1 Question 1

Consider the linear measurement $b = Ax + e$, where b is the measurement value of 50 dimensions, A is the measurement matrix of 50 x 100 dimensions, x is the unknown sparse vector of 100 dimensions and the sparsity is 5, e is the measurement noise of 50 dimensions. A normalized least-squares model for recovering x from b and A is as follows:

$$min\frac{1}{2}\|Ax - b\|_2^2 + p\|x\|_1$$

Where p is the non-negative regularization parameter. Please design the following algorithm to solve the problem:

1. Proximal Gradient(PG)
2. Alternating Direction Method of Multipliers(ADMM)
3. Subgradient
4. Adjacent-point Newton Method (optional)

In the experiment, let the non-zero elements in the truth value of x obey the gaussian distribution with mean value 0 and variance 1, the elements in A obey the gaussian distribution with mean value 0 and variance 1, and the elements in e obey the gaussian distribution with mean value 0 and variance 0.1. For each calculation method, please provide the distance between the calculated result and the truth value in each step and the distance between the calculated result and the optimal solution in each step. In addition, please discuss the influence of normalizing parameter p on the calculation results.

2 Solution 1

Since the target function is given, we can calculate the gradient and each iteration is dicided. I directly use Google colab to avoid set up the environments and there is likely to be some conflicts. You could go to my colab to run the code online:https://colab.research.google.com/drive/1TR3CgkuQBeGNcD8jroQZenjFoKayZ_zS

2.1 Proximal Gradient(PG)

Problems that are suitable to be solved by the PG method have the following forms:

$$minf_0(x) = s(x) + r(x)$$

Where s is a function easy to be differentiated while r is a function hard to be differentiated but can be used to calculate the proximal points gradient. The algorithm's main idea is to use differnet gradient-decent method for the two parts and its update steps are:

$$x^{k+\frac{1}{2}} = x^k - \alpha\nabla s(x^k)$$

$$x^{k+\frac{1}{2}} = argmin_x((\frac{p}{2}\|x\|_1 + \frac{1}{2*\alpha}\|x - x^{k+\frac{1}{2}}\|^2)$$

The target function in this question can be easily divided into such 2 parts and the second part is a vector' 1st norm which can use Proximal-point projection to approximate gradient (deduced examples in class). The expression for this problem is directly calculated here:

$$x^{k+\frac{1}{2}} = x^k - 2 * \alpha * A^T(Ax^k - b)$$

$$x^k = \begin{cases} x^{k+\frac{1}{2}} + \alpha * \frac{1}{2}p, & x^{k+\frac{1}{2}} < \alpha * \frac{1}{2}p \\ x^{k+\frac{1}{2}} - \alpha * \frac{1}{2}p, & x^{k+\frac{1}{2}} > \alpha * \frac{1}{2}p \\ x^{k+\frac{1}{2}}, otherwise \end{cases}$$

The algorithm is implemented according to the above-derived formula and the optimal solution can be updated in the iteration. The calculated results are shown in the following figure:

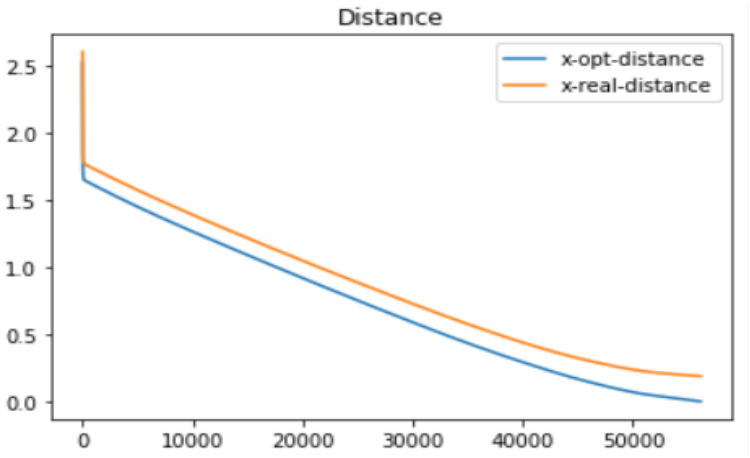


Figure 1: PG

2.2 Alternating Direction Method of Multipliers(ADMM)

Problems that are suitable to be solved by the PG method have the following forms:

$$minf_1(x) + f_2(x)$$

$$s.t.Ax + By = d$$

update mode is as follows:

$$x^{k+1} = argmin_x L_c(x, y^k, v^k)$$

$$y^{k+1} = argmin_y L_c(x^{k+1}, y, v^k)$$
$$v^{k+1} = v^k + (Ax^{k+1} + By^{k+1})$$

