IE 598: Big Data Optimization **Fall 2016**

Lecture 24: Summary and Outlook – November 15

Lecturer: Niao He *Scribers: Joseph Lubars*

24.1 Case Study in Large-Scale Logistic Regression

24.1.1 Logistic Regression Model

In lecture 13, we looked at the performance of various algorithms on logistic regression. Given data $(x_1, y_1), \dots, (x_N, y_N)$, where $x_i \in R^d, y_i \in \{-1, 1\}$, $i = 1, \dots, N$, logistic regression arises from applying maximum likelihood estimation to the following likelihood function:

$$P(y = 1|x, w) = \sigma(w^T x) := \frac{1}{1 + \exp(-w^T x)}. \tag{24.1}$$

Applying a regularization term with λ as the regularization parameter, we get the following optimization problem:

$$\min_w f(w) := \frac{1}{N}\sum_{i=1}^{N} \log(1 + \exp(-y_i w^T x_i)) + \frac{\lambda}{2}\|w\|_2^2 \tag{24.2}$$

Note that this is a smooth, strongly convex function. In practice, N will be very large, on the order of thousands or more. With such a large N, the first part of the above objective resembles an expectation, allowing us to use stochastic algorithms. Furthermore, given the structure of the problem as a finite sum of convex functions, we are in a position to use the incremental gradient methods from last lecture.

24.1.2 Algorithm Comparison

We will compare three algorithms for solving the optimization problem: Gradient Descent (GD), Stochastic Gradient Descent (SGD), and Stochastic Variance Reduced Gradient (SVRG). Their relative performance can be summed up in the following table:

Algorithm	Iteration Complexity
GD	$N\frac{L}{\mu}\log(\frac{1}{\epsilon})$
SGD	$\frac{L}{\mu^2}\cdot\frac{1}{\epsilon}$
SVRG	$(N + \frac{L}{\mu})\log(\frac{1}{\epsilon})$

Notations:

1. μ is the strong convexity parameter
2. L is the constant for L-smoothness

24.1.3 Data Generation

We will do first do experiments using a synthetic dataset. In this dataset, we will use a relatively small N for large-scale problems, with $N = 10000$. We pick $\lambda = 10^{-4}$. This choice of an extremely small λ is intentional, so the regularization term does not dominate. In practice, a choice of $\lambda = 1/N$ is a good default choice and is what is used here. x_i is generated according to a standard normal random variable, normalized to norm 1. y_i is generated according to the logistic model (Equation 24.1). Because L scales as $\lambda + \|x_i\|_2^2/4$, we therefore set $L = \lambda + 1/4$.

Given these parameters, both N and L/μ are on the order of 10^4. Then, the coefficient of the $1/\epsilon$ term for complexity is on the order of 10^8 for both Gradient Descent and Stochastic Gradient Descent. However, for SVRG, $(N + L/\mu)$ is only on the order of 10^4. Because of this, we expect SVRG to have significantly better performance than the other two algorithms.

```
N = 10000
dim = 50
lamda = 1e-4
np.random.seed(1)
w = np.matrix(np.random.multivariate_normal([0.0]*dim, np.eye(dim))).T
X = np.matrix(np.random.multivariate_normal([0.0]*dim, np.eye(dim), size = N))
X = np.matrix(normalize(X, axis=1, norm='l2'))
y = 2 * (np.random.uniform(size = (N, 1)) < sigmoid(X*w)) - 1
```

Listing 1: Python Code for Data Generation

24.1.4 Optimization via CVXPY

First, we use CVXPY to obtain an optimal solution using the interior point method. We set the tolerance to 10^{-15} to allow us to compare our objective function value more accurately. As we will see, SVRG is able to give us similar accuracy in a fraction of the time.

```
import cvxpy as cvx
w = cvx.Variable(dim)
loss = 1.0/N * cvx.sum_entries(cvx.logistic(-cvx.mul_elemwise(y, X*w))) + lamda/2 * cvx.
    sum_squares(w)

problem = cvx.Problem(cvx.Minimize(loss))
problem.solve(verbose=True, abstol=1e-15)
opt = problem.value
print('Optimal Objective function value is: {}'.format(opt))
```

Listing 2: Python Code for CVXPY Optimization

In the end, we get an optimal objective value of 0.605366383115 after running for over 12 seconds.

