Recap: Proximal Gradient Descent

Proximal gradient descent operates on problems of the form

$$\min_x g(x) + h(x),$$

where g is convex, smooth, and h is convex and "simple" (proximal operator is explicitly calculable).

Choose initial $x^{(0)}$ and repeat for $k = 1, 2, 3, \ldots$

$$x^{(k)} = \text{prox}_{t_k}\left(x^{(k-1)} - t_k \nabla g(x^{(k-1)})\right)$$

where

$$\text{prox}_t(x) = \underset{z}{\text{argmin}} \frac{1}{2t}||x - z||_2^2 + h(z).$$

Recall the prox operator tries to find a point that makes h small but is approximately close to x. We've seen it can be explicitly characterized for norms and other functions h. Now the difficulty of iterations is in computing h.

Difficulty of iterations is in applying prox, which only depends on h, assuming ∇g is computable. Proximal gradient descent enjoys the same convergence rate as its fully smooth version, hence useful when the prox is efficient.

The **motivation** for proximal gradient descent is to iteratively minimize a quadratic expansion in g, plus the original h.

$$\begin{aligned} x^+ &= \underset{z}{\text{argmin}} \frac{1}{2t}||x - t\nabla g(x) - z||_2^2 + h(z) \\ &= \underset{z}{\text{argmin}} \nabla g(x)^T(z - x) + \frac{1}{2t}||z - x||_2^2 + h(z) \end{aligned}$$

The quadratic approximation here uses Hessian equal to (a scaled version of) the identity $\frac{1}{t}I$.

A fundamental difference between gradient descent and Newton's method is that the latter also iteratively minimized quadratic approximations, but these used the local Hessian of the function in question.

Proximal Newton Method

What happens if $\frac{1}{t}I$ is replaced with $\nabla^2 g(x)$? Leads us to the proximal Newton method.

Define a **scaled proximal mapping**

$$\text{prox}_H(x) = \underset{z}{\text{argmin}} \frac{1}{2}||x-z||_H^2 + h(z)$$

where $||x||_H^2 = x^T H x$ defines a norm, given a matrix $H \succ 0$. With $H = \frac{1}{t}I$, we get the gradient descent definition.

Starting with $x^{(0)}$, repeat for $k = 1, 2, 3, \ldots$

Apply scaled prox operator to a Newton step:

$$y^{(k)} = \text{prox}_{H_{k-1}}\left(x^{(k-1)} - H_{k-1}^{-1}\nabla g(x^{(k-1)})\right)$$

Treat $y^{(k)}$ as a direction to move in:

$$x^{(k)} = x^{(k-1)} + t_k(y^{(k)} - x^{(k-1)})$$

Here $H_{k-1} = \nabla^2 g(x^{(k-1)})$ and t_k is a step size chosen from backtracking line search.

Back to the motivation, check this indeed minimizes a quadratic approximation of g, plus h.

$$\begin{aligned} y &= \underset{z}{\text{argmin}} \frac{1}{2}||x - H^{-1}\nabla g(x) - z||_H^2 + h(z) \\ &= \underset{z}{\text{argmin}} \nabla g(x)^T(z-x) + \frac{1}{2}(z-x)^T H(z-x) + h(z) \end{aligned}$$

Notes:

- When $h(z) = 0$, we get the usual Newton update. Just like proximal gradient gives the gradient descent update.
- For $H \prec 0$, check that $\text{prox}_H(\cdot)$ retains many of the nice properties of (unscaled) proximal mappings (Lee et al., 2012 [4]). E.g. it is well-defined since the minimizer is unique.
- Difficulty of prox has mostly to do with h, however, now the Hessian of g also plays a role. The structure of this Hessian H can make a difference.

  If g has a diagonal Hessian, computing the prox is much easier.

Backtracking Line Search

Not obvious how to apply backtracking to proximal Newton.

As with Newton's method in fully smooth problems, pure step sizes $t_k = 1$, $k = 1, 2, 3, \ldots$ need not converge. We need to apply backtracking line search with parameters $0 < \alpha \leq 1/2$, $0 < \beta < 1$. Let $v = \text{prox}_H(x - H^{-1}\nabla g(x)) - x$ be the proximal Newton direction at a given iteration. Start with $t = 1$, and while

$$f(x + tv) > f(x) + \alpha t \nabla g(x)^T v + \alpha(h(x + td) - h(x))$$

shrink $t = \beta t$. (Here $f = g + h$).

This scheme is a different spirit than studied in proximal gradient descent as it importantly avoids recomputing the prox at each inner backtracking iteration. In proximal gradient descent, every inner iteration requires recomputing the prox. Important here since the prox is more expensive.

Wait... does this even make sense?

