Quasi-Newton Methods for Machine Learning:

Forget the Past, Just Sample

A.S. Berahas, M. Jahani, P. Richtarik, M. Takac (2021)

MLRG Spring 2022
Betty Shea

I’ll tune out of this talk

- Don’t use quasi-Newton (QN) methods in my research
- Just implementation details

Okay maybe I’ll half listen

- QN methods don’t work well for neural nets
- Paper tries to fix this

Why this paper? Practical aspects

- QN methods don’t work well for nonconvex problems

- How bad is the QN Hessian approximation?
  - Bad enough to just toss it out?

Why this paper? Goes against folk wisdom

- Don’t waste information
  - Reuse historical curvature (gradient and momentum) information

Why this paper? Goes against folk wisdom

- Don’t waste information
    - Reuse historical curvature (gradient and momentum) information
    - It’s not a waste if it’s no longer good info

Why this paper? Goes against folk wisdom

- Don’t waste information
  - Reuse historical curvature (gradient and momentum) information
  - It’s not a waste if it’s no longer good info

- Wastefulness leads to slowness
  - Starting from scratch at every step really bad for first order methods

Why this paper? Goes against folk wisdom

- Don’t waste information
  - Reuse historical curvature (gradient and momentum) information
  - It’s not a waste if it’s no longer good info
- Wastefulness leads to slowness
  - Starting from scratch at every step really bad for first order methods
  - It’s actually not too painful in practice with parallelization

Why this paper? Simple approach

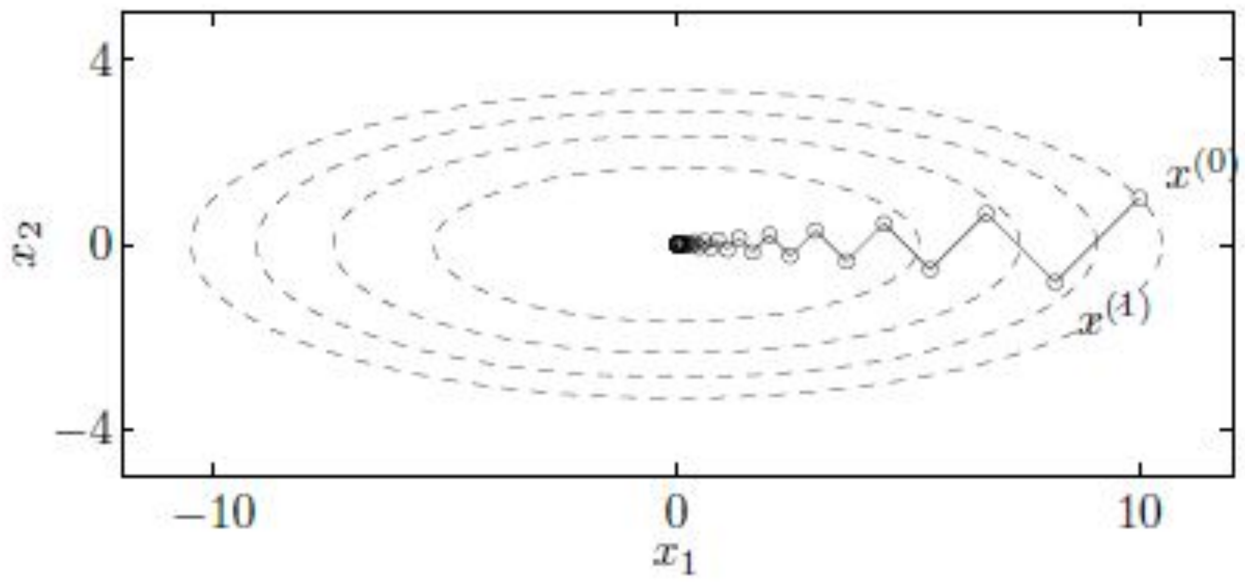
- Rooting for the not-too-clever ideas

- Random sampling is charmingly brute force
  - Replace directions taken in the past with random directions
  - Could this possibly work?