Here we turn the second x in the target function into y and get the constraint $x - y = 0$. So we can use the ADMM method:

$$x^{k+1} = \frac{y^k + v^k + 2a^T b}{A^T * A + c * I}$$

$$y^{k+1} = argmin_y(\frac{p}{2}\|y\|_1 + \frac{c}{2}\|y - x^{k+1} + \frac{v^k}{c}\|_2^2)$$
$$v^{k+1} = v^k + c(y^{k+1} - x^{k+1})$$

A similar method is used to find the minimum value point of function with the 1st norm. The algorithm is implemented according to the above-derived formula and the optimal solution can be updated in the iteration. The calculated results are shown in the following figure:

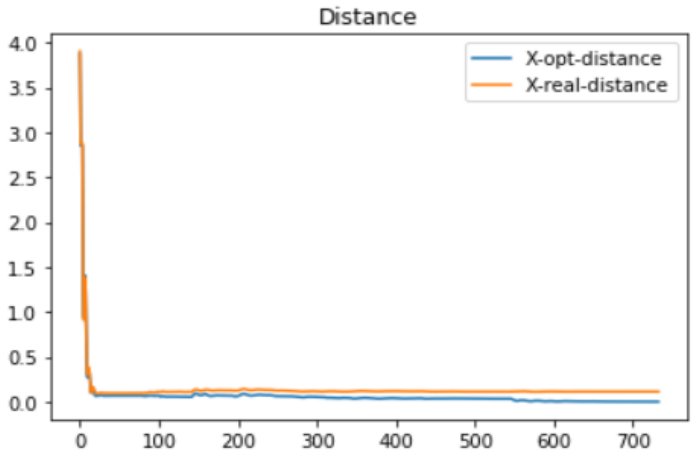


Figure 2: ADMM

2.3 Subgradient

In the sub-gradient method, for a differentiable function, the sub-gradient is its gradient; for non-differentiable points, the sub-gradient is a random value between the gradients on both sides of it. The update mode is as follows:

$$x^{k+1} = x^k - \alpha^k g_0(x^k)$$

The calculated results are shown in the following figure:

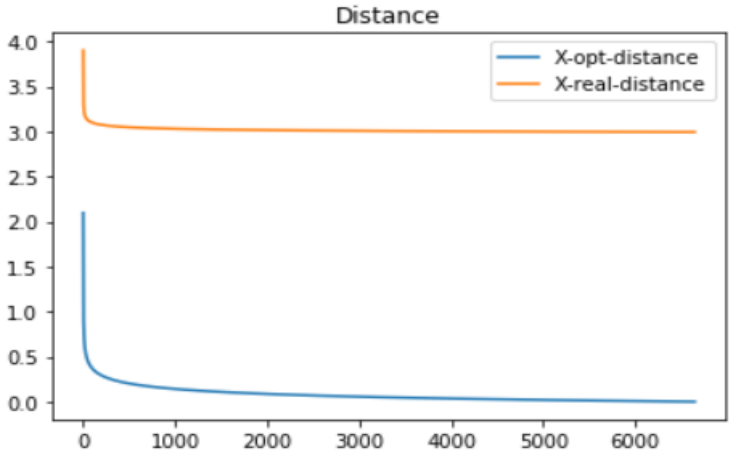
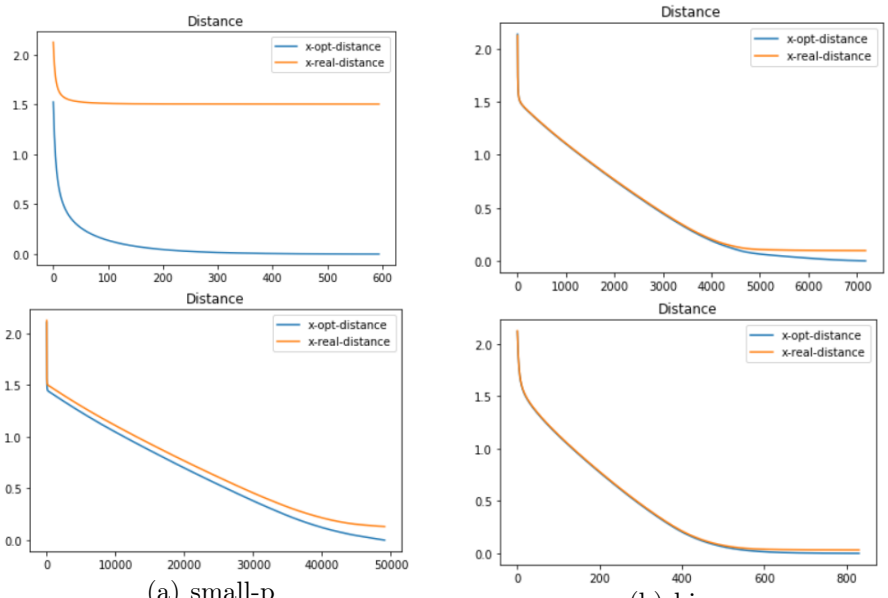