One of the main drivers behind proximal gradient descent is that we can transform the problem

$$\min_x g(x) + h(x)$$

into a sequence of problems where $g(x)$ is essentially replaced by $||b - x||_2^2$. This can be easy, but it depends on h.

Now we have transformed into a sequence of problems where $g(x)$ is essentially replaced by $b^T x + x^T A x$. For dense A, this seems like it would rarely be easy, regardless of h. Evaluating the scaled prox

$$\operatorname*{argmin}_z \nabla g(x)^T (z - x) + \frac{1}{2}(z - x)^T H (z - x) + h(z)$$

seems to be not an easy subproblem, for a generic Hessian H.

All this is true, and the prox operator in proximal Newton is usually extremely expensive, and one that we solve with an optimization subroutine. For example, the prox of $h(x) = ||x||_1$ is the standard lasso problem. Don't want to solve this as an inner problem.

What we hope is that the convergence rate of prox Newton, in terms of the number of iterations (prox evaluations) needed, is like the usual Newton method, which ends up being true.

Therefore, if we have a decent inner solver for the prox step, it can be quite efficient to use proximal Newton (e.g. this is true with ℓ_1 regularized generalized linear models). In general, prox Newton is not to be applied with care.

Well-known implementations using prox Newton: glmnet, QUIC, discussed more in §17.3.

Suppose you want to solve logistic regression with an ℓ_1 penalty

$$\min_\beta \ell(\beta) + \lambda ||\beta||_1.$$

When does it make sense to apply proximal Newton here? If proximal gradient is applied directly, $O(1/\epsilon)$ iterations. With prox Newton using prox gradient as an inner step, each inner step would take $O(1/\epsilon)$ iterations. Conclusion: proximal Newton does not seem like a good tradeoff here. In practice, use coordinate descent, covered later in the course.

Convergence Analysis

Following (Lee et al., 2012 [4]), assume that $f = g + h$, where g, h are convex and g is twice smooth. Assume further that

- $mI \preceq \nabla^2 g \preceq LI$ and $\nabla^2 g$ Lipschitz with parameter M
- $\text{prox}_H(\cdot)$ is exactly evaluable.

Theorem 17.1 *Proximal Newton method with backtracking line search converges global. Furthermore, for all $k \geq k_0$:*

$$||x^{(k)} - x^\star||_2 \leq \frac{M}{2m}||x^{(k-1)} - x^\star||_2^2$$

This is called local quadratic converge. After some point, to get within $f(x^{(k)}) - f^\star \leq \epsilon$, we require $O(\log\log(1/\epsilon))$ iterations. Note: Each iteration uses scaled prox evaluation.

Proof: Sketch, more details in (Lee et al., 2012 [4]).

- To prove global convergence, they show that at any step, the backtracking exit condition will be satisfied by

  $$t \leq \min\left\{1, \frac{2m}{L}(a-\alpha)\right\}$$

  Use this to show that the update direction converges to zero, which can only happen at the global minimum.

- To prove local quadratic convergence, they show that for large enough k, the pure step $t = 1$ eventually satisfies backtracking exit condition. Therefore

  $$\begin{aligned} ||x^+ - x^\star||_2 &\leq \frac{1}{\sqrt{m}}||x^+ - x^\star||_H \\ &\leq ||\operatorname{prox}_H(x - H^{-1}\nabla g(x)) - \operatorname{prox}_H(x^\star - H^{-1}\nabla g(x^\star))||_H \\ &\leq \frac{M}{2m}||x - x^\star||_2^2 \end{aligned}$$

■

Glmnet and QUIC

Two remarkable and widely used variations of proximal Newton method are glmnset and GUIC, which are applied to different classes of problems. They are very popular nowadays.

1. **glmnet** (Friedman et al. 2009, [5]). The idea of this algorithm is to apply Proximal Newton method to solve l_1 regularized generalized linear models. In each inner step, it applies coordinate descent. There is available of this algorithm a wrapper for the popular programs R and Matlab. Note that even when it is highly effective in quadratic loss with the l_1 penalty, it does not work outside this case.

2. **QUIC** (Hsiesh et al. 2011, [2]). It applies proximal Newton to solve graphical lasso problem. Since the Hessian is enormous, this algorithm uses some factorization tricks, so they do not need to totally store the full Hessian on memory. Once again, it uses coordinate descent in the inner steps.

Remark the fact that each one is highly used in their own fields.

As a side note, since proximal Newton perform few iterations, it requires far less evaluations of the gradient than proximal gradient. Hence, if those are expensive, proximal Newton may be a better choice.

