Exeriese 2

Lan Zhang
954517477

1 Problem 1

Solved on 09/17/2019. The seed I used for noise generation is 2612.

1.1 Loss Function

Given an input x with an output y, the prediction is $\hat{y}$. The performance of squared loss function shows in Figure1.

The formulation of squared loss is:

$$squared\_loss = \frac{1}{N}\sum_{0}^{N}(y_i - \hat{y_i})^2$$

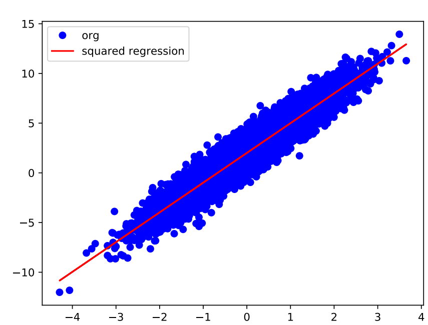

Figure 1: Squared loss

Given the parameter delta = 1.0. The formulation of huber loss is:

$$huber\_loss = \begin{cases} 0.5 * (y - \hat{y}) & if|(y - \hat{y})| <= d \\ d * |(y - \hat{y})| - 0.5 * d^2 & if|(y - \hat{y})| > d \end{cases}$$

The performance of huber loss function shows in Figure2.

I also implement the following hybrid loss function and the performance shows in Figure 3.

$$hybrid\_loss = huber\_loss + |y - \hat{y}|$$

If I didn't use $tf.reduce\_mean$ in squared loss, the result of parameters W and b will turn out to be NaN. I found the issue will happen on tensorflow 2.0. But on tensorflow 1.14, sometimes it can output a correct answer. I'm not sure about the reason. but I think it maybe because something lead to overflow at some point during gradient descent and cause NaNs.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

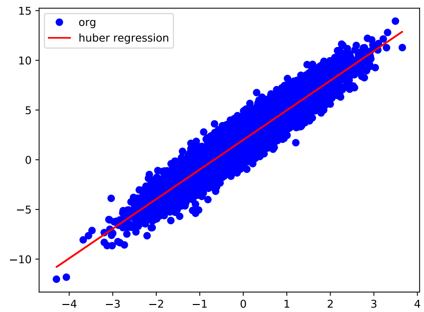

Figure 2: Huber loss

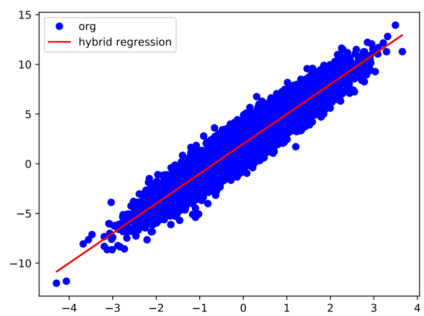

Figure 3: Hybrid loss

It's hard to tell which loss function works better(see Figure4). I've run the code with those two loss function multiple times and both of them work well. If I use the distance of the predicted parameters $\hat{W}, \hat{b}$ and original parameters W, b as a measure of the performance($distances = |W - \hat{W}| + |b - \hat{b}|$), the squared loss(.0133321) work better than huber loss(.0238144).

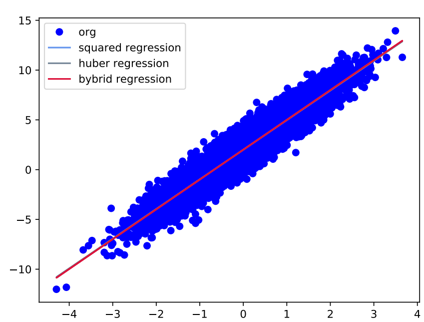

Figure 4: Performance of different loss

1.2 Learning rate

I implemented patience scheduling. Then I tested learning rate from 0.001 to 0.01 with the incrementation 0.002, Shown in Figure 5. Table 1 shows the parameters learned under different learning rate with patience scheduling. The result are quite similar. All of them approximate to the correct parameters.

