Training deep nets with very large SGD batches

Clément Chaffard
Optimization for Machine Learning, EPFL

***Abstract*—In this experiment, we investigate the effect of batch size on training dynamics using stochastic gradient descent on a simple CNN model. We were able to observe how smaller batch sizes training approaches were able to converge faster than larger ones. This put us on the track. If our training method using larger batch size wasn't able to produce good results was because it failed to travel far enough in a reasonable number of training epochs. We slowly increased the learning rate and found an optimal learning rate of 0.1. We were able to recover up to 75% accuracy using full batch training proving to be competitive against small batch size training method. The large batch size training method could prove to be very useful on more complex datasets where smaller batch size approaches fail to converge.**

I. Introduction

We will determine the influence of batch size when training deep neural networks using stochastic gradient descent.

The precise problem that will be investigated is one of classification. Given an two black and white images X and Y of a digit between 0 and 9, the goal is to predict which of the digits is greater.

We will use MNIST dataset and a pytorch environment.

The model we chose is a simple convolutional neural netwok. Unless otherwise stated, the default model was used: 2 convolutional layers (16 and 20 filters with a kernel size of 3 for each), a 2d max pooling with kernel size of 2 and 3 linear layers, generaly following the structure:

$$Input \rightarrow Convolution \rightarrow Activation(ReLu)$$

$$\rightarrow Maxpooling \rightarrow Output$$

Unless otherwise stated we used SGD optimizer, 0.01 learning rate, 0.9 momentum, 80 epochs and we averaged our results on 10 trials. We loaded 2000 samples from MNIST dataset, we normalized it and we split it into 50% training set and 50% testing set.

II. Dataset

We obtained our data from the MNIST dataset. The training and test set contain 1000 pairs each. Each pair is composed of 2 channels (one for each digit of the pair) of 14x14 grayscale pixels. A pair is labelled as 1 if the first digit is smaller or equal to the second, and as 0 otherwise.The training and test set were normalized with respect to the training mean and standard deviation.

III. Stochastic gradient descent

Consider the problem of minimizing an objective function that has the form of a sum:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

f_i is the cost function of the i-th observation, taken from a training set of n observation.

The algorithm comes in the following form:

$$x_{t+1} = x_t - \eta_t \nabla f_i(x_t)$$

$i \in [n]$ is chosen uniformly at random for epochs $t = 0, 1, ...$ and learning rate η_t, that determines the step size at each iteration. At each step we only update the gradient of f_i.

This involves using the current state of the model to make a prediction, comparing the prediction to the expected values, and using the difference as an estimate of the error gradient. This error gradient is then used to update the model weights and the process is repeated.

We used momentum to help accelerate SGD in the relevant direction. It does this by adding a fraction γ of the update vector of the past time step to the current update vector:

$$x_{t+1} = \gamma x_t - \eta_t \nabla f_i(x_t)$$

The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

The more training examples used in the error gradient estimate, the more accurate this estimate will be and

the more likely that the weights of the network will be adjusted in a way that will improve the performance of the model. Alternately, using fewer examples results in a less accurate estimate of the error gradient that is highly dependent on the specific training examples used. Nevertheless, these can result in faster learning and sometimes in a more robust model.

The number of training examples used in the estimate of the error gradient is a hyperparameter for the learning algorithm called the batch size.

A. Mini-batch SGD

Instead of using a single element f_i, we use an average of several of them, thus we have:

$$\widetilde{g_t} = \frac{1}{m} \sum_{j=1}^{m} g_t^j$$

A batch size of 10 means that 10 samples from the training dataset will be used to estimate the error gradient before the model weights are updated. One training epoch means that the learning algorithm has made one pass through the training dataset, where the data has been separated into randomly selected batch size groups.

Small batch size(inferior to data size) properties [1]:

- Advantages: It requires less memory. Since you train the network using fewer samples, the overall training procedure requires less memory. That's especially important if you are not able to fit the whole dataset in your machine's memory. Typically networks train faster with mini-batches. That's because we update the weights after each propagation.
- Disadvantages: The smaller the batch the less accurate the estimate of the gradient can be. Furthermore, the model is not guaranteed to converge to the global optima.

Large batch size properties[1]:

- Advantages: Guaranteed convergence.
- Disadvantages: Training models with large batch size increase the generalization error (see Figure 1). This generalization gap seems to remain even when the models were trained without limits, until the loss function ceased to improve.

IV. Results

A.

