Learning goal Ch. 2, 3 and 6: Understand this slide
Line-search unconstrained optimization

$$\min_x f(x)$$

1. Initial guess x_0
2. While termination criteria not fulfilled
   a) Find descent direction p_k from x_k
   b) Find appropriate step length α_k ; set $x_{k+1} = x_k + \alpha_k p_k$
   c) $k = k+1$
3. $x_M = x^*$? (possibly check sufficient conditions for optimality)
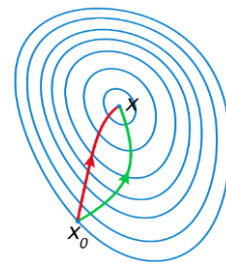
A comparison of steepest descent and Newton's method. Newton's method uses curvature information to take a more direct route. (wikipedia.org)

Termination criteria:
Stop when first of these become true:
- $\|\nabla f(x_k)\| \leq \epsilon$ (necessary condition)
- $\|x_k - x_{k-1}\| \leq \epsilon$ (no progress)
- $\|f(x_k) - f(x_{k-1})\| \leq \epsilon$ (no progress)
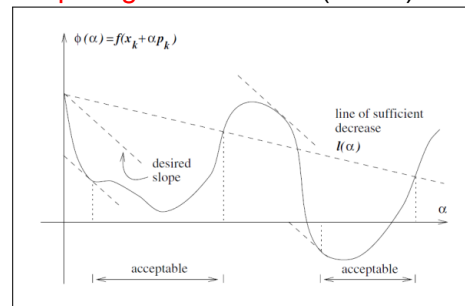- $k \leq k_{\max}$ (kept on too long)

Descent directions:
- Steepest descent
  $$p_k = -\nabla f(x_k)$$
- Newton
  $$p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$
- Quasi-Newton
  $$p_k = -B_k^{-1} \nabla f(x_k)$$
  $$B_k \approx \nabla^2 f(x_k)$$

How to calculate derivatives – Ch. 8

Step length line search (Wolfe):

How many iterations? (Convergence rates)

Outline today

Quasi-Newton
- Q-N efficiently produce good search directions
  - Steepest descent: Many iterations, but each iteration cheap (need only gradient)
  - Newton: Few iterations, but each iteration expensive (need also Hessian)
  - Quasi-Newton: Few iterations by approximating the Hessian using only the gradient
- Secant condition
- BFGS (and DFP) Hessian approximation update formulas

- What is this used for? A lot!
  General optimization, nonlinear MPC, machine learning,
  image processing, …

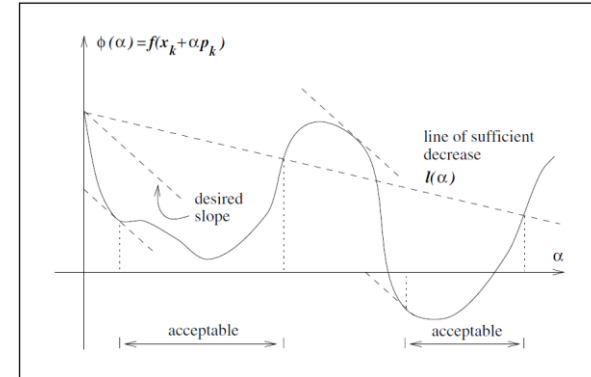Reference: N&W Ch.6-6.1 (Superficially 7.1)