Figure 3: SG

2.4 Influence of p-value

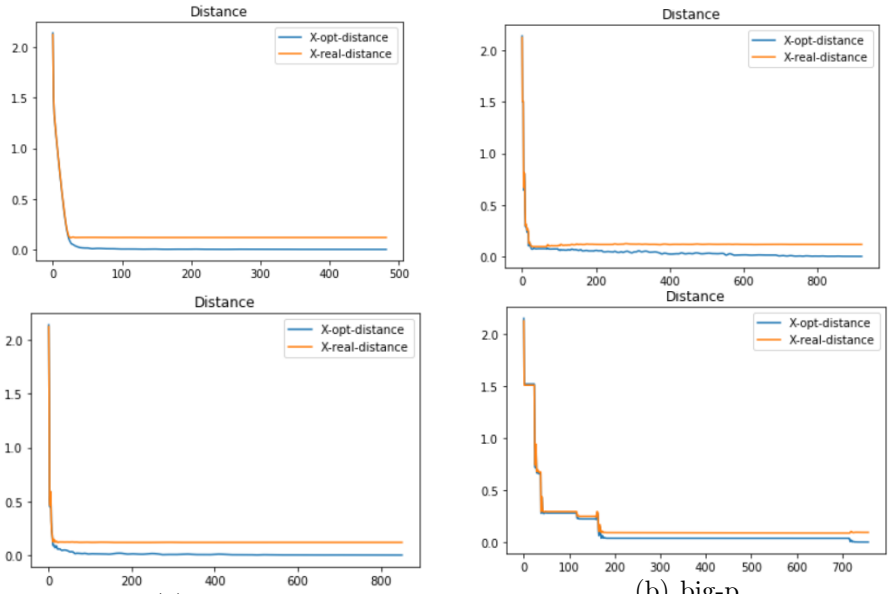
Here is the three methods with different p-value:



(a) small-p (b) big-p

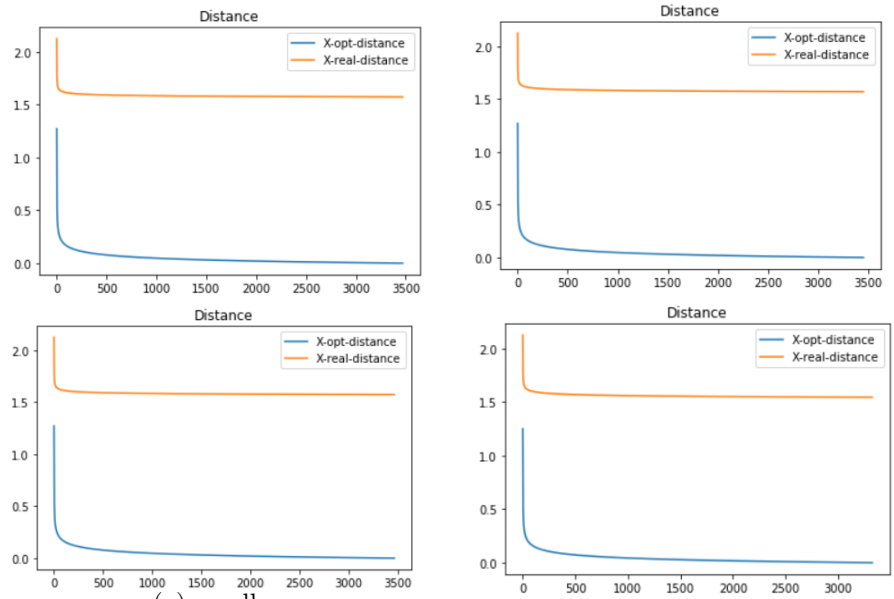
Figure 4: PG with different p

P is the coefficient before the 1-st norm penalty term to make the optimal solution as sparse as possible. Therefore, in theory, if we pick a bigger p, the optimal solution will be closer to the real value. In fact, the experimental results are true. When P=1, PG method can achieve a good effect. The ADMM method has a similar phenomenon while SG method is not sensitive to the change of p-value(as it is shown in the figures below).



