SDS 385 Ex 04:

Lazy Updates for L2 Regularization in Adagrad

September 30, 2016

Jennifer Starling

Problem:

For sparse matrices, any of the coefficients β_j are not necessarily updated at every iteration. Only active (non-zero) β_j's are updated at a given iteration. However, even if a β_j is not updated, the L2 regularization penalty still accumulates over each iteration.

The next time β_j is updated, before the update occurs, the accumulated L2 regularization penalty must be added to the β_j.

Example:

Iteration	Update
10	β_j updated
11	No update to β_j
12	No update to β_j
13	No update to β_j
14	β_j updated

For iteration 14, two things must happen:

(1) Add the accumulated regularization penalty for iterations 11, 12 and 13.
(2) Then update β_j like normal.

L2 Regularization Penalty for A Single Iteration

For a single iteration, β_j is updated as follows.

$\beta_j^{(i+1)} = \beta_j^{(i)} - \lambda\gamma^{(i)}\beta_j^{(i)}$ where:

$\beta_j^{(i)}$ is the β_j from the previous iteration.
λ is the L2 regularization λ penalty coefficient.
$\gamma^{(i)}$ is the $(step * adj.grad_j)$, the previous Adagrad scaled step size.

Recall, $adj.grad_j = grad_j * invSqrt(hist.grad_j + \epsilon)$, where $hist.grad_j =$ running sum of $(grad_j)^2$.

L2 Regularization Penalty for Consecutive Iterations

Let $\beta_j^{(1)}$ be a normal update. Let $\gamma^{(1)}$ be the $adj.grad_j$ corresponding to this update.
A key feature of Adagrad is that $adj.grad_j$ only changes when β_j is updated.

For subsequent iterations, if β_j were being updated, the recursion that occurs is as follows.

$$\beta_j^{(2)} = \beta_j^{(1)} - \lambda\gamma^{(1)}\beta_j^{(1)} = (1-\lambda\gamma^{(1)})\beta_j^{(1)}$$
$$\beta_j^{(3)} = \beta_j^{(2)} - \lambda\gamma^{(1)}\beta_j^{(2)} = (1-\lambda\gamma^{(1)})\beta_j^{(1)}) - \lambda\gamma^{(1)}(1-\lambda\gamma^{(1)})\beta_j^{(1)}) = (1-\lambda\gamma^{(1)})^2\beta_j^{(1)}$$
$$\vdots$$
$$\beta_j^{(i+1)} = (1-\lambda\gamma^{(1)})^i\beta_j^{(1)}$$

Summing Consecutive Iteration Penalties

The above recursion yields a β_j update for each iteration after the first one. Must sum these terms for the iterations where an update did not occur before proceeding with the update as usual.

• Let $i = 0$ the last update; $\beta_j^{(0)}$ is the last regularly updated β_j, with corresponding $\gamma^{(0)}$.
• Let $curr.iter$ represent the current iteration where β_j is now being updated again.
• Let $last.update = i = 0$, and call $skip = curr.iter - last.update$.

Sum the penalty terms since the last known update:

$$\beta_j^{\Delta} = \beta_j^{(1)} + \beta_j^{(2)} + \beta_j^{(3)} + ...$$

$$\beta_j^{\Delta} = \sum_{i=0}^{(skip-1)} (1 - \lambda\gamma^{(0)})^{i+1} \beta_j^{(0)}$$

The summation notation is effectively beginning with $\beta_j^{(1)}$ by raising to the power $(i + 1)$.

The summation stops at $curr.iter - last.update - 1 = skip - 1$ because we want to accumulate penalties up to and including the current β_j update.

Example:

- If the last update was iteration 10, and the current update is iteration 15, want
  to add penalties for 11, 12, 13 and 14.
- The penalty for iteration 15 will occur as part of the regular update of β_j in iteration 15.
- $curr.iter - last.update = 15 - 10 = 4$ here, so $skip = 4$.
- In this example,
  iteration 10 is i=0, so we raise to the (i+1)=1 power for (i=0), adding the 11th iteration penalty.
  iteration 11 is i=1, so we raise to the (i+1)=2 power for (i=1), adding the 12th iteration penalty.
  iteration 12 is i=2, so we raise to the (i+1)=3 power for (i=2), adding the 13th iteration penalty.
  iteration 13 is i=3, so we raise to the (i+1)=4 power for (i=3), adding the 14th iteration penalty.
  Now we are at i=4, but we have already added the 14th iteration penalty.
  This is why we are stopping at $skip - 1$ instead of stopping at $skip$.