24.1.5 Gradient Descent

We set the constant L as described above. We also fix the number of passes over the data for fair comparison between algorithms. Gradient descent gives us an objective value of 0.609, within 0.004 of the optimal value obtained from CVXPY above.

```
L = lambda + 1.0/4
num_pass = 50

## Define the objective and gradient oracles.
```

```
def obj(w):
    return 1.0/N * np.sum( np.log(1 + np.exp(-np.multiply(y, (X*w)))) ) + 1.0/2 * lamda * (w.T*w)

def grad(w,X,y):
    return 1.0/(X.shape[0]) * X.T * np.multiply( y, sigmoid(np.multiply(y, X*w)) - 1) + lamda*w

## Gradient Descent
w = np.matrix([0.0]*dim).T
obj_GD = []
max_iter = num_pass
for t in range(0, max_iter):
    obj_val = obj(w)
    w = w - 2.0/(L+lamda) * grad(w, X, y)
    obj_GD.append(obj_val.item())

print('Objective function value is: {}'.format(obj_GD[-1]))
```

Listing 3: Python Code for Gradient Descent

24.1.6 Stochastic Gradient Descent

Now, we use SGD. A batch size of 50 is used, improving the algorithm by averaging a number of data points together at each step. The total number of steps is adjusted so that the number of passes is the same as used for full gradient descent. Notably, the theoretical stepsize is not used here. Because $1/\lambda = 10000$ for this example, the theoretical stepsize would start out at 10000, which is much too large for convergence. This effect is mitigated in practice both by using $1/(t + t_0)$ to start with a smaller time-varying factor, and by adding a constant of 0.1 in front to further reduce the initial stepsize. A comparison of the performance with the two step sizes can be found in Figure 24.1. Because SGD is a stochastic algorithm, the objective does not decrease monotonically. The final objective value also varies between runs.

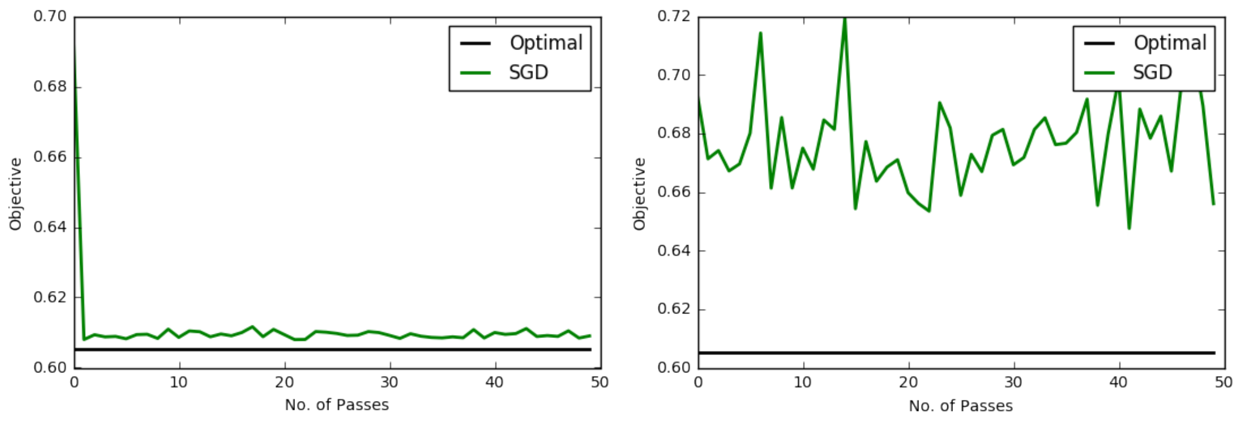
```
## Stochastic Gradient Descent
w = np.matrix([0.0]*dim).T
obj_SGD = []
batch = 50
for s in range(num_pass):
    obj_val = obj(w)
    obj_SGD.append(obj_val.item())
    max_iter = int(N/batch)
    for t in range(max_iter):
        rand_idx = np.random.randint(0, N-1,batch)
        yt = y[rand_idx, 0]
        xt = X[rand_idx, :]
        # gamma = 1/(lamda*(t+1))              # theoretical stepsize
        gamma = 0.1/(lamda*(t+100))            # better stepsize
        w = w -  gamma * grad(w,xt,yt)

print('Objective function value is: {}'.format(obj_SGD[-1]))
```