QN: high-level goal

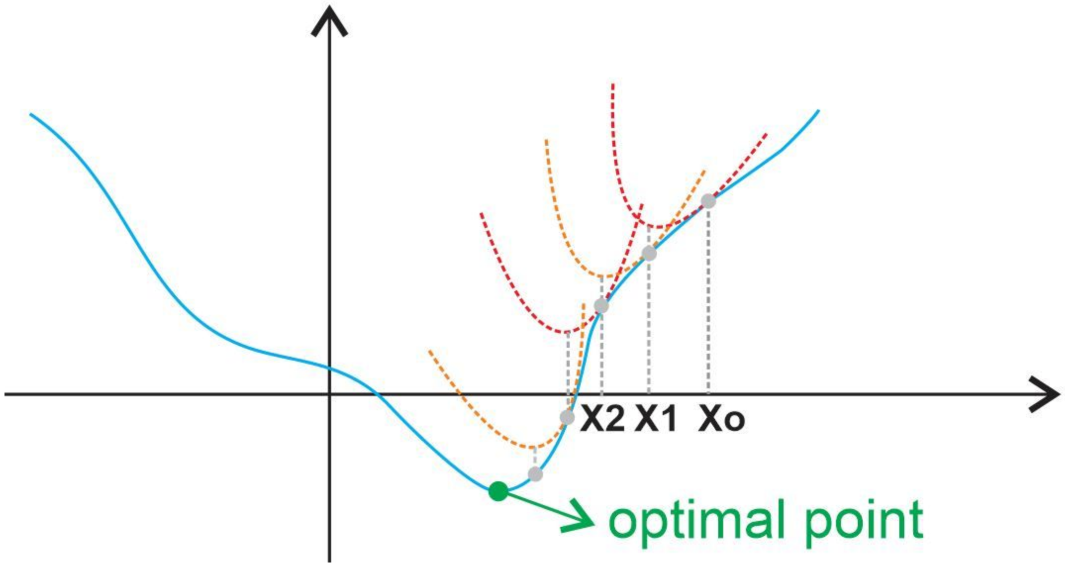
- Quasi-Newton methods
  - “attempt to combine the speed of Newton’s method and the scalability of first-order methods by incorporating curvature information in a judicious manner….”

Gradient descent (GD)



Picture from Boyd and Vandenberghe, Convex Optimization, 2004.

Newton’s method



Picture from Ardian Umam’s blog post https://ardianumam.wordpress.com/

QN: some details

- Approximate Hessian with just first order information
- Curvature pairs (s_k, y_k)

$$s_k = w_k - w_{k-1}, \quad y_k = \nabla F(w_k) - \nabla F(w_{k-1})$$

- Characteristics of approximation
    - Positive definite
    - Symmetric
    - Secant equation

$$B_{k+1} s_k = y_k$$

QN: some details

- Update at kth iteration (H_k is QN’s approx of the inverse Hessian)

$$w_{k+1} = w_k - \alpha_k H_k \nabla F(w_k)$$

- H_k lies between identity (GD) and true Hessian (Newton’s)

- Low rank update of H_k to get H_{k+1}

QN: some details

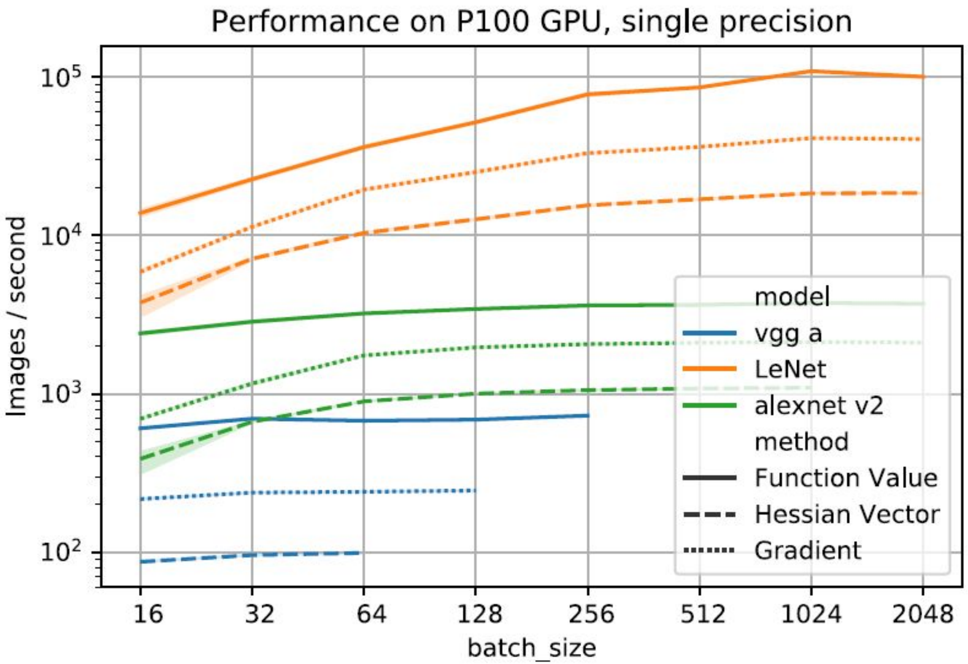
- Many flavors
  - Paper looks at BFGS and SR1
- Memory efficient versions (“limited” memory)
  - Paper looks at l-BFGS and l-SR1