Line search
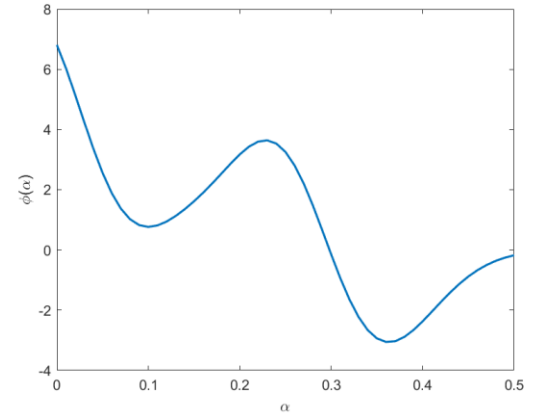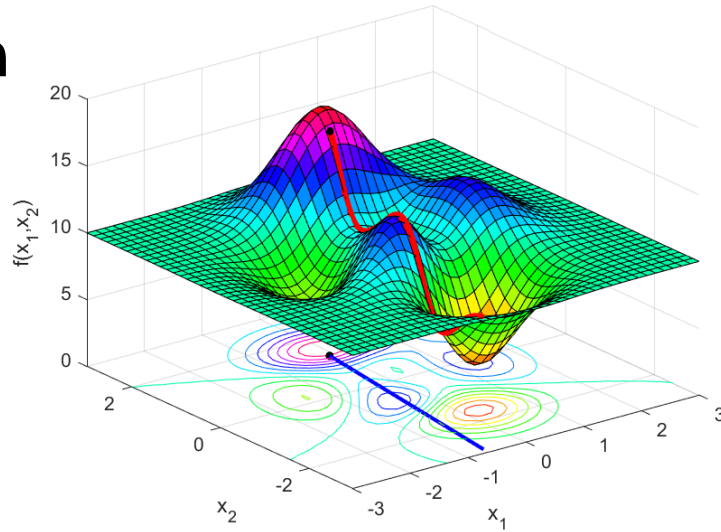
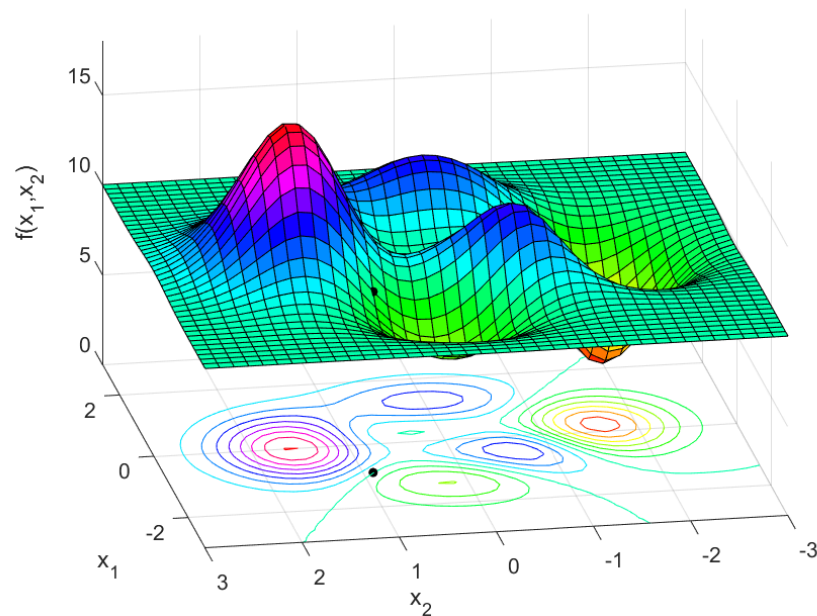Conditions for a good step length: Wolfe conditions

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^\top p_k \qquad \text{Sufficient decrease (Armijo condition)}$$

$$\nabla f(x_k + \alpha_k p_k)^\top p_k \geq c_2 \nabla f_k^\top p_k \qquad \text{Desired slope (Curvature condition)}$$
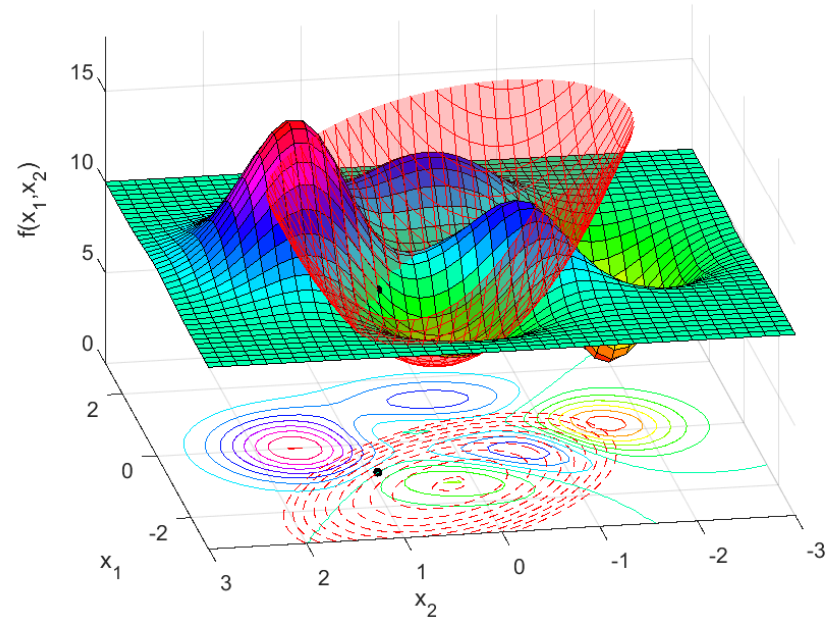
Newton's method

$$x_0 = (-0.9, 0.9)^\top$$

Newton's method

$$f(x_k + p) \approx m_k(p) = f_k + \nabla f_k^\top p + \tfrac{1}{2} p^\top \nabla^2 f_k p$$