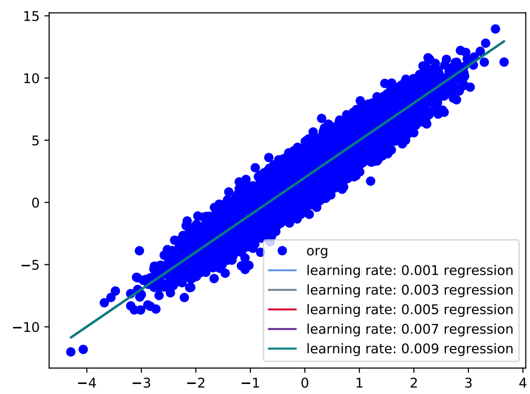

Figure 5: Learning rate

Table 1: The performance of different learning rate

Learning rate	W	b
0.001	2.9962702	2.0017185
0.003	2.9964287	2.001716
0.005	2.996503	2.0016577
0.007	2.996541	2.001543
0.009	2.9966085	2.0013998

1.3 Training steps

I've tested different training steps from 500 to 1500 with the incrementation 200. Shown in Figure 6. With longer duration, the performance tend to be better. However, after "enough" steps, this effects start to stabilize (see Table 2).

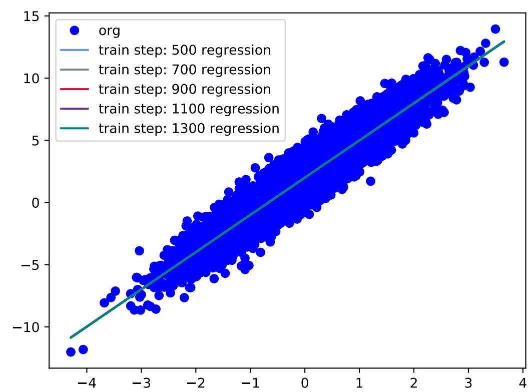

Figure 6: Training steps

1.4 Initial Value of parameters

Table 3 and Table 4 respectively denote predicted parameters with different initial parameter W and b(both of them are from -1000 to 1000 with the incrementation 500).

Table 2: The performance of different training steps

Training steps	W	b
500	2.9877446	1.9952948
700	2.995941	2.001462
900	2.9962356	2.0016909
1100	2.9962835	2.001727
1300	2.9962983	2.0017405

Table 3: The performance of different W after 1000 epochs

Initial W	W	b
-1000	-984.0698	-0.16686472
-500	-484.0088	-0.12897438
0	2.9962702	2.0017185
500	484.0088	0.2609221
1000	984.0698	0.22580884

Table 4: The performance of different b after 1000 epochs

Initial b	W	b
-1000	-0.19055721	-979.8584
-500	-0.19055721	-479.8584
0	2.9962702	2.0017185
500	0.19055721	479.8584
1000	0.19055721	979.8584

If I initialize the parameter W with a small number like -1000 with hybrid Loss, the predicted parameter W will become -984.0698 after 1000 epochs. That is because the model has not yet converged after 1000 epochs. Figure 7 is the performance of different traing epoch with the initial parameter $W = -100$. Desipt the predicted parameters after 9000 epochs are far from the correct answer, it is gradually approaching to the correct answer.

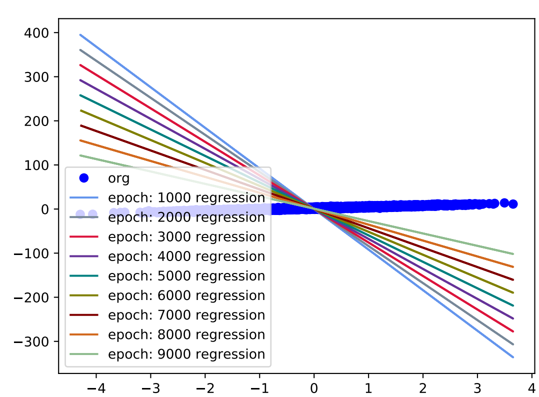

Figure 7: Training duration with a large initiail parameter

1.5 Noise in data

Before this section, I generated noise in data using normal distribution. Table 5 shows the results of different noise distribution. The distribution of noise can affect the prediction. Table 6 shows the results of normal distribution with different mean. Table 7 shows the results of normal distribution with different std. The level of noise is higher, the model is harder to converge.