Paper outline

- Exploratory experiments
- Algorithms
- Convergence analysis
- Main experiments

Exploratory experiments

- “the cost of computing function values, gradients and Hessian vector products appears to be comparable”



Finding new curvature pairs

Algorithm 1 Compute new (S, Y) curvature pairs

Input: w (iterate), m (memory), r (sampling radius), $S = [\ ]$, $Y = [\ ]$ (curvature pair containers).

1: Compute $\nabla F(w)$
2: **for** $i = 1, 2, ..., m$ **do**
3: Sample a random direction σ_i
4: Construct $\bar{w} = w + r\sigma_i$
5: Set $s = w - \bar{w}$ and

$$y = \begin{cases} \nabla F(w) - \nabla F(\bar{w}), & \text{Option I} \\ \nabla^2 F(w) s, & \text{Option II} \end{cases}$$

6: Set $S = [S\ s]$ and $Y = [Y\ y]$
7: **end for**

Output: S, Y

Proposed algorithm for sampled lBFGS

Algorithm 2 Sampled LBFGS (S-LBFGS)

Input: w_0 (initial iterate), m (memory), r (sampling radius).

1: **for** $k = 0, 1, 2, \ldots$ **do**
2: Compute new (S_k, Y_k) pairs via Algorithm 1
3: Compute the search direction $p_k = -H_k \nabla F(w_k)$
4: Choose the steplength $\alpha_k > 0$
5: Set $w_{k+1} = w_k + \alpha_k p_k$
6: **end for**

- Does not start with the gradient direction
- Samples new pairs instead of updating with newest pair
- Steplength either constant or set with a line search
- Sampled version of l-SR1 with trust region

Computational cost and storage

Table 2. Summary of Computational Cost and Storage (per iteration) for different Quasi-Newton methods.

method	computational cost	storage
BFGS	$nd + d^2 + \kappa_{ls} nd$	d^2
LBFGS	$nd + 4md + \kappa_{ls} nd$	$2md$
S-LBFGS	$nd + mnd + 4md + \kappa_{ls} nd$	$2md$
SR1	$nd + d^2 + nd + \kappa_{tr} d^2$	d^2
LSR1	$nd + nd + \kappa_{tr} md$	$2md$
S-LSR1	$nd + mnd + nd + \kappa_{tr} md$	$2md$

- m = size of memory, n = number of examples, d = dimensionality
- Same storage requirements as limited memory versions
- Extra mnd computational cost per iteration

Convergence analysis

- Deterministic
- Constant stepsize and with Armijo linesearch
- Proposed methods not worse than regular limited memory versions
    - Strongly convex → converges linearly to optimal solution
    - Nonconvex → converges to a stationary point
        - Probability of accepting curvature pairs that satisfy $s^T y > \epsilon ||s||^2$

Main experiments

1. Toy problem (find boundary between two classes)

Table 4. Toy Classification Problem: Neural Network Details

network	structure	d
small	2-2-2-2-2-2	36
medium	2-4-8-8-4-2	176
large	2-10-20-20-10-2	908