Hessian modification for Newton's method

- For $p_k = -B_k^{-1} \nabla f(x_k)$ to be a descent direction, we need $B_k > 0$

- In general, this does not hold true for Newton, $B_k = \nabla^2 f(x_k)$. We therefore modify the Hessian when it is not positive definite:

Algorithm 3.2 (Line Search Newton with Modification).

Given initial point x_0;

for $k = 0, 1, 2, \ldots$

Factorize the matrix $B_k = \nabla^2 f(x_k) + E_k$, where $E_k = 0$ if $\nabla^2 f(x_k)$ is sufficiently positive definite; otherwise, E_k is chosen to ensure that B_k is sufficiently positive definite;

Solve $B_k p_k = -\nabla f(x_k)$;
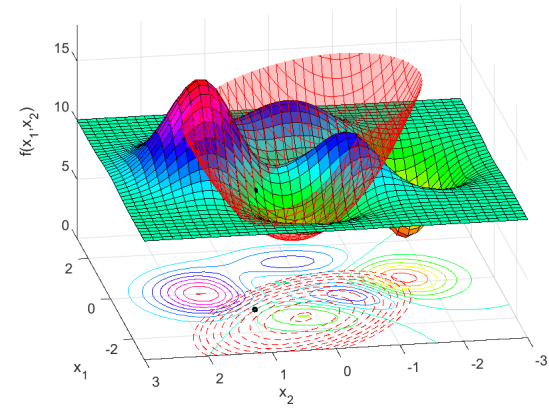
Set $x_{k+1} \leftarrow x_k + \alpha_k p_k$, where α_k satisfies the Wolfe, Goldstein, or Armijo backtracking conditions;

end

- A good, but inefficient method: $E_k = \tau_k I, \quad \tau_k = \max\left(0, \delta - \lambda_{\min}\left(\nabla^2 f(x_k)\right)\right)$
- More efficient to do "similar" changes during factorization process
  - E.g. "modified Cholesky" method

Newton's method

$$x_{k+1} = x_k + \alpha_k p_k; \qquad p_k = -\left[\nabla^2 f_k\right]^{-1} \nabla f_k$$

Quasi-Newton
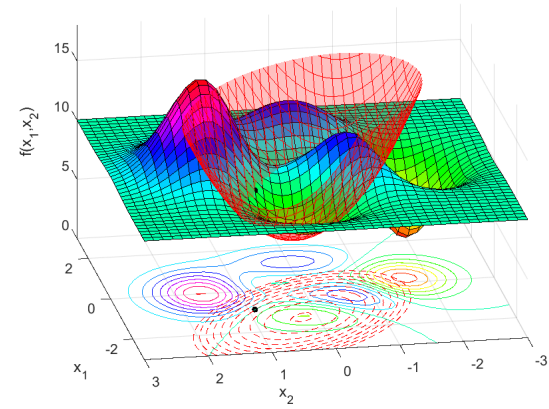
Quasi-Newton

- Invented by Bill Davidon, physicist at Argonne National Labs, around mid 1950s
- "The Davidon-Fletcher-Powell (DFP) update formula"

- "One of the most creative ideas in nonlinear optimization", tremendous impact
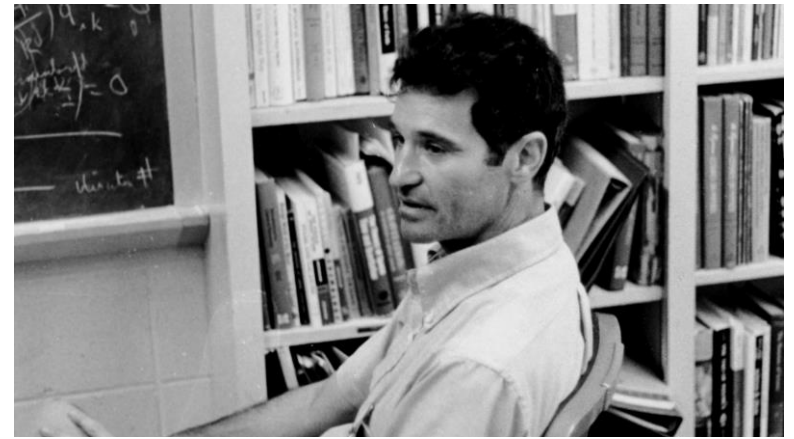
Fun fact 1: His first paper on Q-N was not accepted for publication before 1991, over thirty years later