Listing 4: Python Code for Stochastic Gradient Descent

24.1.7 Stochastic Variance Reduced Gradient

In order to keep the number of passes through the data consistent, we set the number of epochs to 15. For each epoch, we pass through the data three times, so the total number of passes through the data is close to 50. we keep the minibatch size at 50 from SGD.



(a) SGD with better stepsize

(b) SGD with theoretical stepsize

Figure 24.1: A comparison of Stepsizes for SGD. With the theoretical stepsize, it never seems to converge.

```python
w = np.matrix([0.0]*dim).T
obj_SVRG = []
passes_SVRG = []

Epochs = 15
k = 2
batch = 50
for s in range(Epochs):
    obj_val = obj(w)
    obj_SVRG.append(obj_val.item())
    passes_SVRG.append(s*k+s)

    w_prev = w
    gradient = grad(w, X, y)

    obj_SVRG.append(obj_val.item())
    passes_SVRG.append(s*k+s+1)

    max_iter = int(k*N/batch)
    for t in range(max_iter):
        rand_idx = np.random.randint(0, N-1, batch)
        yt = y[rand_idx, 0]
        xt = X[rand_idx, :]
        gamma = 1/L
        w = w - gamma * (grad(w,xt,yt) - grad(w_prev,xt,yt) + gradient)

print('Objective function value is: {}'.format(obj_SVRG[-1]))
```

Listing 5: Python Code for Stochastic Variance Reduced Gradient

The final objective value is even lower than the optimal value calculated from the interior point method, indicating that SVRG is converging to the optimal solution extremely quickly. In fact, this value is reached after just 6 epochs.

24.1.8 Comparing Algorithm Performance on Synthetic Data

Now that we have run all three algorithms (GD, SGD, SVRG) on our synthetic dataset, it is time to compare their performance. The relative objective error of the three algorithms can be found in Figure 24.2. Note that SVRG is tremendously better than the other two algorithms, achieving an error of 10^{-10} in just 12 passes through the data. The jumps in the graph are due to the fact that the solution is not updated with every pass through the data with SVRG. SGD is doing better than GD, but still does not approach the performance of SVRG.