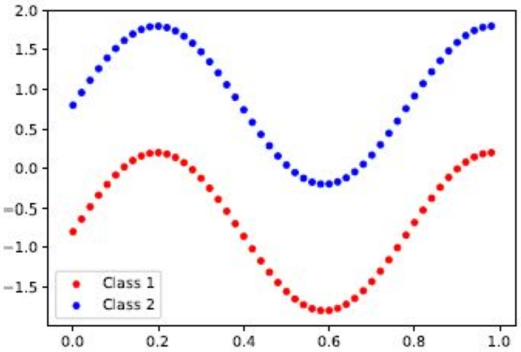


Figure 9. Toy Classification Problem

Main experiments

2. L2-reg logistic regression on rcv1 (d=47,236) and w8a (d=300)

3. Nonlinear least squares on rcv1 and w8a

4. Train on MNIST and CIFAR10 with deep NNs

Table 1. Deep Neural Networks used in the experiments.

model	d	input	# classes
LeNet	3.2M	$28 \times 28 \times 3$	10
alexnet v2	50.3M	$224 \times 224 \times 3$	1,000
vgg a	132.8M	$224 \times 224 \times 3$	1,000

Main experiments

Benchmarks used

- ADAM. "we tuned the steplength and batch size for each problem independently"
- GD. Armijo for steplength
- BFGS. Armijo for steplength. Full (inverse) Hessian approximations
- SR1. TR subproblem solved using CG-Steihaug. Full (inverse) Hessian approximation
- lBFGS. two-loop recursion
- SR1. Compact representations of SR1 matrices

Results: toy problem

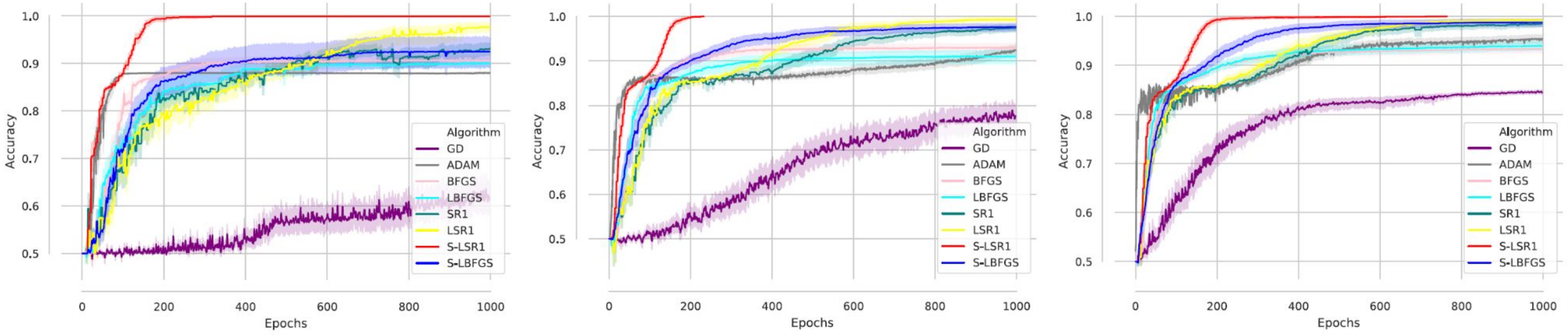


Figure 10. Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on toy classification problems. Networks: `small` (left); `medium` (center); `large` (right).

