snippet of the possible wrong code.

```
def eval(imgs, labels):

    batches = range(0, len(labels), batch)
    objective = 0
    accuracy = 0
    for k in batches:
        inp.set(t_imgs[k:k+batch]) #The labels here should be imgs not t_imgs
        lab.set(t_labels[k:k+batch]) #The labels here should be labels not t_labels
        edf.Forward()
        objective += np.mean(loss.value)
        accuracy += acc.value

    return accuracy/len(batches), objective/len(batches)
```

Figure 1: Code that is possibly wrong

2.1. Problem 2a

The results of problem 2a are shown in figure 1. I have the batchsize = 10, 100 and eta = eta(batchsize), 0.37

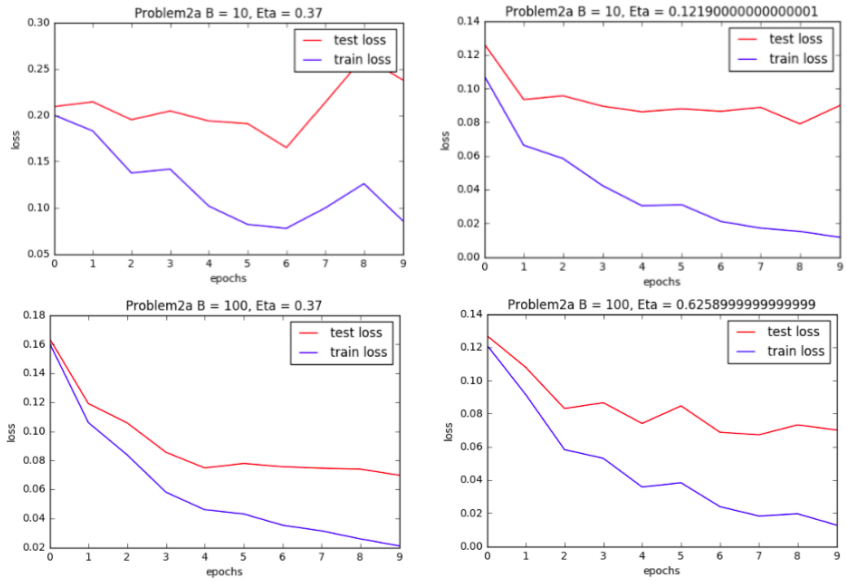

Figure 2: Loss in problem 2a

Let eta denote the learning rate and B denote the batch size. As is shown in figure 2, I have several findings:

- When B is equal to 10, we can see that eta = 0.37 is probably too high, one reason is that the loss decreases fast in the beginning, but later it is hard to converge to the minimum. Another reason is that we can see a sharp increase around epoch 8. By comparison, with smaller learning rate eta = 0.12, the loss decreases slowly and is closer to the minimum.
- When B = 100, it has the same situation; Larger eta will stuck at some point with higher loss value. By comparison, a small learning rate may result in a slow decrease in loss function, but I think in the end it will converge to a smaller value of loss.

2.2. Problem 2b

Let eta denote the learning rate and B denote the batch size. In problem 2b we fix $\mu = 0.55$. The results of problem 2b is displayed in figure 3.

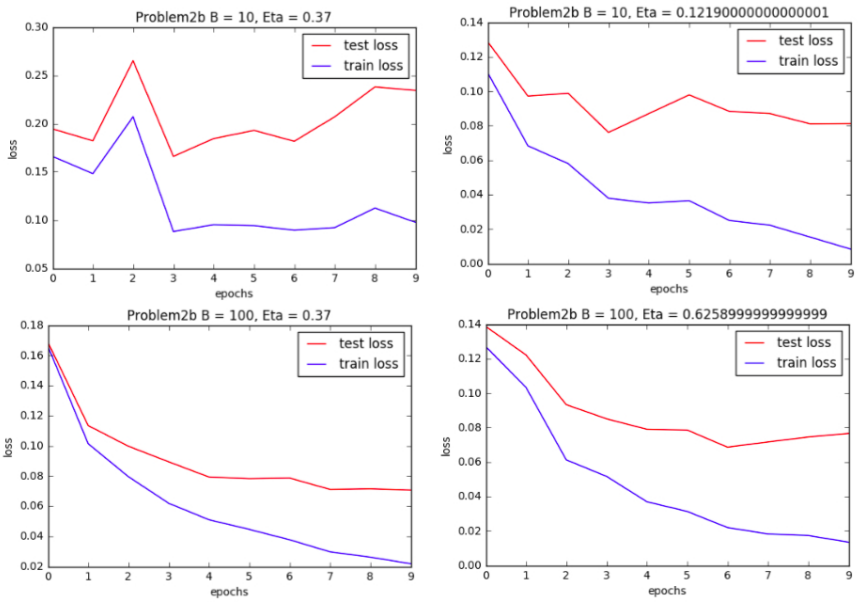

Figure 3: Loss in problem 2b

Figure 3 is very similar to figure 2, except the one on the top left (B = 10, eta = 0.37). Based on the figure we can have several findings, and the first one is similar to those mentioned above.

- For B = 10 and 100, a higher learning rate tends to decreases quickly in

the beginning and ends up in a higher loss value. By contrast, a lower learning rate has a smoother curve and is closer to the minimum.

- Combining the results from problem 2a and problem 2b, it can be seen that networks with B = 100 and eta = 0.37 always have a smooth decrease in loss value.

- The image on the top left is worth noticing here. It is clear that momentum "overshoots" the target, and it has a increase in loss rate in the very beginning. However, overall speaking, it gets to the minimum much faster (reaches 0.1 quite early in epoch 3) than SGD. However, in some caes, SGD and momentum have almost the same results.