Table 5: The performance of different noise in data

Noise distribution	W	b
normal(mean=0.0, stddev=1.0)	2.9794447	1.9967409
gamma(alpha=1)	2.98501	2.8410964
uniform(minval=0)	2.9989622	2.502729

Table 6: The performance of normal distribution with different mean

mean	W	b
0.0	2.9794447	1.9967409
1.0	2.964156	2.990973
2.0	2.9483728	3.9745786
3.0	2.924222	4.9402742

Table 7: The performance of normal distribution with different std

mean	W	b
1.0	2.9794447	1.9967409
2.0	2.8706882	1.9283952
3.0	2.6891909	1.8024324

1.6 Noise in data during training

First, I tested adding noises(normal distribution: $mean = 0.$, $stddev = 1.$) in data per epoch. The predicted parameters are $\hat{W} = 1.4970689$, $\hat{b} = 1.8999192$.

Then, I added noises(normal distribution: $mean = 0.1$, $stddev = 0.1$) in weights per epoch. The predicted parameters are $\hat{W} = -15.267623$, $\hat{b} = 1.1710013$.

Finally, I added noises(normal distribution: $mean = 0.001$, $stddev = 0.002$) in learning rate per epoch. The predicted parameters are $\hat{W} = 2.9953077$, $\hat{b} = 2.007313$.

Noise in data and weight will change the performance of the model. If noise in learning rate is reasonable, it may affect training time. But it will not change the general performance of the model, except it reaches local minima. For example, it may lead to faster convergence or make the model hard to converge.

On other classification problems and mathematical models, noise in data and weight will also reduce model's performance. But adding small noise in learning rate may make it possible to get rid of local minima.

1.7 Significance of seed

If I use the same seed to generate noise, I will get the same results every time I run the code without change any parameters. Because the pseudorandom number generating algorithms is designed to performing operation on a seed. So the result of algorithms is determined by the seed. If I want to change the result every time, I can use current time as the seed.

1.8 GPU VS CPU

I tested training time with GPU and CPU on Calab. I used each of them to run the same code for 1000 epochs. And the approximate running time per epoch of CPU is 0.030322 and of GPU is 0.073198.

The training time looks unreasonable. I think it may be because it's not the correct running time considering the network speed and the design of Calab.

2 Problem 2

Solved on 09/21/2019. I implemented two model for fminist. The first one is Vgg16(shown as Figure 8). Another one is multilayer CNNs with batch norm. The accuracy of Vgg16 on test dataset can achieve 92.43% after 14 epochs. The following reports is based on multilayer CNNs model not Vgg16, because Vgg16 takes for a while per epoch.

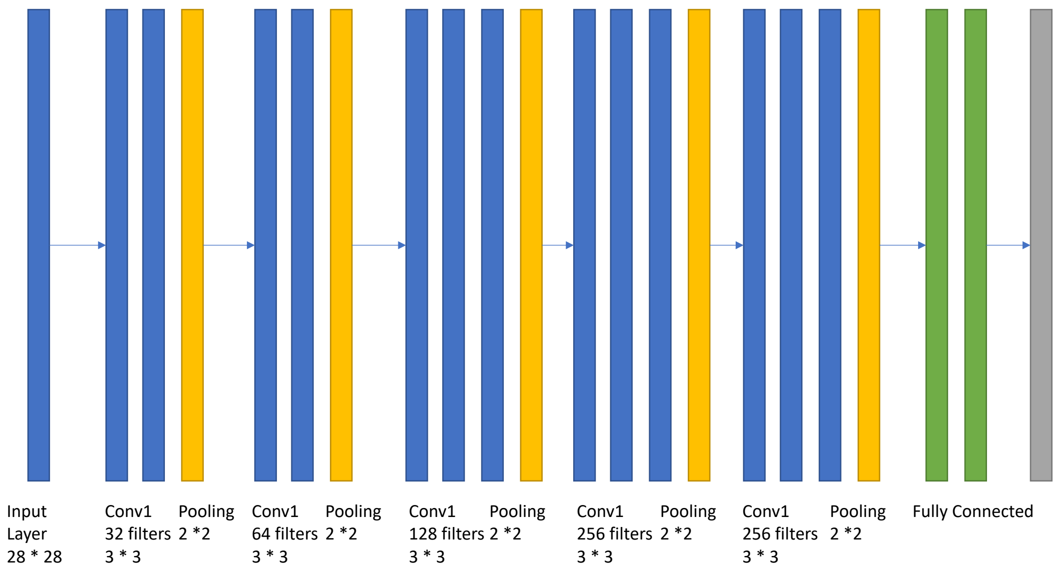

Figure 8: Vgg16

2.1 Optimizer

2.2 Train/Val Split

2.3 Batch size

2.4 GPU VS CPU

2.5 Dropout

2.6 Performance