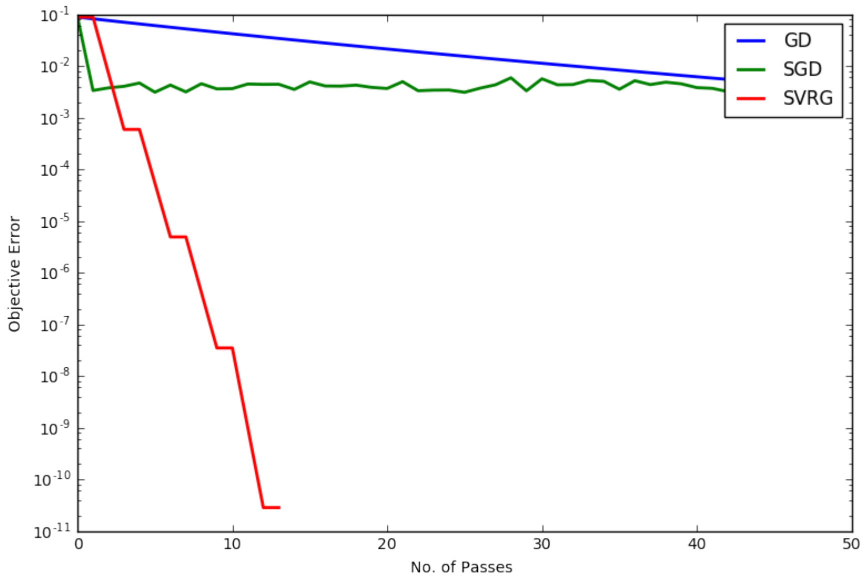


Figure 24.2: Comparison of GD, SGD, and SVRG on a synthetic dataset

24.1.9 Comparing Algorithm Performance on Real Data

Now, we use the Covertype dataset, where the forest cover type is predicted from cartographic information. This dataset is very popular in the machine learning community. In this test case, we are using $N = 58101$ and the dimension of the data is 54. The relative performance of the algorithms can be found in Figure 24.3. As you can see, SGD continues to significantly outperform normal gradient descent. SVRG beats the benchmark set by interior point method after just 12 passes through the data.

24.2 Course Summary and Outlook

In big data and machine learning, there are many different tasks, including:

1. Image/Signal Processing
2. Supervised Learning
3. Unsupervised Learning

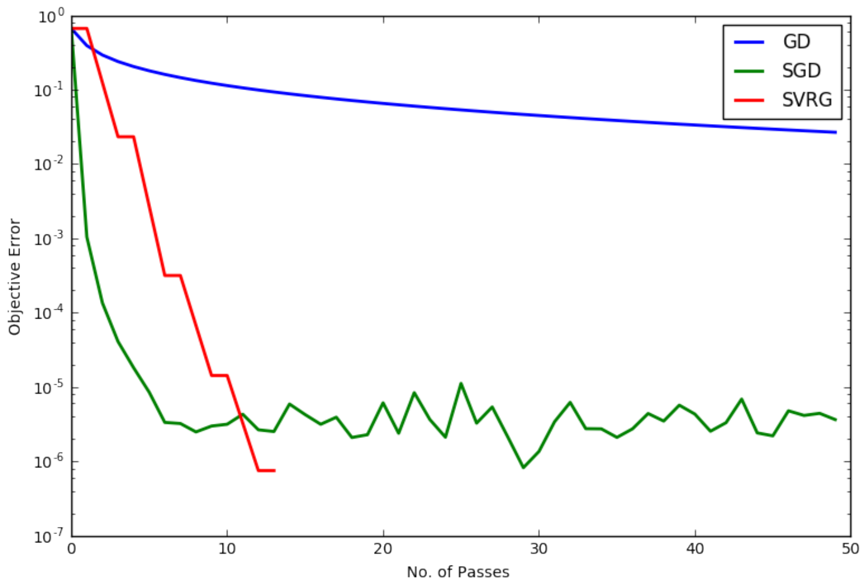


Figure 24.3: Comparison of GD, SGD, and SVRG on the Covertype dataset

4. Reinforcement Learning
5. Recommendation Systems

However, in the end, at the heart of all of these tasks is optimization. In this course, we have explored optimization theories, algorithms, and big data applications, gaining the tools to tackle these tasks. Along the way, we have learned how to use the structure of various problems to design optimization algorithms, and studied the complexity of algorithms required to solve these various classes of problems. With the tools from this class, we can identify which algorithm to use for any given problem.

What follows is a summary of all the topics that have been covered:

24.2.1 Summary of Topics Covered

The primary focus of this class has been convex optimization problems, because these are the problems that we can solve. Critically, any local optimum for a convex problem is also a global optimum, which is crucial in the operation of the algorithms that we have studied.