Results: logistic regression

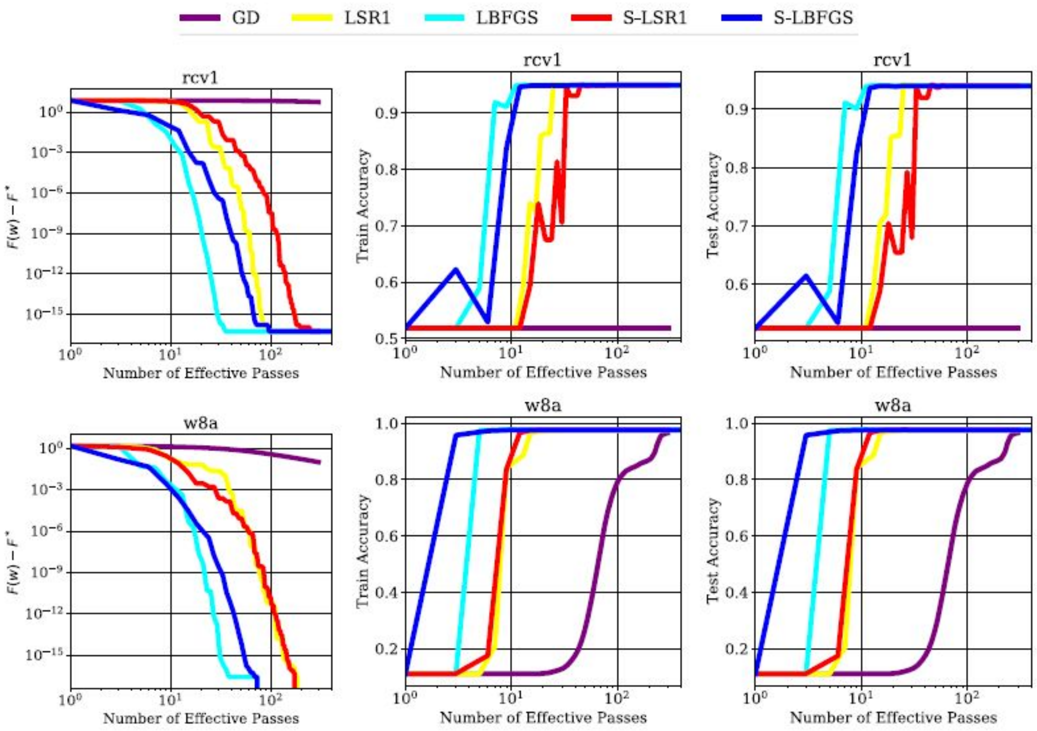


Figure 12. Performance of GD, LBFGS, LSR1, S-LSR1 and S-LBFGS on Logistic Regression problems; `rcv1` dataset (first row) and `w8a` dataset (second row).

- Better on w8a than rcv1?
- Sampled version seems better initially?
- "competitive with the classical variants"

Results: nonlinear least squares

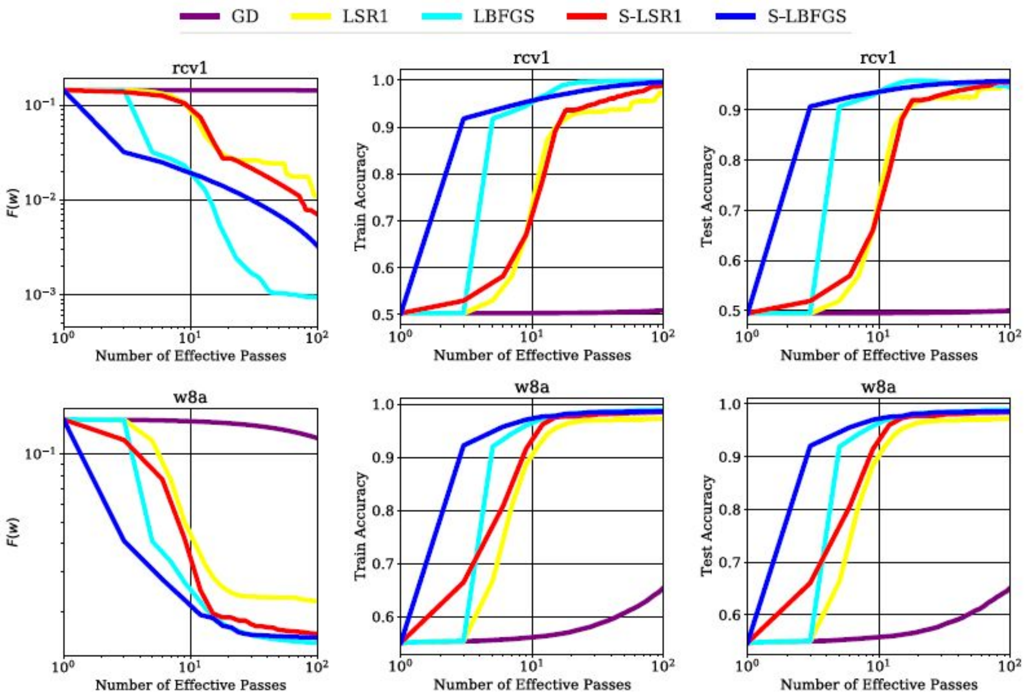


Figure 13. Performance of GD, LBFGS, LSR1, S-LSR1 and S-LBFGS on Nonlinear Least Squares problems; `rcv1` dataset (first row) and `w8a` dataset (second row).