Fun fact 2: Bill Davidon was a peace activist, mastermind behind a break-in at an FBI office in 1971 (Movie: "1971")

Secant condition

Positive definite requirement

DFP update formula

Inverse update formula

DFP formula for updating B_k:

$$B_{k+1} = \left(I - \rho_k y_k s_k^\top\right) B_k \left(I - \rho_k s_k y_k^\top\right) + \rho_k y_k y_k^\top, \quad \rho_k = \frac{1}{y_k^\top s_k}$$

But: since we need B_k^{-1} in $p_k = -B_k^{-1}\nabla f_k$, can we update $H_k = B_k^{-1}$ instead?

Yes: Multiply out DFP as

$$B_{k+1} = B_k + \begin{pmatrix} B_k s_k & y_k \end{pmatrix} \begin{pmatrix} 0 & -\rho_k \\ -\rho_k & \rho_k + s_k^\top B_k s_k \rho_k^2 \end{pmatrix} \begin{pmatrix} s_k^\top B_k \\ y_k^\top \end{pmatrix}, \quad \rho_k = \frac{1}{y_k^\top s_k},$$

use the *matrix inversion lemma* (Sherman-Morrison-Woodbury formula)

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

to obtain the inverse DFP formula:

$$H_{k+1} = H_k - \frac{H_k y_k y_k^\top H_k}{y_k^\top H_k y_k} + \frac{s_k s_k^\top}{y_k^\top s_k}$$

NTNU | Norwegian University of Science and Technology

BFGS update formula

BFGS

Broyden, Fletcher, Goldfarb, Shanno

BFGS method

Algorithm 6.1 (BFGS Method).

Given starting point x_0, convergence tolerance $\epsilon > 0$,
   inverse Hessian approximation H_0;

$k \leftarrow 0$;

while $\|\nabla f_k\| > \epsilon$;

   Compute search direction

$$p_k = -H_k \nabla f_k;$$

   Set $x_{k+1} = x_k + \alpha_k p_k$ where α_k is computed from a line search
      procedure to satisfy the Wolfe conditions (3.6);

   Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;

   Compute H_{k+1} by means of (6.17);

   $k \leftarrow k + 1$;

end (**while**)

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$$

Local convergence rates (close to optimum)

Steepest descent:
Linear convergence

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq r \quad \text{for all } k \text{ sufficiently large}, r \in (0,1)$$
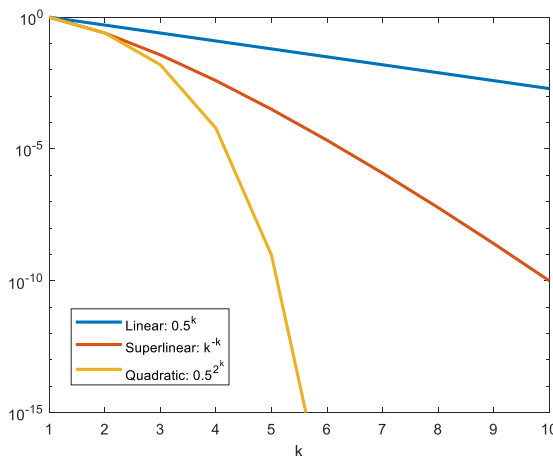
Newton:
Quadratic convergence

