Cross Validated

How could stochastic gradient descent save time comparing to standard gradient descent?

Asked 3 years, 3 months ago Active 1 year, 1 month ago Viewed 9k times

15

11

Standard Gradient Descent would compute gradient for the entire training dataset.

```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(loss_function, data, params)
  params = params - learning_rate * params_grad
```

For a pre-defined number of epochs, we first compute the gradient vector weights_grad of the loss function for the whole dataset w.r.t. our parameter vector params.

Stochastic Gradient Descent in contrast performs a parameter update for each training example x(i) and label y(i).

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function, example, params)
    params = params - learning_rate * params_grad
```

SGD is said to be much faster. However, I do not understand how it can be much faster if we still have a loop over all data points. Does the computation of the gradient in GD is much slower than computation of GD for each data point separately?

Code comes from here.

machine-learning optimization gradient-descent computational-statistics sgd

edited Oct 22 '18 at 8:45
Ferdi
4,371 5 28 56

asked Aug 27 '16 at 15:25
Alina
695 6 17

1 In the second case you'd take a small batch to approximate the whole data set. This usually works pretty well. So the confusing part is probably that it looks like the number of epochs is the same in both cases, but you would not need as many epochs in case 2. The "hyperparameters" would be different for those two methods: GD nb_epochs != SGD nb_epochs. Let's say for the purpose of the argument: GD nb_epochs = SGD examples * nb_epochs, so that the total number of loops is the same, but the calculation of the gradient is way faster in SGD. – Nima Mousavi Jul 14 '17 at 10:39

This answer on CV is a good and related one. – Zhubarb Feb 14 '18 at 8:55

22

Short answer:

- In many big data setting (say several million data points), calculating **cost or gradient** takes very long time, because we need to sum over all data points.
- We do **NOT need to have exact gradient to reduce the cost** in a given iteration. Some approximation of gradient would work OK.
- Stochastic gradient decent (SGD) approximate the gradient using only one data point. So, evaluating gradient saves a lot of time compared to summing over all data.
- With "reasonable" number of iterations (this number could be couple of thousands, and much less than the number of data points, which may be millions), stochastic gradient decent may get a reasonable good solution.

Long answer:

My notation follows Andrew NG's machine learning Coursera course. If you are not familiar with it, you can review the lecture series here.

Let's assume regression on squared loss, the cost function is

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

and the gradient is

$$\frac{dJ(\theta)}{d\theta} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

for gradient decent (GD), we update the parameter by

$$\theta_{new} = \theta_{old} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

For stochastic gradient decent we get rid of the sum and $1/m$ constant, but get the gradient for current data point $x^{(i)}, y^{(i)}$, where comes time saving.

$$\theta_{new} = \theta_{old} - \alpha \cdot (h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

Here is why we are saving time:

Suppose we have 1 billion data points.

- In GD, in order to update the parameters once, we need to have the (exact) gradient. This requires to sum up these 1 billion data points to perform 1 update.

batch). Therefore, in SGD, we can update the parameters very quickly. In addition, if we "loop" over all data (called one epoch), we actually have 1 billion updates.

The trick is that, in SGD you do not need to have 1 billion iterations/updates, but much less iterations/updates, say 1 million, and you will have "good enough" model to use.

I am writing a code to demo the idea. We first solve the linear system by normal equation, then solve it with SGD. Then we compare the results in terms of parameter values and final objective function values. In order to visualize it later, we will have 2 parameters to tune.

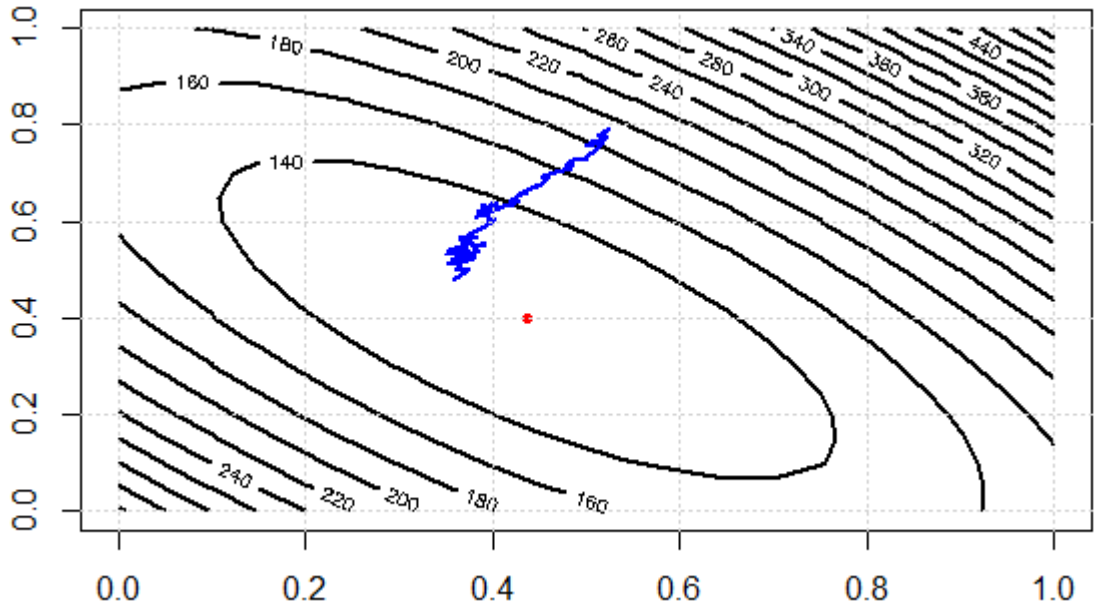
```
set.seed(0);n_data=1e3;n_feature=2;
A=matrix(runif(n_data*n_feature),ncol=n_feature)
b=runif(n_data)
res1=solve(t(A) %*% A, t(A) %*% b)