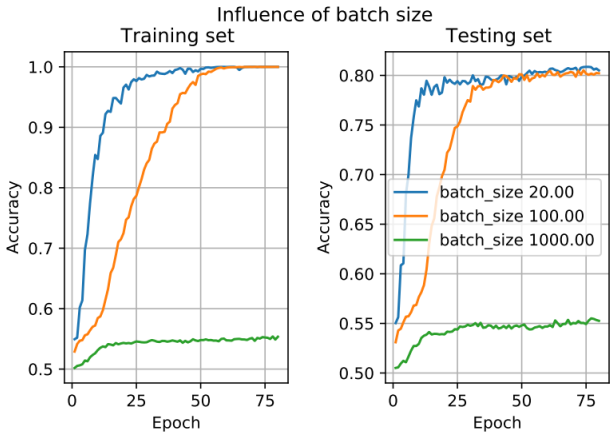

Fig. 1. Batch size influence on accuracy

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. But, we can see on figure 1 that, even using a batch size of 20 over 1000 samples, our model isn't able to generalize properly. We have an accuracy of nearly 1 on the training set for an accuracy of about 80% on the testing set.

We can see that the 75% accuracy on the testing set is achieved at the 10th epoch with a batch size of 20. With a batch size of a 100 this same threshold is achieved later at epoch 25th. The scalability of neural networks refers to its ability to adjust the trade-offs between response time and accuracy. As a result, scalable neural networks can always be particularly useful in budgeted and limited time, which is important for real-world applications. On the other hand our full batch model accuracy doesn't seem to improve over time and is limited to 55% accuracy after 80 epochs.

Based on those results, we make the hypothesis that larger batch sizes don't generalize as well because the model cannot travel far enough in a reasonable number of training epochs.

In the following experiment, we will slowly grow the learning rate to see if we can recover some accuracy using a full batch training for our model.

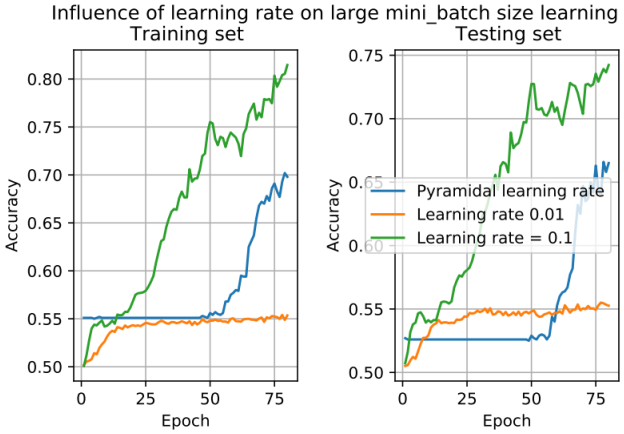

Fig. 2. Learning rate influence on accuracy when training with full batch size

The pyramidal learning was built the following way. We successively trained the model using the respective learning rates $[0.01, 0.05, 0.05, 0.1, 0.1, 0.05, 0.05, 0.01]$ for 10 epochs. We observed that the accuracy started to greatly improve after the 50th epoch, when the learning rate was maximum, set at 0.1. We chose to retrain the model, this time using a learning rate of 0.1 for the full 80 epochs. This new full batch model greatly surpasses the previous ones with an accuracy of 80% on the training set and an accuracy of 75% on the testing set. We can see that it almost reached the performance of the 20 batch size model which had 80% accuracy on the testing set.

V. Conclusion

We investigated the effect of batch size on training deep neural networks using stochastic gradient descent on a simple CNN model. We observed that using batch size of 20 and 100 we had almost perfect accuracy on the testing set but the model wasn't able to generalize properly, achieving only 80% accuracy on the testing set. The smaller batch size was able to converge more rapidly reaching 75% accuracy on the 10th epoch. In a second step, we tried to recover some accuracy on our full batch size training by growing the learning rate, making the hypothesis that the low accuracy was because the model couldn't travel far enough in a reasonable number of training epochs. As a result, we were able to obtain 75% accuracy on the test set of our dataset using full batch learning. On our dataset, the full batch approach wasn't able to over-class the smaller batch sizes. But this could be a promising path for more complex datasets, where small batch approaches fail to converge.

References

[1] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *arXiv preprint arXiv:1711.00489*, 2017.

[2] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," *arXiv preprint arXiv:1705.08741*, 2017.

[3] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.