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^2} \leq M \quad \text{for all } k \text{ sufficiently large}, M > 0$$
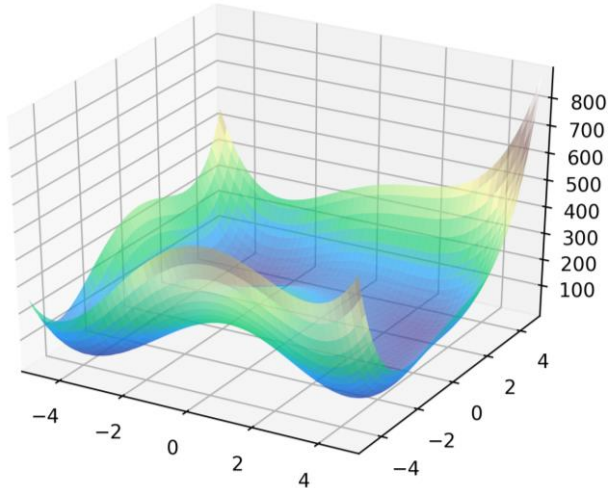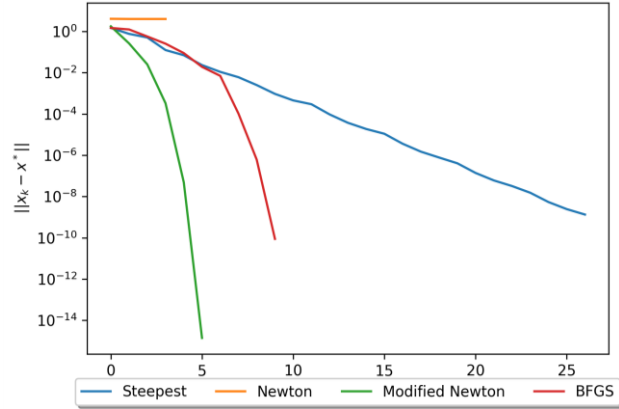
Quasi-Newton:
Superlinear convergence

$$\lim_{k \to \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0$$

$$\frac{\|x_{k+1} - x^*\|}{\|x_0\|}$$

Example: Himmelblau

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$
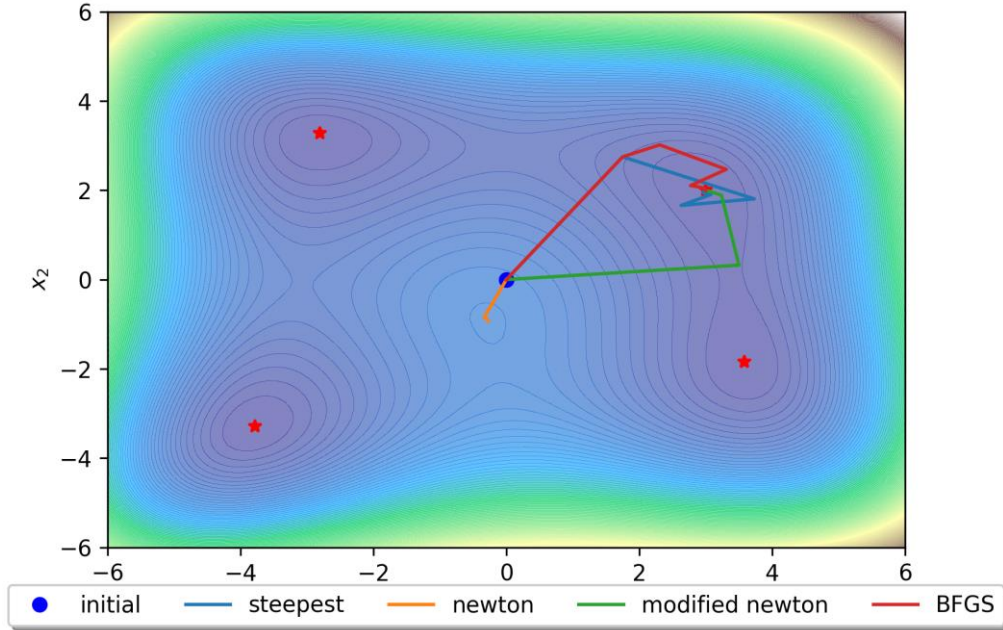
Example (from book)

- Using steepest descent, BFGS and inexact Newton on Rosenbrock function

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

wikipedia.org

- Iterations from starting point (-1.2,1):
  - Steepest descent: 5264
  - BFGS: 34
  - Newton: 21

- Last iterations; value of $\|x_k - x^*\|$
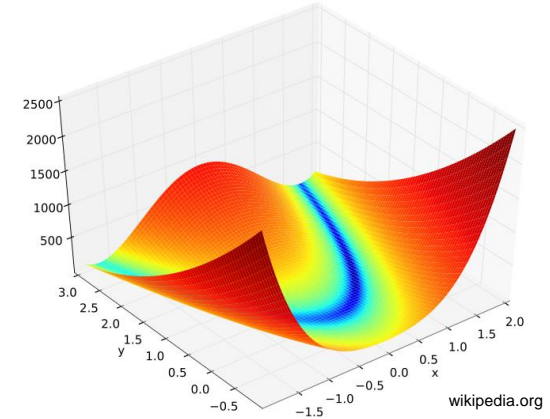
steepest descent	BFGS	Newton
1.827e-04	1.70e-03	3.48e-02
1.826e-04	1.17e-03	1.44e-02
1.824e-04	1.34e-04	1.82e-04
1.823e-04	1.01e-06	1.17e-08