(a) small-p (b) big-p

Figure 5: ADMM with different p



(a) small-p (b) big-p

Figure 6: SG with different p

3 Question 2

Use the following algorithm to solve the Logistic Regression problem on the mnist dataset

1.GD
2.SGD

For each algorithm, please provide the distance between the calculation result of each step and the optimal solution and the classification accuracy corresponding to the calculation result of each step on the test set. In addition, discuss the effect of Mini Batch size on the calculation results in stochastic gradient method.

4 Solution 2

I directly use Google colab to avoid set up the environments and there is likely to be some conflicts. You could go to my colab to run the code online:https://colab.research.google.com/drive/1tcisRA4rpZLfwKhIxo1CPwafbL7jsXqT

Essentially, we can view the gradient descent method and the stochastic gradient descent method as the same method. They differ only in the sense that we use the combination of the gradients of all samples or we just use the gradient of a random sample as the optimal direction. This could be done by adjusting the batch size, so we don't need to implement a piece of code for full gradient descent and random gradient descent separately.

4.1 GD

Let θ be the parameters of the neural network, so our goal is to:

$$min f_0(\theta)$$

where f_0 is the loss function used to measure the difference between our forecast and the real value. Assuming we have n samples and all samples are equally important, the above objective function can be written as follows:

$$min \frac{1}{n} \sum_{i=1}^{n} f_i(\theta)$$

So we can easily write out the update steps:

$$\theta^{k+1} = \theta^k - \alpha * \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta^k)$$

In the code implementation, we need to calculate the gradient of each sample's objective function with respect to θ and store it temporarily until all the gradients are calculated and then average them as the overall updated gradient direction.

From the calculation process, we can see that each update of the Batch Gradient Descent Method is a better result for the global (sample).

This update is accurate when the sample size is not too large. But we need to compute all the sample gradients every time and store them, which adds to the computational overhead and storage overhead.

The experimental results are as follows:

