



Cross Validated is a question and answer site for people interested in statistics, machine learning, data analysis, data mining, and data visualization. Join them; it only takes a minute:

Sign up


Here's how it works:



Anybody can ask a question



Anybody can answer



The best answers are voted up and rise to the top

Tradeoff batch size vs. number of iterations to train a neural network

When training a neural network, what difference does it make to set:

- batch size to *[Math Processing Error]* and number of iterations to *[Math Processing Error]*
- vs. batch size to *[Math Processing Error]* and number of iterations to *[Math Processing Error]*

where $ab = cdab = cd$?


To put it otherwise, assuming that we train the neural network with the same amount of training examples, how to set the optimal batch size and number of iterations? (where batch size * number of iterations = number of training examples shown to the neural network, with the same training example being potentially shown several times)

I am aware that the higher the batch size, the more memory space one needs, and it often makes computations faster. But in terms of performance of the trained network, what difference does it make?


neural-networks

train

edited Mar 3 at 12:04

 Fábio Perez
123 5

asked Aug 5 '15 at 21:19

 Franck Démoncourt
12.9k 6 46 108

2 Answers

From Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. <https://arxiv.org/abs/1609.04836> :

The stochastic gradient descent method and its variants are algorithms of choice for many Deep Learning tasks. These methods operate in a small-batch regime wherein a fraction of the training data, usually 32--512 data points, is sampled to compute an approximation to the gradient. **It has been observed in practice that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize.** There have been some attempts to investigate the cause for this generalization drop in the large-batch regime, however the precise answer for this phenomenon is, hitherto unknown. In this paper, we present ample numerical evidence that supports the view that large-batch methods tend to converge to sharp minimizers of the training and testing functions -- and that sharp minima lead to poorer generalization. In contrast, small-batch methods consistently converge to flat minimizers, and our experiments support a commonly held view that this is due to the inherent noise in the gradient estimation. We also discuss several empirical strategies that help large-batch methods eliminate the generalization gap and conclude with a set of future research ideas and open questions.

[...]

The lack of generalization ability is due to the fact that large-batch methods tend to converge to *sharp minimizers* of the training function. These minimizers are characterized by large positive eigenvalues in $\nabla^2 f(x) \nabla^2 f(x)$ and tend to generalize less well. In contrast, small-batch methods converge to flat minimizers characterized by small positive eigenvalues of $\nabla^2 f(x) \nabla^2 f(x)$. We have observed that the loss function landscape of deep neural networks is such that large-batch methods are almost invariably attracted to regions with sharp minima and that, unlike small batch methods, are unable to escape basins of these minimizers.

[...]

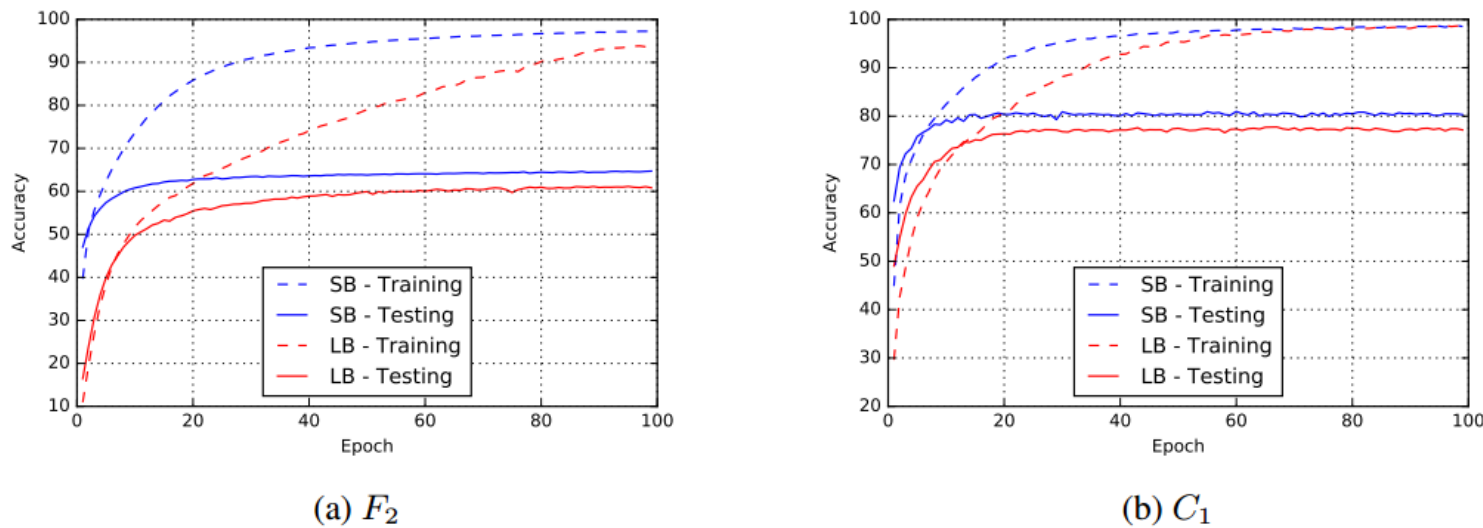


Figure 2: Convergence trajectories of training and testing accuracy for SB and LB methods