sq_loss<-function(A,b,x){
  e=A %*% x -b
  v=crossprod(e)
  return(v[1])
}

sq_loss_gr_approx<-function(A,b,x){
  # note, in GD, we need to sum over all data
  # here i is just one random index sample
  i=sample(1:n_data, 1)
  gr=2*(crossprod(A[i,],x)-b[i])*A[i,]
  return(gr)
}

x=runif(n_feature)
alpha=0.01
N_iter=300
loss=rep(0,N_iter)

for (i in 1:N_iter){
  x=x-alpha*sq_loss_gr_approx(A,b,x)
  loss[i]=sq_loss(A,b,x)
}
```

The results:

```
as.vector(res1)
[1] 0.4368427 0.3991028
x
[1] 0.3580121 0.4782659
```

Note, although the parameters are not too close, the loss values are 124.1343 and 123.0355 which are very close.

Here is the cost function values over iterations, we can see it can effectively decrease the loss, which illustrates the idea: we can use a subset of data to approximate the gradient and get "good enough" results.



Now let's check the computational efforts between two approaches. In the experiment, we have 1000 data points, using SD, evaluate gradient once needs to sum over them data. BUT in SGD, `sq_loss_gr_approx` function only sum up 1 data point, and overall we see, the algorithm converges less than 300 iterations (note, not 1000 iterations.) This is the computational savings.

edited Feb 14 '18 at 8:33
F. Tusell
5,889 15 26

answered Aug 27 '16 at 15:38
Haitao Du
24.7k 8 68 167

I thought the argument about "speed" is more about how many operations/iterations are needed to converge

decent - code is different from SDG (and exactly there he uses only a small fraction of the data). Also, in the explanation you provided, although we get rid of sum in SDG, we still compute the update for each data point. I still do not understand how updating a parameter while looping over each data point is faster than just take a sum over all datapoints at once. – Alina Aug 27 '16 at 16:03

@GeoMatt22 In the link I provided it states: "On the other hand, this ultimately complicates convergence to the exact minimum, as SGD will keep overshooting." Meaning it does not converge to better optima. Or did I get it wrong? – Alina Aug 27 '16 at 16:04

@Tonja I am no expert, but for example this highly influential paper in deep learning gives the "faster more reliable training" argument for stochastic gradient descent. Note that it does not use the "raw" version, but uses various curvature estimates to set the (coordinate-dependent) learning rate. – GeoMatt22 Aug 27 '16 at 16:10

1 @Tonja, yes. any "weak" approximation of gradient would work. You can check "gradient boosting", which is similar idea. On the other hand, I am writing some code to demo the idea. I will post it when it is ready. – Haitao Du Aug 27 '16 at 21:22