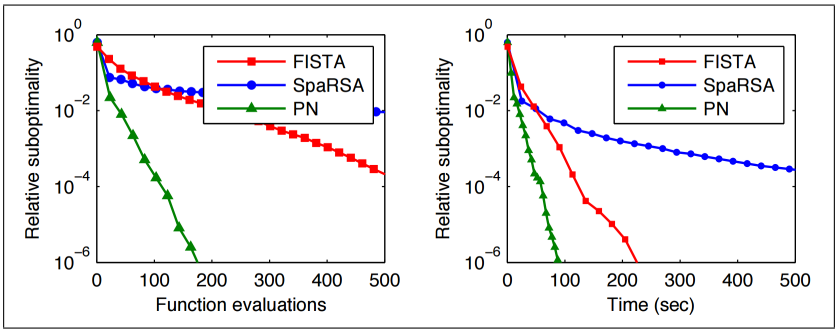
Example: lasso logistic regression

Now we are going to show and explain an example of Lasso regularized logistic regression. This example can be found in Lee et al. (2012). This experiment compares the performance of 3 different algorithms:

- FISTA (accelerated proximal gradiend descend)
- spaRSA (spectral projected gradiend method)
- PN (proximal Newton method)

Those three use the same data, with $n = 5000$, $p = 6000$, and a dense feature matrix X. The result is shown in figure 17.1:

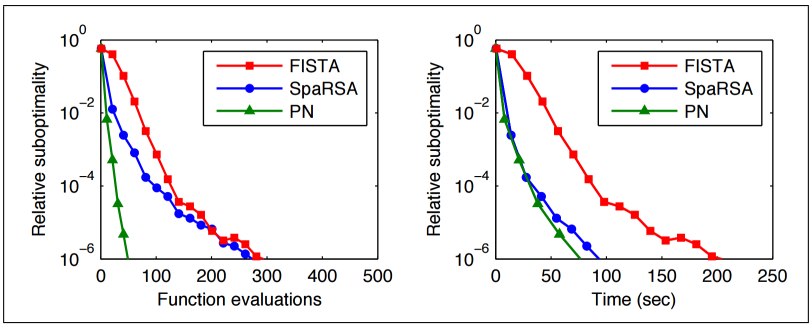
Figure 17.1: Relative suboptimality - X dense



As the figure shows, proximal Newton method uses far less iterations to achieve a high accurate solution, as expected. Here X is dense, and g and ∇g require expensive exp or log evaluations. Hence, they dominate the computational cost. However, and remarkable, even when the hessian may be also huge and highly costly, proximal Newton beat in time to their competitors. The reason is the expensive ∇g, which is evaluated far less times.

The next experiment uses $n = 542,000$, $p = 47,000$, and a sparse matrix X. The results are shown in figure 17.2:

Figure 17.2: Relative suboptimality - X sparse



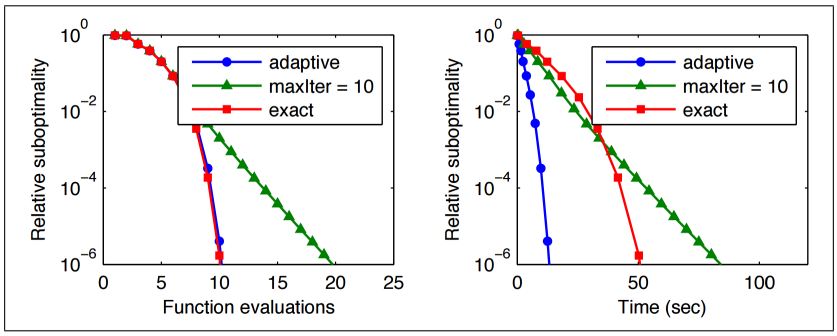
As expected, proximal Newton is the fastest method in time, since the hessian is now sparse. Once again, g and ∇g require expensive exp or log evaluations. Nevertheless, since X is sparse, they are not as costly as the previous example.

Inexact prox evaluations

Another different with proximal gradient is the fact that in reality, we always perform an inexact proximal evaluation. We never evaluate the proximal Newton explicitly because it is not usually characterizable in closed form.

In figure 17.3 is shown another example from Lee et al. (2012), once again about graphical lasso estimation. Here we use the following data: $n = 72$ and $p = 1255$.

Figure 17.3: Relative suboptimality - Third example



Looking at the first table, we can see that 10 inner iterations is not enough to ensure quadratic convergence (fast convergence). fast (quadratic convergence), but their adaptive stopping rule is

For the original smooth Newton method, the inner problem minimizes $\tilde{g}_{k-1}(z)$, a quadratic approximation to g with respect to $x^{(k-1)}$. Furthermore, the stopping rule for a specifically chosen "forcing" sequence $\eta_k, k = 1, 2, 3, ...$ is based on the following equation:

$$\|\nabla \tilde{g}_{k-1}(z)\|_2 \leq \eta_k \|\nabla g(x^{(k-1)})\|_2$$

Nevertheless, for proximal Newton, Lee et al. (2012) introduce generalized gradients instead of gradients. The resulting equation is:

$$\|\nabla G_{\tilde{f}_{k-1}/M}(z)\|_2 \leq \eta_k \|\nabla G_{f/M}(x^{(k-1)})\|_2$$