Example: image deblurring
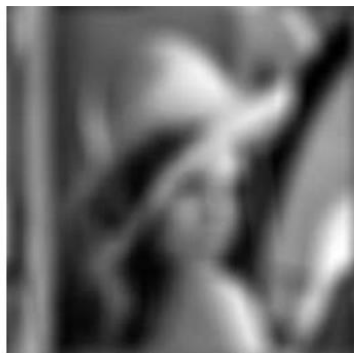
Original image   **Blurred image**   **Reconstruction**

Figures from (Wang et. al, 2009)

Given corrupted $m \times n$ image represented as vector $y \in \mathbb{R}^{m \cdot n}$, find $x \in \mathbb{R}^{m \cdot n}$ by solving the optimization problem

$$\operatorname*{minimize}_{x} \quad \|K*x-y\|_2^2 + \lambda \left(\sum_{i=1}^{n-1} |x_{mi} - x_{m(i+1)}| + \sum_{i=1}^{m-1} |x_{ni} - x_{n(i+1)}| \right)$$

where $K*$ denotes 2D convolution with some filter K

Example: machine learning

Virtually all machine learning algorithms can be expressed as minimizing a *loss function* over observed data

Given inputs $x^{(i)} \in \mathcal{X}$, desired outputs $y^{(i)} \in \mathcal{Y}$, hypothesis function $h_\theta : \mathcal{X} \to \mathcal{Y}$ defined by parameters $\theta \in \mathbb{R}^n$, and loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$

Machine learning algorithms solve optimization problem

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{m} \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

http://www.cs.cmu.edu/afs/cs/academic/class/15780-s16/www/slides/optimization.pdf

Quasi-Newton in machine learning

On optimization methods for deep learning

Quoc V. Le — QUOCLE@CS.STANFORD.EDU
Jiquan Ngiam — JNGIAM@CS.STANFORD.EDU
Adam Coates — ACOATES@CS.STANFORD.EDU
Abhik Lahiri — ALAHIRI@CS.STANFORD.EDU
Bobby Prochnow — PROCHNOW@CS.STANFORD.EDU
Andrew Y. Ng — ANG@CS.STANFORD.EDU

Computer Science Department, Stanford University, Stanford, CA 94305, USA

Abstract

The predominant methodology in training deep learning advocates the use of stochastic gradient descent methods (SGDs). Despite its ease of implementation, SGDs are difficult to tune and parallelize. These problems make it challenging to develop, debug and scale up deep learning algorithms with SGDs. In this paper, we show that more sophisticated off-the-shelf optimization methods such as Limited memory BFGS (L-BFGS) and Conjugate gradient (CG) with line search can significantly simplify and speed up the process of pretraining deep algorithms. In our experiments, the difference between L-BFGS/CG and SGDs are more pronounced if we consider algorithmic extensions (e.g., sparsity regularization) and hardware extensions (e.g., GPUs or computer clusters). Our experiments with distributed optimization support the use of L-BFGS with locally connected networks and convolutional neural networks. Using L-BFGS, our convolutional network model achieves 0.69% on the standard MNIST dataset. This is a state-of-the-art result on MNIST among algorithms that do not use distortions or pretraining.

2008; Zinkevich et al., 2010). A strength of SGDs is that they are simple to implement and also fast for problems that have many training examples.

However, SGD methods have many disadvantages. One key disadvantage of SGDs is that they require much manual tuning of optimization parameters such as learning rates and convergence criteria. If one does not know the task at hand well, it is very difficult to find a good learning rate or a good convergence criterion. A standard strategy in this case is to run the learning algorithm with many optimization parameters and pick the model that gives the best performance on a validation set. Since one needs to search over the large space of possible optimization parameters, this makes SGDs difficult to train in settings where running the optimization procedure many times is computationally expensive. The second weakness of SGDs is that they are inherently sequential: it is very difficult to parallelize them using GPUs or distribute them using computer clusters.

Batch methods, such as Limited memory BFGS (L-BFGS) or Conjugate Gradient (CG), with the presence of a line search procedure, are usually much more stable to train and easier to check for convergence. These methods also enjoy parallelism by computing the gradient on GPUs (Raina et al., 2009) and/or distributing that computation across machines (Chu et al., 2007). These methods, conventionally considered to

Norwegian University of Science and Technology