- “more recent, local and reliable curvature information indispensable in the nonconvex setting”
- “outperforms their classical counterparts across the board”

Results: MNIST

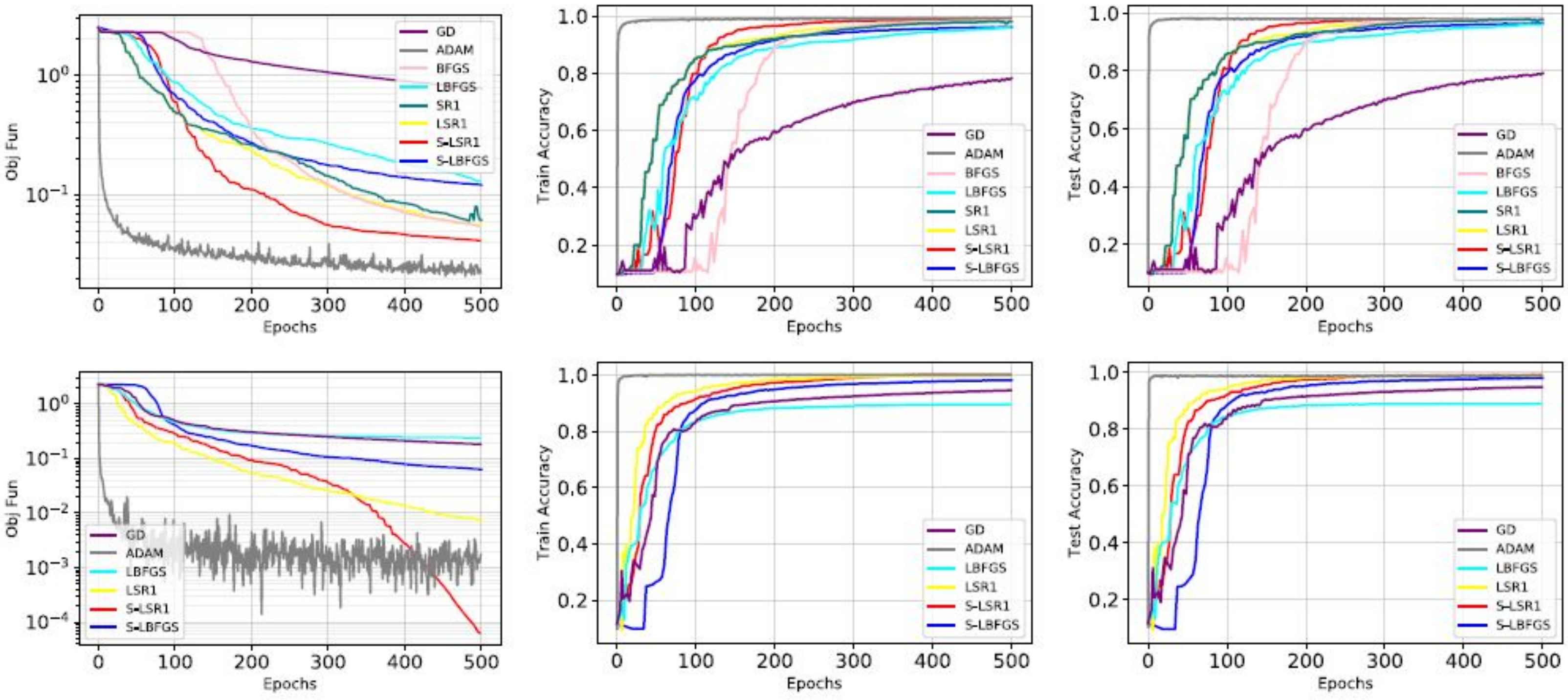


Figure 14. Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on MNIST problems on Net1 (first row) and Net2 (second row).

