Using Gradient Descent for Optimization and Learning

Nicolas Le Roux

15 May 2009

Why having a good optimizer?

- plenty of data available everywhere
- extract information efficiently

Why gradient descent?

- cheap
- suitable for large models
- it works

Optimization vs Learning

Optimization

- function f to minimize
- time $T(\rho)$ to reach error level ρ

Optimization vs Learning

Optimization

- function f to minimize
- time $T(\rho)$ to reach error level ρ

Learning

- measure of quality f (cost function)
- get training samples $x_1, \ldots, x_n$ from p
- choose a model $\mathcal{F}$ with parameters θ
- minimize $E_p[f(\theta, x)]$

Outline

The basics

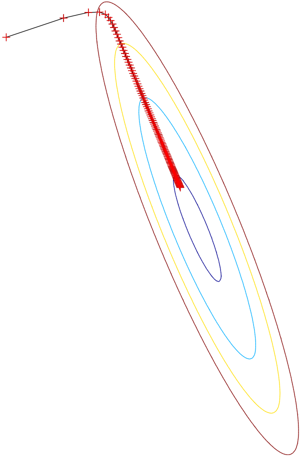
Taylor expansion of f to the first order:

$$f(\theta + \varepsilon) = f(\theta) + \varepsilon^T \nabla_\theta f + o(\|\varepsilon\|)$$

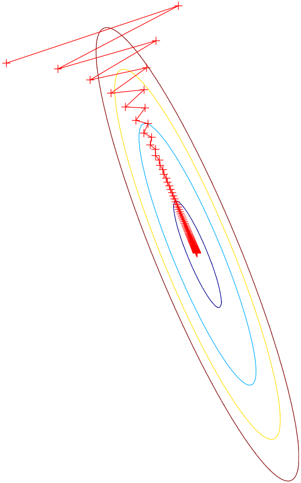
Best improvement obtained with

$$\varepsilon = -\eta \nabla_\theta f \qquad \eta > 0$$

Quadratic bowl

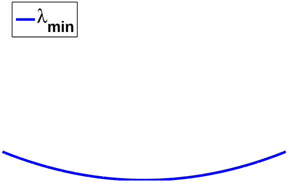


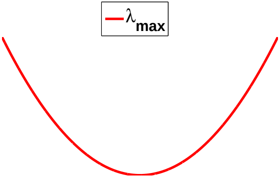
$\eta = .1$



$\eta = .3$

Optimal learning rate





$$\eta_{\text{min,opt}} = \frac{1}{\lambda_{\text{min}}}$$
$$\eta_{\text{min,div}} = \frac{2}{\lambda_{\text{min}}}$$

$$\eta_{\text{max,opt}} = \frac{1}{\lambda_{\text{max}}}$$
$$\eta_{\text{max,div}} = \frac{2}{\lambda_{\text{max}}}$$

$$\eta = \eta_{\text{max,opt}}$$
$$\kappa = \frac{\eta}{\eta_{\text{min,opt}}} = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}$$

$$T(\rho) = O\left(d\kappa \log \frac{1}{\rho}\right)$$

Limitations of gradient descent

- Speed of convergence highly dependent on κ
- Not invariant to linear transformations

$$\begin{aligned} \theta' &= k\theta \\ f(\theta' + \varepsilon) &= f(\theta') + \varepsilon^T \nabla_{\theta'} f + o(\|\varepsilon\|) \\ \varepsilon &= -\eta \nabla_{\theta'} f \end{aligned}$$

But $\nabla_{\theta'} f = \frac{\nabla_\theta f}{k}$!

