Stochastic Gradient Descent and Learning Rates

John Abrahams

May 2024

> It is not enough to be industrious; so are the ants. What are you industrious about? - Henry David Thoreau

1 Introduction

This paper was written to summarize the work done under Professor Arielle Carr and Ph.D student Yuesheng Wu at Lehigh University.

We begin with an introduction of stochastic gradient descent, which we will often reference as SGD in this paper. Next we summarize Elastic Averaging SGD, a method for computing stochastic gradient descent in parallel computing environments. We discuss the work done with Yuesheng Wu regarding finding optimal learning rates for the ADAM function, and we end with a practical implementation of SGD with simple least-squares optimization.

2 Introduction to Stochastic Gradient Descent

Stochastic Gradient Descent is an algorithm that is commonplace in almost every machine learning problem. Stochastic Gradient Descent succeeds for two reasons:

1. It is not nearly as expensive as Gradient Descent
2. It generalizes well to unseen test data.

The point of stochastic gradient descent (as you may have already guessed by its name) is to introduce randomness into finding the optimal value. Finding the exact minimum is akin to fitting n data points with an $n-1$ polynomial: we over-fit to our training set and do not perform well on unseen data. SGD introduces the concept of a mini-batch, which allows us to compute the gradient with respect to only some observations in our dataset. This solves both of our problems: it allows us to be computationally efficient and generalize well to unseen test data.

3 Elastic Averaging with Stochastic Gradient Descent

Stochastic gradient descent is incredibly powerful. Despite its computational efficiency with respect to gradient descent, it still remains expensive. Elastic Averaging is a method that works to provide parallelism for the SGD algorithm and solve this bottleneck. The term "Elastic" is derived from the Greek word "elastikos" meaning "propulsive." This is a nice way to sum up exactly what the algorithm is doing: Worker threads drive the master position forward without venturing too far away. There are two key-ideas to this algorithm:

1: Workers maintain their local parameters.

2: We do not let local parameters go far away from the central parameter.

We begin with the following problem statement:

$$\min_x F(x) := \mathbf{E}[f(x, \theta)]$$

Note that $F(x) = \int_\Omega f(x, \theta)\mathbf{P}(d\theta)$, where x is the model parameter to be estimated. This is all to say that $f(x, \theta)$ is an unbiased estimator of $f(x)$. This allows us to sample from the dataset according to probability distribution $\mathbf{P}$. The optimization problem above becomes:

$$\sum_{i=1}^{p}[f_i(w^i) - \frac{\rho}{2}||w^i - \tilde{w}||^2]$$

where $p \in \mathbf{N}$ is the number of workers. This is our objective function. The latter term, $\frac{\rho}{2}||w^i - \tilde{w}||^2$, does two things: it makes sure the workers do not go too far away from the central parameter (as explained above), and it provides the elastic force to move our central variable forward. The ρ allows us to modify the tradeoff between exploration and exploitation. For example, smaller values allow for more exploration: it allows the x_i to fluctuate further from the center variable $\tilde{x}$.

We take the gradient descent step with respect to x^i and $\tilde{x}$:

$$x_{t+1}^i = x_t^i + \eta(g_t^i(x_t^i) + \rho(x_t^i - \tilde{x}_t))$$

$$\tilde{x}_{t+1} = \tilde{x}_t + \eta \sum_{i=1}^{p} \rho(x_t^i - \tilde{x}_t)$$

Where η is the learning rate and $g_t^i(x_t^i)$ denotes the stochastic gradient descent of $F(x)$ at iteration t for worker i. Notice the elastic force is defined as $\eta\rho(x^i - \tilde{x})$ for each worker, which then updates the center variable. The elastic force brings the workers back after moving forward from their stochastic gradient descent calculations. After a fixed time period, the value of the master updates with respect to the elastic force, moving the master closer to a local minimum in a stable manner. The algorithm is here below:

Algorithm 1: Asynchronous EASGD: Processing by worker i and the master

Input : learning rate η, moving rate α, communication period $\tau \in \mathbf{N}$
Initialize: $\tilde{x}$ is initialized randomly, $x_i = x$, $t_i = 0$

repeat
  $x \leftarrow x_i$;
  if (τ *divides* t_i) **then**
    $x_i \leftarrow x_i - \alpha(x - \tilde{x})$;
    $\tilde{x} \leftarrow \tilde{x} + \alpha(x - \tilde{x})$;
  $x_i \leftarrow x_i - \eta g_i(x)$;
  $t_i \leftarrow t_i + 1$;
until *forever*;

Notice what the algorithm is doing: We continue with regular stochastic gradient descent. Then, at fixed intervals, we move the workers back towards the master position and move the master position down according to the worker's elastic force.

4 Choosing Optimal Learning Rates

Learning rates are not trivial! They are incredibly important in developing effective gradient descent algorithms. For the ADAM algorithm, we initially used the following learning rates: $\{0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5\}$ We then incremented/decremented around them with the following code:

```
def continuous_alpha(alpha):
  start = alpha - .5*alpha
  stop =  alpha + .5*alpha
  step = .1 * alpha
  alphas = np.arange(start, stop, step)
  return alphas
```

We optimized based on how quickly we converged and our overall training loss during the 20 epochs. For the ADAM optimizer, we found the learning rate $\alpha = 0.0781$ provided the fastest convergence and best training loss.