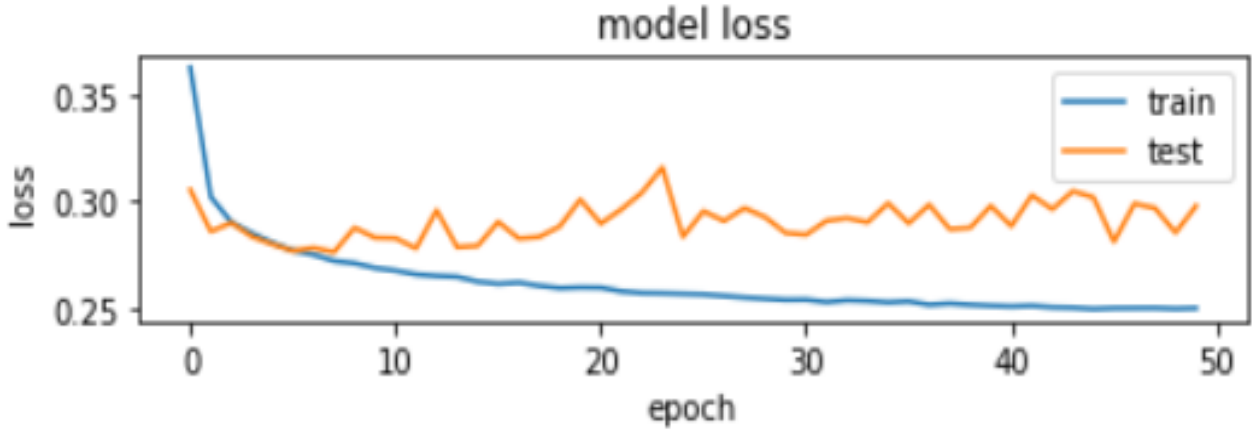


Figure 7: BGD loss

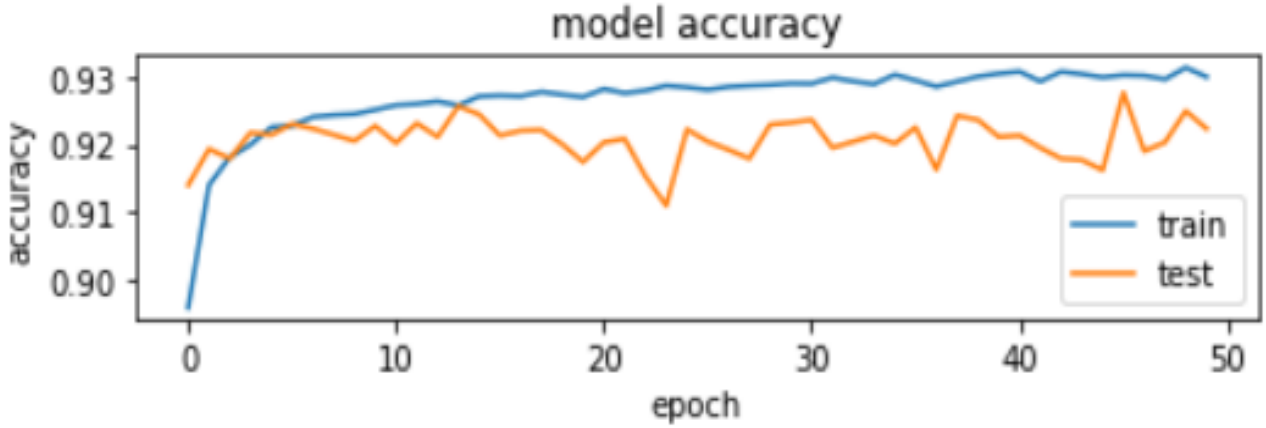


Figure 8: BGD accuracy

4.2 SGD

To avoid having to calculate the gradient of all samples at each update, we choose the stochastic gradient descent method to solve this problem. For each update, we randomly

select the gradient direction of a sample as the update direction, so the update formula becomes as follows:

$$\theta^{k+1} = \theta^k - \alpha^k \nabla f_{i^k}(\theta^k)$$

where $f_{i^k}(\theta^k)$ means the loss function of i_{th} sample at time k. α^k means the step length of time k, which is a decreasing step (usually $\frac{1}{k}$).

The experimental results are as figure 9 and figure 10.