Results: CIFAR10

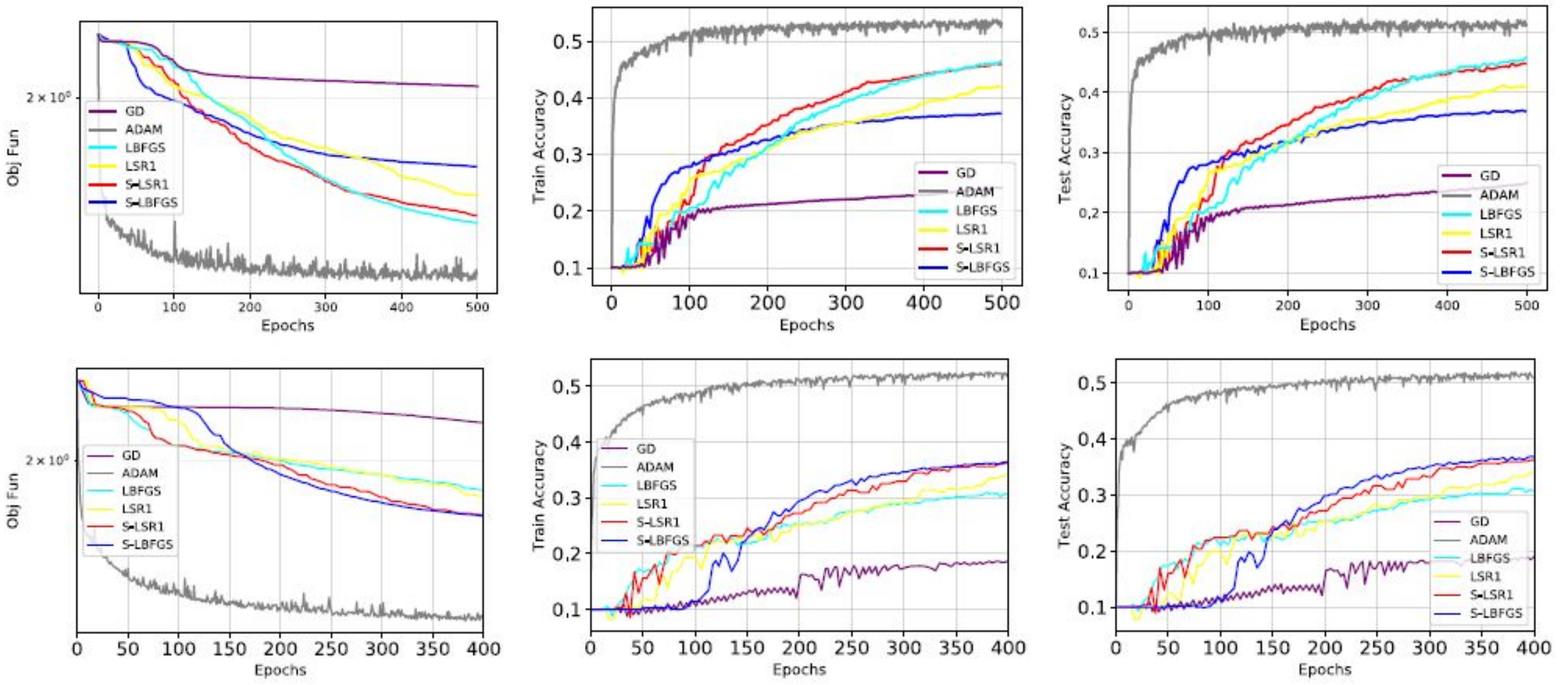


Figure 15. Performance of GD, ADAM, BFGS, LBFGS, SR1, LSR1, S-LSR1 and S-LBFGS on `CIFAR10` problems on `Net3` (first row) and `Net4` (second row).

Results: training deep NNs

- outperformed classical variants

- “goal of these experiments is not to perform better than ADAM”
  - stochastic vs deterministic
  - well-tuned ADAM

- S-LSR1 has better performance than S-LBFGS
  - “possible utilization of negative curvature in the updates”

Discussion

- Contrarian approach to QN methods
  - theoretically not-worse
  - maybe better in practice
- Could this method be useful in deep learning?
  - Yes if you are already using QN methods
  - Maybe not if you areusing ADAM
- Is the QN approximation so bad we can just throw it out?
  - Maybe in nonconvex settings
- Does this address the issues QN has in nonconvex settings?
  - Not sure - maybe more to look into