5 Random Kaczmarz and Least Squares Optimization

In this section, we summarize the stochastic gradient descent algorithm used for computing $Ax = b$. This algorithm is called Randomized Kaczmarz. We randomly sample rows in the matrix A, and continually calculate the gradient. This is an incredibly intuitive algorithm that helps in understanding what is really going on behind the scenes. For example, one may ask, why is this considered a stochastic gradient descent algorithm? We introduce that below:

Gradient descent for least squares takes on the following form for minimizing the objective (loss) function:

$$F(x) = \frac{1}{2}||Ax - b||^2 \equiv \frac{1}{2}\sum_i (a_i^T x - b_i)^2$$

Regular Gradient Descent requires we take the derivative with respect to x:

$$\frac{\partial f}{\partial x} = \sum_i a_i(a_i^T x - b_i) \equiv \sum_i a_i a_i^T x - a_i b_i$$

As a quick sanity check, we make sure our dimensions are correct for the matrix multiplications we are about to do:

$$\begin{aligned} a_i &: n \times 1 \\ a_i^T &: 1 \times n \\ b_i &: \text{scalar} \\ x &: n \times 1 \end{aligned}$$

We also notice our sum is equivalent to the following:

$$a_1 a_1^T x - (b_1)a_1 + a_2 a_2^T x - (b_2)a_2 + \cdots + a_n a_n^T x - (b_n)a_n$$

Grouping like terms:

$$a_1 a_1^T x + a_2 a_2^T x + \cdots + a_n a_n^T x - ((b_1)a_1 + (b_2)a_2 + (b_n)a_n)$$

In matrix form:

$$A^T Ax - A^T b$$

We find that the normal equations exactly satisfy our loss function:

$$\frac{\partial f(x)}{\partial x} = A^T Ax - A^T b$$

How can we get stochastic gradient descent out of this? Stochastic gradient descent tells us it is much more advantageous computationally to compute one random $\frac{\partial f(x)}{\partial x_i}$ on each step. This corresponds to one row in the matrix, or one observation in the dataset. We modify the algorithm slightly:

$$\frac{\partial f(x)}{\partial x_i} = a_i a_i^T x - a_i b_i$$

Our gradient descent equation is:

$$x_{k+1} = x_k - s\nabla f(x_k) = x_k - s(a_i a_i^T x_k - a_i b_i) = x_k + s(a_i b_i - a_i a_i^T x_k) = x_k + s(b_i - a_i^T x_k)a_i$$

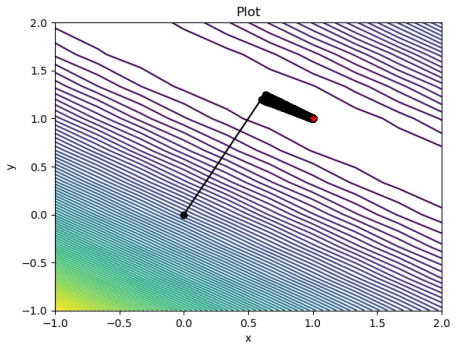

Figure 1: Convergence with Random Kaczmarz on 2x2 matrix

We choose $s = 1/a^T a$:

$$= x_k + (1/a^T a)(b_i - a_i^T x_k)a_i$$

We test the Kaczmarz iterations for two datasets, the former of small scale and the latter of a much larger scale, with 1000 and 100 iterations, respectively, using Julia. The code used is below:

```
function ord_stoch_grad_des(xk, b, a_i, a_i_t)
    term = ((b - sum(a_i_t .* xk))/sum(a_i_t .* a_i_t)) * a_i
    return xk + term

end
```

For the following equations:

$$x + 2y = 3$$
$$2x + 3y = 5$$

We find that the solution converges to [1 1], as in Figure 1. Note the red star indicates this value.

Now for a more impressive example. We develop a linear regression equation using random data. The matrix is of size 100x2. The red star indicates the the "best vector" found with the projection matrix: $(A^T A)^{-1} A^T b$

Using the code above, we find in figure 2 that it does indeed converge.

Notice the difference of stochastic gradient descent for these two figures; with an exact solution, we find the algorithm does converge but only after 10000 iterations. With a solution that is not exact however, we find we continue moving around the minimum, touching it at times but never absolutely converging. Notice also we take an incredibly large step originally, followed by small oscillatory steps toward the minimum in both cases. In practice, it is best to stop early so as to avoid over-fitting for the larger case. However for the smaller case, since there exists an exact solution there is no need to worry about over-fitting. Therefore with SGD we get the best of both worlds: we are able to generalize to unseen data when desired and find exact solutions when desired.

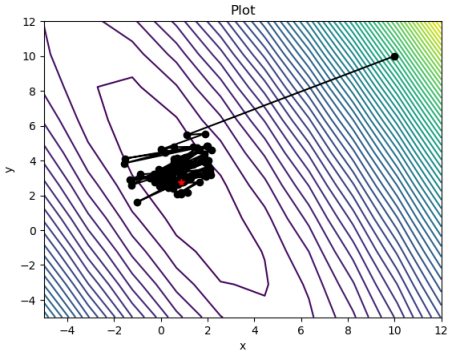

Figure 2: Convergence with Random Kaczmarz on 100x2 matrix

6 Conclusion

This independent study could not have been done without the lectures posted on MIT OpenCourseware for 18.065 and the corresponding textbook "Learning from Data." Moreover, problem sets provided by Professor Steven G. Johnson on Github for this course proved very helpful. Professor Arielle Carr and Yuesheng Wu were very helpful as well.

Stochastic Gradient Descent is one of the most widely used optimization algorithms in deep learning applications. Its ability to perform on unseen test data remains unmatched. Moreover, EASGD provides an interesting application with respect to providing parallelism to the algorithm. Random Kaczmarz provides an interesting and more concrete application of SGD. It would be interesting (potentially in future work), to find out what EASGD looks like in terms of least squares.