Newton method

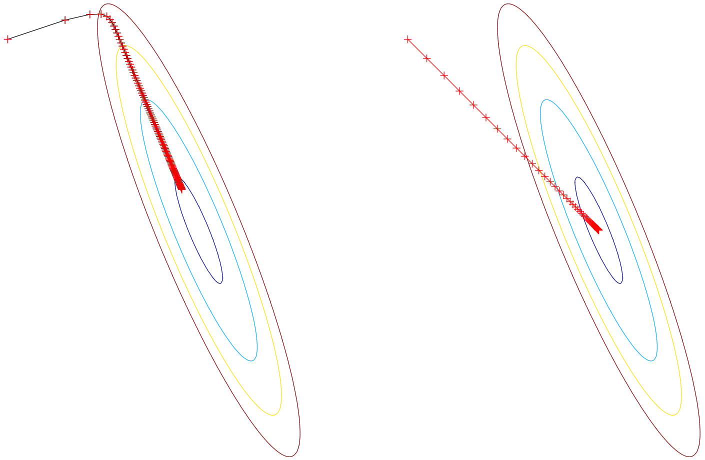
Second-order Taylor expansion of f around θ:

$$f(\theta + \varepsilon) = f(\theta) + \varepsilon^T \nabla_\theta f + \frac{\varepsilon^T H \varepsilon}{2} + o(\|\varepsilon\|^2)$$

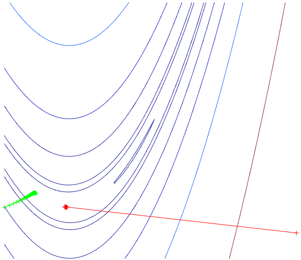
Best improvement obtained with

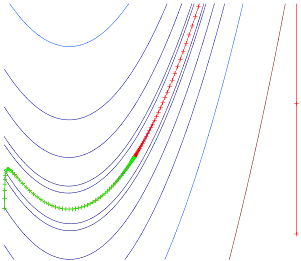
$$\varepsilon = -\eta H^{-1} \nabla_\theta f$$

Quadratic bowl



Rosenbrock function





Properties of Newton method

- Newton method assumes the function is locally quadratic (*Beyond Newton method*, Minka)
- H must be positive definite
- storing H and finding ε are in d^2

$$T(\rho) = O\left(d^2 \log\log \frac{1}{\rho}\right)$$

Limitations of Newton method

- Newton method looks powerful

but...

Limitations of Newton method

- Newton method looks powerful

but...

- H may be hard to compute
- it is expensive

Gauss-Newton

Only works when f is a sum of squared residuals!

$$
\begin{aligned}
f(\theta) &= \frac{1}{2}\sum_i r_i^2(\theta) \\
\frac{\partial f}{\partial \theta} &= \sum_i r_i \frac{\partial r_i}{\partial \theta} \\
\frac{\partial^2 f}{\partial \theta^2} &= \sum_i \left[r_i \frac{\partial^2 r_i}{\partial \theta^2} + \left(\frac{\partial r_i}{\partial \theta}\right)\left(\frac{\partial r_i}{\partial \theta}\right)^T \right] \\
\theta &= \theta - \eta \left(\sum_i \left(\frac{\partial r_i}{\partial \theta}\right)\left(\frac{\partial r_i}{\partial \theta}\right)^T \right)^{-1} \frac{\partial f}{\partial \theta}
\end{aligned}
$$

Properties of Gauss-Newton

$$\theta = \theta - \eta \left(\sum_i \left(\frac{\partial r_i}{\partial \theta} \right) \left(\frac{\partial r_i}{\partial \theta} \right)^T \right)^{-1} \frac{\partial f}{\partial \theta}$$

Discarded term: $r_i \frac{\partial^2 r_i}{\partial \theta^2}$

- Does not require the computation of H
- Only valid close to the optimum where $r_i = 0$
- Computation cost in $O(d^2)$

Levenberg-Marquardt

$$\theta = \theta - \eta \left(\sum_i \left(\frac{\partial r_i}{\partial \theta} \right) \left(\frac{\partial r_i}{\partial \theta} \right)^T + \lambda I \right)^{-1} \frac{\partial f}{\partial \theta}$$

- “Damped” Gauss-Newton
- Intermediate between Gauss-Newton and Steepest Descent
- Slower optimization but more robust
- Cost in $O(d^2)$

Quasi-Newton methods

- Gauss-Newton and Levenberg-Marquardt can only be used in special cases
- What about the general case?