2.3. Problem 2c

The results from problem 2c are displayed in figure 4 (B = 10) and figure 5 (B = 100).

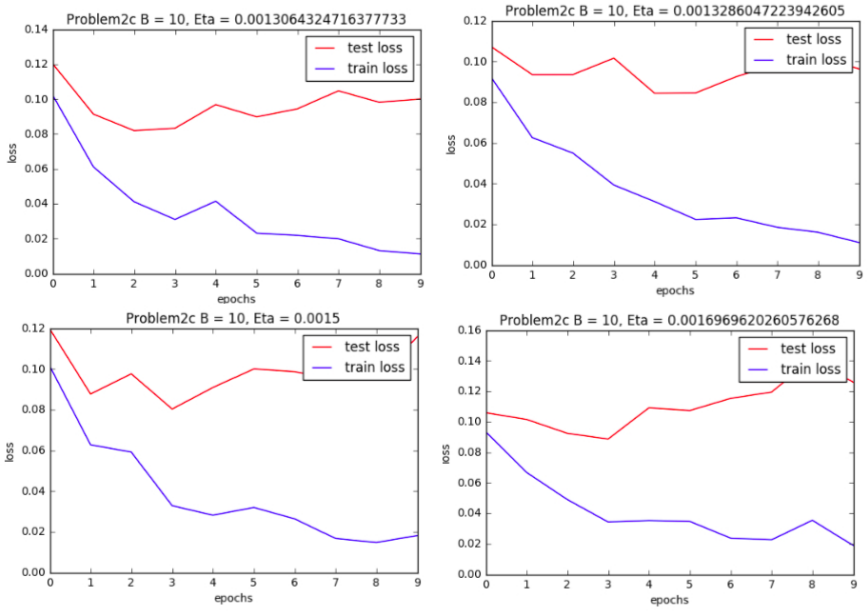

Figure 4: Loss in problem 2c B = 10

Figure 4 above displays results for B = 10. Based on figure 4, We can have the observations:

- The increase of loss value appears in all the 4 images.

- It is very clear that learning rates around 0.0013 have smaller loss values (only 0.10) than that of 0.0015 (0.20) and 0.0016 (0.20). So we might find a better learning rate around 0.0013 when B = 10.

Figure 5 below displays results for $B = 100$. Since the difference between data is very small, to observe the results better, I also provided table 1 here.

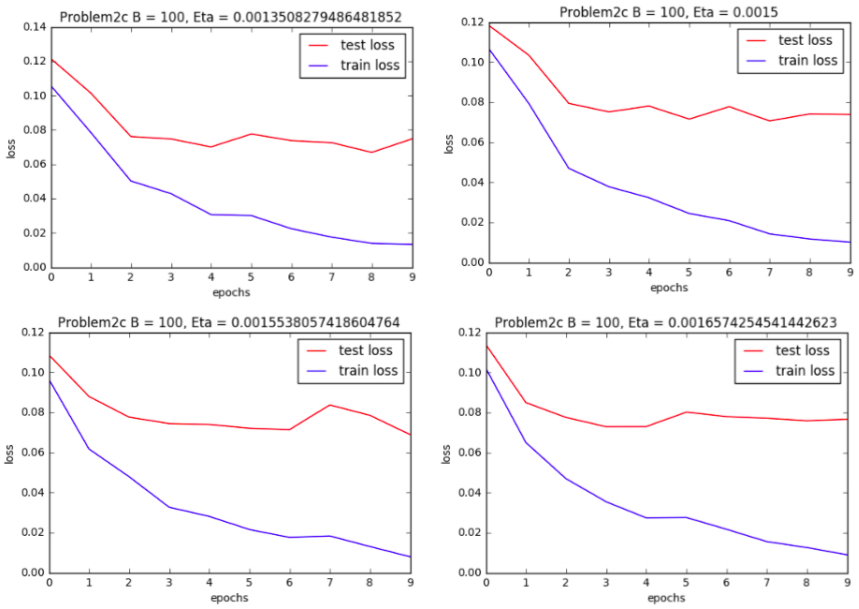

Figure 5: Loss in problem 2c B = 100

Table 1: Loss value in the end for problem 2c

	eta = 0.001350	eta = 0.001500	eta = 0.001553	eta = 0.001657
Loss Value	0.0133	0.0101	0.0079	0.0089

Based on figure 5 and table 1, We can have the observations:

- All the curves have similar shapes.
- As can be seen from table 1, learning rates around 0.00155 have smaller loss values. So we might find a better learning rate around 0.00155 when B = 100.

2.4. Problem 2d

I think possible explanations to explain the results from problem2a to problem2c are as follows:

- One obvious observation is that all high learning rates tend to stuck somewhere with high loss value. The reason might be that with high learning rate we will skip the optimal solution, and just jumping around it.
- We have also seen that momentum sometimes produces nearly the same result as SGD. The possible reason is that the gradient of each step is similar so momentum has nearly the same result each time.
- We can see problem 2a and problem 2b that networks with B = 100 and eta = 0.37 always have the smoothest curve. My interpretation for this is that here we might have a good combination of batch size and learning rate. In other words, we have a moderate batch size and a proper learning rate together that provide both the ability to converge and jump out of shallow valleys.
- We see that in problem 2c B = 10 has the optimal value smaller than 0.0015 while B = 100 has the optimal value greater than 0.0015. I think one possible explanation is that larger batchsize tends to reduce the variance of your stochastic gradient updates, so with a bit higher learning rate it can reach the minimum faster. By comparison, small batchsize tends to be noisier, so it has to be more careful in terms of updating parameters.