Retrospective Approximation

(MITACS GRI Internship)

V Pramodh Gopalan, Fabian Bastin

Dept. Informatique et Recherche Opérationnelle
Université de Montréal

July 28, 2022

Table of Contents

Sample Average Approximation (SAA)

- Consider the minimization problem:

$$\min_{x \in \mathbb{R}^p} f(x) = \mathbb{E}_\xi[F(x, \xi)]$$

- F is deterministic, differentiable, and is relatively easy to compute. It can be non-convex. ξ is a random vector. x is of a large dimension.
- Even after knowing ξ's distribution, computing $f(x)$ tough. Hence we approximate function as:

$$\min_{x \in \mathbb{R}^p} \tilde{f}_N(x) = \frac{1}{N} \sum_{i=1}^{N} F(x, \xi_i)$$

- $\{\xi_i\}_{i=1}^N$ are IID realisations of ξ. $\tilde{f}_N(x)$ is now deterministic, can use known techniques. This is Sample Average Approximation (SAA).

SAA

- Hope that as N grows, we converge to actual solution. Special cases include Neural Network training.
- To have theoretical guarantees, need N to be large. Standard Optimization techniques become infeasible.
- Can we get away with poorer approximations of f at the beginning?

Retrospective Approximation(RA)

- RA improves upon SAA by considering a sequence of SAA problems.
- We need a non-decreasing sequence of sample sizes $\{N_k\}$, sequence of error-tolerances ϵ_k, and a solver capable of solving SAA problems to any desired tolerance level ϵ.

Here is the RA algorithm: given an initial guess X_0

1. At iteration k, generate an SAA problem with sample size N_k.
2. With X_{k-1} as initial guess, and ϵ_k as the error-tolerance, use the solver to solve the SAA.
3. Obtain the solution X_k.

- **Disadvantage:** In lower sample sizes, the solver can be prone to overfitting.

L-BFGS

- LBFGS (Limited-Memory Broyden–Fletcher–Goldfarb–Shanno) is an optimization algorithm designed to approximate BFGS but using limited memory.
- BFGS approximates the Inverse Hessian, to precondition the gradient to get the descent direction. Cheaper than using true hessian or full matrix approximation.
- BFGS stores dense $p \times p$ Hessian matrices, which becomes a limitation.
- L-BFGS stores last m (< 10) updates of the iterates and gradients. Lesser memory consumption. These are used to construct inverse Hessian approximations.

Common Random Numbers (CRN)

- CRN is a simple method for increasing the efficiency of estimating *difference* in performance.
- It introduces dependence in between random variables to reduce variance. Usually, this means the systems be simulated using same stream of random numbers.
- Precisely, given two random variables X and Y, and two functions f and g, we want to find

$$\mathbb{E}[f(X) - g(Y)]$$

- The effort required to find this is dependant on the variance

$$\mathbf{Var}[f(X) - g(Y)] = \mathbf{Var}[f(X)] + \mathbf{Var}[g(Y)] - 2\mathbf{Cov}[f(X), g(Y)]$$

- if X and Y were to be independent, $\mathbf{Cov}[f(X), g(Y)] = 0$, but using CRN causes positive correlation, which reduces variance.

How to construct stopping tests in RA?

- Can we use CRN to find out when a function stops decreasing in value?
- The optimization problem we have is:

$$\min_{x \in \mathbb{R}^p} \tilde{f}_N(x) = \frac{1}{N} \sum_{i=1}^{N} f(x, \xi_i)$$

- Say we have current iterate X_k and $X_{k+1} = X_k + \eta d$, where d is a direction of descent yielded by a solver, η being an appropriate step size.

How to construct Stopping tests in RA?

- In the RA scheme, the subsampled dataset for each iteration is picked at random. So, given a sample size N_k, we calculate

$$\sigma^2 = \textbf{Var}[f_{N_k}(X_k) - f_{N_k}(X_{k+1})]$$

- Now, we build an α confidence interval around $\mu = \mathbb{E}[f_{N_k}(X_k) - f_{N_k}(X_{k+1})]$ using the variance from earlier step.
- Therefore, if

$$\mu - z_{1-\alpha/2}\sigma > 0$$

we continue the algorithm. Here z_γ returns the γ quantile of the standard Normal distribution.

- These stopping tests take into consideration the noise in gradient, and prevent overfitting in lower sample size problems.

Comparing RA and SGD

- We use a logistic regression example with 10 covariates and $10,000$ samples as a toy dataset to demonstrate that RA can be better than SGD.
- Use the logistic loss function, and set the mini batch sizes and ϵ (gradient norm here) by hand.
- Note, we need to use the number of gradient calls as a metric. It serves as a placeholder for amount of work done.
- Even for the best case batch size for SGD, we see that it takes $1.6x$ more gradient calls than RA (SGD as inner solver).
- One disadvantage is tuning the epsilons by hand, which is time consuming in large scale applications.

Including Stopping Tests

- In this case, we follow a setup illustrated in [Newton et al., 2021].
- Ordinary Least Squares Regression Problem with $1,000$ covariates and $10,000$ data samples.
- We use an L-BFGS algorithm, but here the stored history is carried over from one inner iteration to next. We use the Armijo backtracking line search.
- For minibatch scaling, we use a geometric progression, where $N_{k+1} = 1.2N_k$, with $N_0 = 2000$, for 5 inner iterations.
- In this case, RA performs just as well as SGD, taking approximately the same amount of gradient calls.

Future Improvements

- Does the sample size have to be non-decreasing? Decreasing sample sizes might help escape saddle points.
- Try stopping tests on non-convex functions.
- Reuse computation to adapt it to online settings.

References I

Newton, D., Bollapragada, R., Pasupathy, R., and Yip, N. K. (2021).
Retrospective approximation for smooth stochastic optimization.