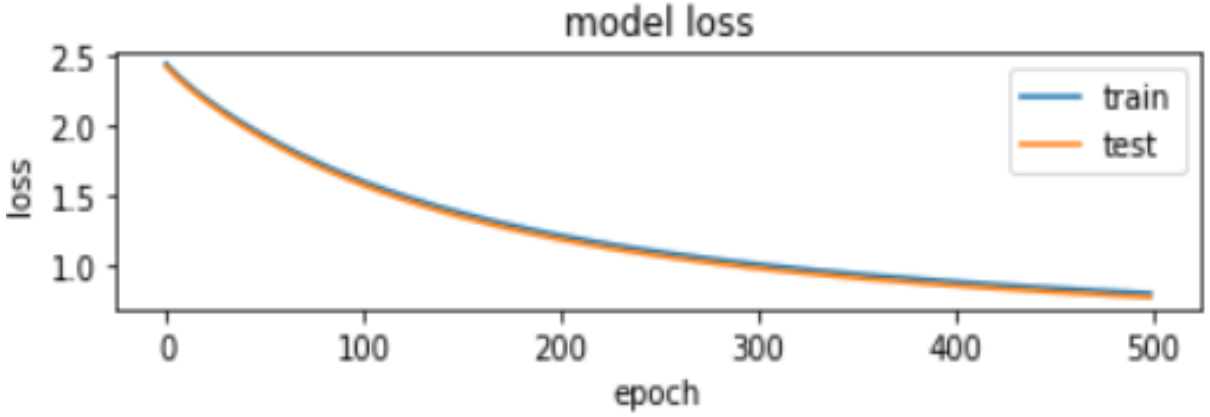


Figure 9: SGD loss

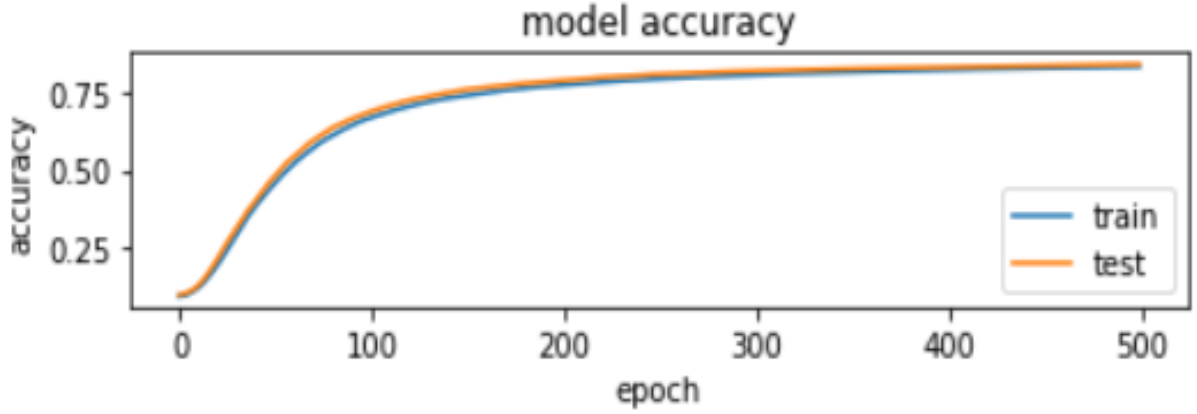
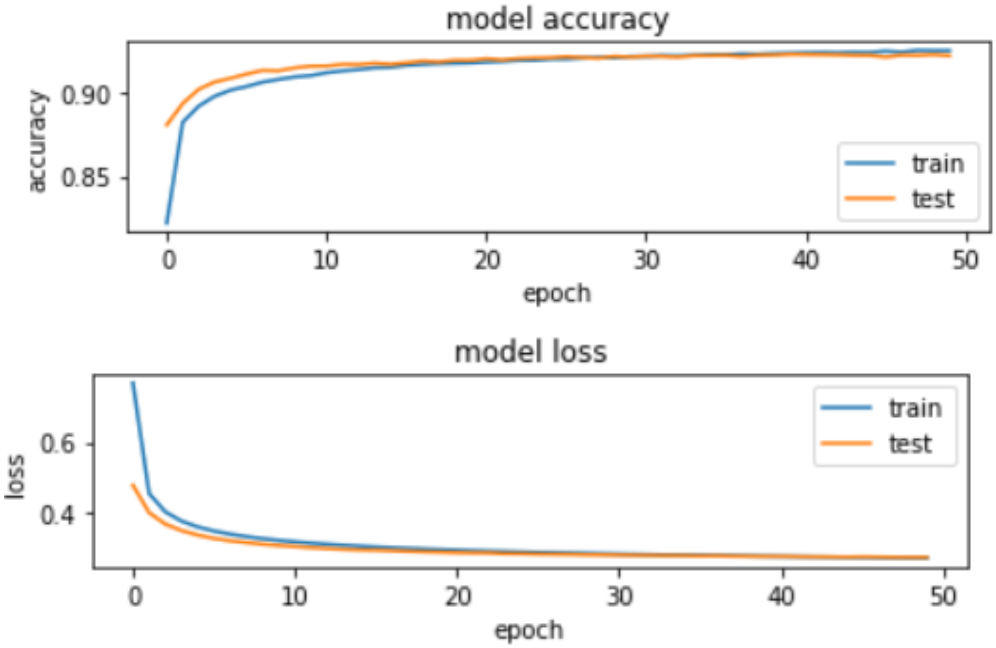