Table 2: Performance of small-batch (SB) and large-batch (LB) variants of ADAM on the 6 networks listed in Table 1

Network Name	Training Accuracy		Testing Accuracy	
	SB	LB	SB	LB
F_1	99.66% ± 0.05%	99.92% ± 0.01%	98.03% ± 0.07%	97.81% ± 0.07%
F_2	99.99% ± 0.03%	98.35% ± 2.08%	64.02% ± 0.2%	59.45% ± 1.05%
C_1	99.89% ± 0.02%	99.66% ± 0.2%	80.04% ± 0.12%	77.26% ± 0.42%
C_2	99.99% ± 0.04%	99.99 ± 0.01%	89.24% ± 0.12%	87.26% ± 0.07%
C_3	99.56% ± 0.44%	99.88% ± 0.30%	49.58% ± 0.39%	46.45% ± 0.43%
C_4	99.10% ± 1.23%	99.57% ± 1.84%	63.08% ± 0.5%	57.81% ± 0.17%

Also, some good insights from [Ian Goodfellow](#) answering to [why do not use the whole training set to compute the gradient?](#) on Quora:


The size of the learning rate is limited mostly by factors like how curved the cost function is. You can think of gradient descent as making a linear approximation to the cost function, then moving downhill along that approximate cost. If the cost function is highly non-linear (highly curved) then the approximation will not be very good for very far, so only small step sizes are safe. You can read more about this in Chapter 4 of the deep learning textbook, on numerical computation: <http://www.deeplearningbook.org/contents/numerical.html>

When you put m examples in a minibatch, you need to do O(m) computation and use O(m) memory, but you reduce the amount of uncertainty in the gradient by a factor of only O(sqrt(m)). In other words, there are diminishing marginal returns to putting more examples in the minibatch. You can read more about this in Chapter 8 of the deep learning textbook, on optimization algorithms for deep learning: <http://www.deeplearningbook.org/contents/optimization.html>


Also, if you think about it, even using the entire training set doesn't really give you the true gradient. The true gradient would be the expected gradient with the expectation taken over all possible examples, weighted by the data generating distribution. Using the entire training set is just using a very large minibatch size, where the size of your minibatch is limited by the amount you spend on data collection, rather than the amount you spend on computation.

Related: [Batch gradient descent versus stochastic gradient descent](#)

edited Apr 13 at 12:44

 Community ♦
1

answered Sep 22 '16 at 16:19

 **Franck Démoncourt**
12.9k 6 46 108

I assume you're talking about reducing the batch size in a mini batch stochastic gradient descent algorithm and comparing that to larger batch sizes requiring fewer iterations.

Andrew Ng. provides a good discussion of this and some visuals in his online coursera class on ML and neural networks. So the rest of this post is mostly a regurgitation of his teachings from that class.

Let's take the two extremes, on one side each gradient descent step is using the entire dataset. You're computing the gradients for every sample. In this case you know *exactly* the best directly towards a local minimum. You don't waste time going the wrong direction. So in terms of numbers gradient descent steps, you'll get there in the fewest.

Of course computing the gradient over the entire dataset is expensive. So now we go to the other extreme. A batch size of just 1 sample. In this case the gradient of that sample may take you completely the wrong direction. But hey, the cost of computing the one gradient was quite trivial. As you take steps with regard to just one sample you "wander" around a bit, but on the average you head towards the same local minimum as in full batch gradient descent.

This might be a moment to point out that I have seen some literature suggesting that perhaps this bouncing around that 1-sample stochastic gradient descent does might help you bounce out of a local minima that full batch mode wouldn't avoid, but that's debatable. The general literature suggests that both converge to the same place in most cases.


In terms of computational power, while the single-sample stochastic GD process takes many many more iterations, you end up getting there for less cost than the full batch mode, "typically." This is how Andrew Ng puts it.

Now let's find the middle ground you asked about. We might realize that modern BLAS libraries make computing vector math quite efficient, so computing 10 or 100 samples at once, presuming you've vectorized your code properly, will be barely more work than computing 1 sample (you gain memory call efficiencies as well as computational tricks built into most efficient math libraries). And averaging over a batch of 10, 100, 1000 samples is going to produce a gradient that is a more reasonable approximation of the true, full batch-mode gradient. So our steps are now more accurate, meaning we need fewer of them to converge, and at a cost that is only marginally higher than single-sample GD.



Optimizing the exact size of the mini-batch you should use is generally left to trial and error. Run some tests on a sample of the dataset with numbers ranging from say tens to a few thousand and see which converges fastest, then go with that. Batch sizes in those ranges seem quite common across the literature. And if your data truly is IID, then the central limit theorem on variation of random processes would also suggest that those ranges are a reasonable approximation of the full gradient.

Deciding exactly when to stop iterating is typically done by monitoring your generalization error against an untrained on validation set and choosing the point at which validation error is at its lowest point. Training for too many iterations will eventually lead to overfitting, at which point your error on your validation set will start to climb. When you see this happening back up and stop at the optimal point.

answered Feb 20 '16 at 20:55



David Parks

165  1  12