24.2.1.1 Convex Optimization Fundamentals

We learned the fundamentals of convex optimization, including definitions and characterizations of convex sets and convexity, operations that preserver convexity, and optimality conditions. We also learned about Lagrangian duality, saddle points, and KKT conditions. For more fundamentals, consider taking IE 521.

Conic optimization is a family that already covers most relevant convex problems. We learned three types of conic problems: LP, SOCP, and SDP. We also learned our first polynomial-time algorithm for solving a convex problem: the Interior Point Method.

24.2.1.2 Convex Optimization Algorithms

We learned a large number of algorithms and their iteration complexity, along with the situations where they are useful. In general, you should consider the structure of the problem and choose the algorithm that best utilizes this structure. For example, in the case study above, SVRG best utilizes the finite sum structure of the problem, and therefore outperforms SGD. Broken into classes, with a few notes, here is a rough list. See the lecture slides for the iteration complexities/costs:

- Smooth Convex Optimization
  - No restrictions: (Accelerated) Gradient Descent
  - Need to project: Projected Gradient Descent
  - Hard to calculate the projection: Frank-Wolfe
  - High-Dimensional: Block Coordinate Gradient Descent (BCGD)
- Nonsmooth Convex Optimization
  - Subgradient Descent
  - Mirror Descent (More general than Subgradient Descent)
  - When proximal operators are easy to compute: (Accelerated) Proximal Point Algorithm
- Nonsmooth Convex Optimization, in the form $f(x) = max_{y \in Y}\{\langle Ax + b, y\rangle - \phi(y)\}$
  - Often, problems can be represented as the maximum of convex functions, allowing you to use these algorithms:
  - Nesterov's Smoothing + (Accelerated) Gradient Descent
  - Mirror Prox: Good with an easy prox-mapping
- Nonsmooth Convex Optimization, in the form $\min_x f(x) + g(x)$ (e.g. LASSO)
  - (Accelerated) Proximal Gradient
  - Douglas-Rachford Splitting: Good when you can efficiently compute proximal operator for both f and g
  - Krasnosel'skii-Mann (KM): A generalization of fixed-point algorithms, splitting algorithms
- Stochastic Convex Optimization
  - Sample Average Approximation (SAA)
  - When f is strongly convex: Stochastic Approximation (SA)
  - For general convex functions: Mirror Descent SA
- Finite Sum of Convex Functions
  - When gradient is easy to calculate: Gradient Descent
  - For large N: Stochastic Gradient Descent
  - To utilize the sum structure: SVRG/S2GD

24.2.1.3 Topics Not Covered

We have covered a huge number of first-order algorithms, but there are still many topics that we did not cover. Within convex optimization, these topics include:

1. Nonsmooth First-Order Optimization Algorithms: There are still several algorithms that we did not cover, including bundle methods and primal-dual methods.
2. Stochastic Optimization Algorithms: The algorithms we have discussed have variations in stochastic situations (e.g. Stochastic Frank Wolfe)
3. Second-Order Methods: We have all learned Newton's Method, but there are other second-order methods with superlinear convergence rates.
4. Zero-Order Methods: Sometimes, even the gradient is too expensive to calculate. In these situations, it is necessary to use a zero-order method.
5. Distributed Algorithms: Many algorithms that we have learned can be made parallel or distributed. One example is Hogwild!, an parallel SGD algorithm that does not use locking.

There are also other related convex problems, where the goal is not simply minimizing an objective. These include saddle point problems and online convex optimization, and have their own algorithms.

Finally, beyond convex optimization, many problems are non-convex, but still important to solve. Sometimes, it is possible to convexify the problem or solve a convex relaxation. However, many important problems such as deep learning, clustering, graphical models, and various combinatorial optimization problems are non-convex optimization problems, for which various non-convex algorithms (and even convex algorithms like Gradient Descent) are used to solve. This creates a number of problems, including escaping from saddle points. There are ways of dealing with each of these problems, but it is an active research area to discover when it is possible to converge to a global optimum solution.