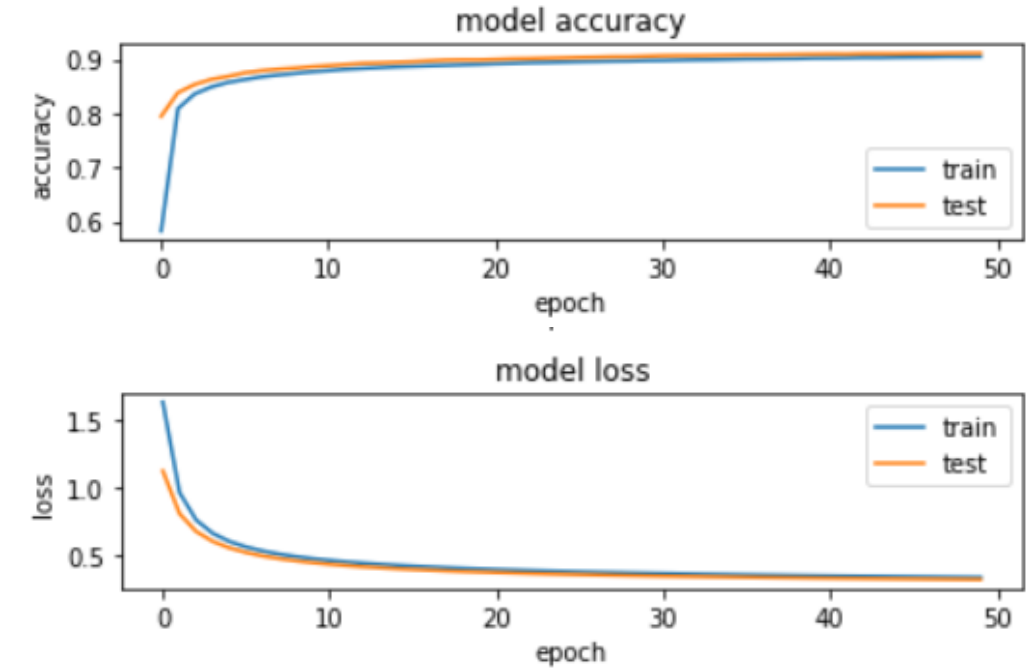
Figure 10: SGD accuracy

4.3 Influence of mini-batch

The above two algorithms can be viewed as the same method with batch= 60,000 and batch=1 (the step length is also different). Therefore, based on the above code implementation, we test the values of several batches from 1 to 60000, and the results are as follows:

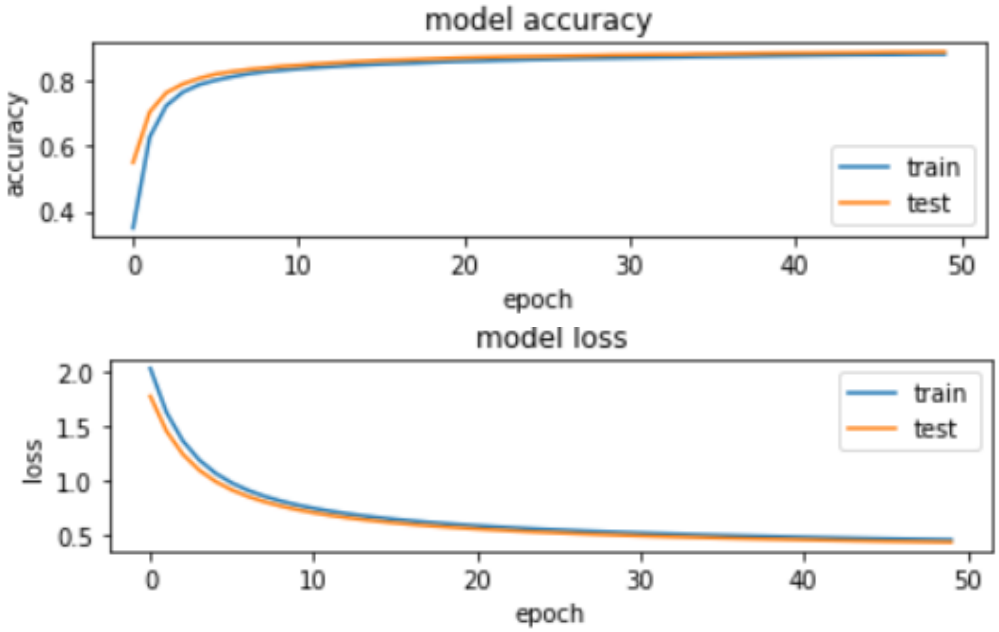


(a) batch = 32



(b) batch = 256

We can find such a phenomenon: when the value of batch_size is very small, the model often achieves a good effect in the first training round ($> 91\%$) and does not improve much in the subsequent training round. But the time cost of each episode is huge, up to 70 seconds.



(c) batch = 1024

When the value of batch_size is large, the training speed is significantly accelerated, and the accuracy of the previous episodes is not as high as that of the batch gradient descent method. More episodes are needed to achieve a high accuracy. We can observe an increase in accuracy in successive beginning episodes.