Note that the summation just reuses $\beta_j^{(0)}$ and $\gamma^{(0)}$ from the last update; it is not required to know any interim β_j updates, thanks to the recursion.

The summation $\beta_j^{\Delta} = \sum_{i=0}^{(skip-1)} (1 - \lambda\gamma^{(0)})^{i+1} \beta_j^{(0)}$ has the form $\beta_j^{(0)} * \sum_{i=0}^{n} r^k = \beta_j^{(0)} \frac{1-r^{n+1}}{1-r}$ where

- $r = 1 - \lambda\gamma^{(0)}$
- $(n + 1) = (skip - 1) + 1 = skip$

Result

The cumulative L2 penalty term that must be added prior to the current update of β_j is:

$\beta_j^{\Delta} = \beta_j^{(0)} \left(\frac{1-(1-\lambda\gamma^{(0)})^{skip}}{\lambda\gamma^{(0)}} \right)$ with:

- $\beta_j^{(0)}$ = last updated β_j
- $\gamma^{(0)}$ = scaled adagrad step corresponding to last updated β_j.
- $step = current.iteration.number - last.updated.iteration.number$ for the β_j being updated.

Note regarding Minimization vs Maximization

The above result is for maximizing the gradient of the likelihood. If you are minimizing the gradient of the negative log-likelihood (as I am doing in my C++ code), the β_j^{Δ} term changes as follows.

For minimizing the gradient of the negative log-likelihood, updates to the gradient, including the penalty, are:

$$grad_j^{(i+1)} = (m_i * w_i - Y_i)X_{ij} + 2\lambda\beta_j^{(i)}$$

In this case, the penalty term is being added instead of subtracted.

Therefore, the summation $\beta_j^{\Delta} = \sum_{i=0}^{(skip-1)}(1+\lambda\gamma^{(0)})^{i+1}\beta_j^{(0)}$ has the form $\beta_j^{(0)} * \sum_{i=0}^{n} r^k = \beta_j^{(0)} \frac{1-r^{n+1}}{1-r}$ where

- $r = 1 + \lambda\gamma^{(0)}$
- $(n+1) = (skip - 1) + 1 = skip$

The summation therefore simplifies to:

$\beta_j^{\Delta} = \beta_j^{(0)} \left(\frac{1-(1+\lambda\gamma^{(0)})^{skip}}{1-\lambda\gamma^{(0)}} \right)$ with:

- $\beta_j^{(0)}$ = last updated β_j
- $\gamma^{(0)}$ = scaled adagrad step corresponding to last updated β_j.
- $step = current.iteration.number - last.updated.iteration.number$ for the β_j being updated.

Pseudo-code to illustrate this change is as follows, to provide clarity on correct signs of operations.

```
//Step 1: Apply accumulated l2 penalty to beta_hat_j.
//Skip = Number of iters since last update. (Skip=1 means updated last iter.)
skip = iter - last_updated(j);

//Calculate accum penalty.  Based on recursion defined in my notes.
gam = step*adj_grad(j);
accum_l2_penalty = beta_hat(j) * ((1-pow(1+lambda*gam,skip))/(1-lambda*gam));

//Add accum l2 penalty to beta_hat_j before doing current iteration update.
beta_hat(j) -= accum_l2_penalty;

//Step 2: Continue with updates for jth row in ith col.

//Calculate l2 norm penalty.
double l2penalty = 2*lambda*beta_hat(j);

//Update the jth gradient term.  Note: it.value() looks up Xji for nonzero entries.
grad_j = (mi*wi-Yi) * it.value() + l2penalty;

//Update the jth hist_grad term for Adagrad.
hist_grad(j) += grad_j * grad_j;

//Calculate the jth adj_grad term for Adagrad.
adj_grad(j) = grad_j * invSqrt(hist_grad(j) + epsilon);

//Calculate the updated jth beta_hat term.
beta_hat(j) -= step*adj_grad(j);
```