Here, $\tilde{f}_{k-1} = \tilde{g}_{k-1} + h$. Considering as usual the function strongly convex with parameter m and Lipschitz continuous with parameter M, we know that $m \preceq \nabla^2 g \preceq MI$. Setting η_k as follows:

$$\eta_k = \left\{ \frac{m}{2}, \frac{\|\nabla G_{\tilde{f}_{k-2}/M}(x^{(k-1)})\|_2 - \|\nabla G_{f/M}(x^{(k-1)})\|_2}{\|\nabla G_{f/M}(x^{(k-2)})\|_2} \right\}$$

This value of η_k proves that inexact proximal Newton has a local superlinear rate, but it cannot achieve the quadratic rate.

Proximal quasi-Newton methods

If the number of variables is large, the Hessian is prohibitive. The solution to this problem is quasi-Newton methods, which avoid computing the whole Hessian. We would like to remark two variations:

- On one hand, Lee et al. (2012) propose to iteratively update H_{k1} at each step using the denoted BFGS-type rules. Empirically, they show a very strong performance, with a local superlinear convergence.
- On the other hand, Tseng and Yun (2009) consider smooth plus block separable problems, and recommend approximating the Hessian in a blockwise fashion, combined with block coordinate descent method. This can be very helpful because only small Hessians are ever needed. They prove linear convergence.

Lastly, remark the fact that quasi-Newton methods are not only effective when the Hessian is expensive, but also when it is ill-conditioned (i.e. singular or close to singular).

Proximal Newton versus Tseng and Yuns method

In this section, we are going to discuss two different algorithms, that are usually not clearly distinguish in literature, but they are different to each other, with different method and guaranties.

Hence, we are going to compare the Proximal Newton for the next problem:

For instance, consider the following problem:

$$\min_x \quad g(x) + h(x)$$

where $h(x) = \sum_{b=1}^{B} h_b(x_b)$ separates over B blocks of coordinates. We can now compare Proximal Newton method, with the Tseng and Yun method (2009). This method was originally called coordinate gradient descent, but it is a bad name since it does not perform any coordinate gradient. Let us called it as: block proximal Newton. The distinction between both problem is when the quadratic approximation is performed.

- On one hand, Proximal Newton method replaces $g(x+\Delta)$ with $\tilde{g}(x+\Delta) = \nabla g(x)^T\Delta + \frac{1}{2}\Delta^T H\Delta$, and minimizes $\tilde{g}(x+\Delta) + h(x+\Delta)$ to find the update Δ. Finally, Δ is found by using block coordinate descent.
- On the other hand, Tseng and Yun's method iterates, for each block $b = 1, ..., B$, then replaces the smooth part with $\tilde{g}_b(x_b + \Delta_b) = \nabla_b g(x)^T\Delta_b + \frac{1}{22}\Delta_b^T H_b\Delta_b$, and finally it minimizes $\tilde{g}_b(x_b + \Delta_b) + h_b(x_b + \Delta_b)$ to find the update Δ_b for the specific block b.

In conclusion, they are not the same method and they should not be confused.

Whats wrong with projected Newton?

In this last section, we will show a problem related to projected Newton. Recall the indicator function of a convex set C, denoted $h = 1_C(x)$, and consider the following problem:

$$\min_x g(x) \text{ subject to } C$$

For this problem, it was proved that proximal gradient is reduced to projected gradient.

Nevertheless, this is not the case of projected Newton. Let expand the expression:

$$y = \arg\min_{z \in C} \frac{1}{2} \|x - H^{-1} \nabla g(x) - z\|_H^2$$

$$= \arg\min_{z \in C} \nabla g(x)^T (z - x) + frac12 (z - x)^T H (z - x)$$

The above is not a projection in general, only if $H = I$, $x - \nabla g(x)$ would be a projection onto C, but that is not the general case. Actually, this problem is quite hard. Thus, projected Newton does not usually follow from proximal Newton's method. In advanced topic of this course, a way to fix this problem will be discussed.

References

[1] J. Friedman and T. Hastie and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent", 2009

[2] C.J. Hsiesh and M.A. Sustik and I. Dhillon and P. Ravikumar, "Sparse inverse covariance matrix estimation using quadratic approximation", 2011

[3] M. Patriksson, "Cost approximation: a unified framework of descent algorithms for nonlinear programs", 1998.

[4] J. Lee and S. Yun, "Proximal Newton-type methods for minimizing composite functions", 2012.

[5] J. Friedman and T. Hastie, "A coordinate gradient descent method for nonsmooth separable minimization", 2009.