Quasi-Newton methods

- Gauss-Newton and Levenberg-Marquardt can only be used in special cases
- What about the general case?
- H characterizes the change in gradient when moving in parameter space
- Let's find a matrix which does the same!

BFGS

We look for a matrix B such that

$$\nabla_\theta f(\theta + \varepsilon) - \nabla_\theta f(\theta) = B_t^{-1}\varepsilon$$: Secant equation

BFGS

We look for a matrix B such that

$$\nabla_\theta f(\theta + \varepsilon) - \nabla_\theta f(\theta) = B_t^{-1}\varepsilon$$: Secant equation

- Problem: this is underconstrained
- Solution: set additional constraints: small $\|B_{t+1} - B_t\|_W$

BFGS - (2)

- $B_0 = I$
- while not converged:
  - $p_t = -B_t \nabla f(\theta_t)$
  - $\eta_t = linemin(f, \theta_t, p_t)$
  - $s_t = \eta_t p_t$ (change in parameter space)
  - $\theta_{t+1} = \theta_t + s_t$
  - $y_t = \nabla f(\theta_{t+1}) - \nabla f(\theta_t)$ (change in gradient space)
  - $\rho_t = (s_t^T y_t)^{-1}$
  - $B_{t+1} = (I - \rho_t s_t y_t^T) B_t (I - \rho_t y_t s_t^T) + \rho_t s_t s_t^T$ (stems from Sherman-Morrisson formula)

BFGS - (3)

- Requires a line search
- No matrix inversion required
- Update in $O(d^2)$
- Can we do better than $O(d^2)$?

L-BFGS

- Low-rank estimate of B
- Based on the last m moves in parameters and gradient spaces
- Cost $O(md)$ per update
- Same ballpark as steepest descent!

Conjugate Gradient

We want to solve

$$Ax = b$$

- Relies on conjugate directions
- u and v are conjugate if $u^T A v = 0$.
- d mutually conjugate directions form a base of R^d
- Goal: to move along conjugate directions close to steepest descent directions

Nonlinear Conjugate Gradient

- Extension to non-quadratic functions
- Requires a line search in every direction (Important for conjugacy!)
- Various direction updates (Polak-Ribiere)

Going from batch to stochastic

- dataset composed of n samples
- $f(\theta) = \frac{1}{n} \sum_i f_i(\theta, x_i)$

Do we really need to see all the examples before making a parameter update?

Stochastic optimization

- Information is redundant amongst samples
- We can afford more frequent, noisier updates
- But problems arise...

Stochastic (Bottou)

Advantage

- much faster convergence on large redundant datasets

Disadvantages

- Keeps bouncing around unless η is reduced
- Extremely hard to reach high accuracy
- Theoretical definitions for convergence not as well defined
- Most second-orders methods will not work

Batch (Bottou)

Advantage

- Guaranteed convergence to a local minimum under simple conditions
- Lots of tricks to speed up the learning

Disadvantage

- Painfully slow on large problems

Problems arising in stochastic setting

- First order descent: $O(dn) \Longrightarrow O(dn)$
- Second order methods: $O(d^2 + dn) \Longrightarrow O(d^2n)$
- Special cases: algorithms requiring line search
  - BFGS: not critical, may be replaced by a one-step update
  - Conjugate Gradient: critical, no stochastic version

Successful stochastic methods

- Stochastic gradient descent
- Online BFGS (Schraudolph, 2007)
- Online L-BFGS (Schraudolph, 2007)

Conclusions of the tutorial

Batch methods

- Second-order methods have much faster convergence
- They are too expensive when d is large
- Except for L-BFGS and Nonlinear Conjugate Gradient

Conclusions of the tutorial

Stochastic methods

- Much faster updates
- Terrible convergence rates
  - Stochastic Gradient Descent: $T(\rho) = O\left(\frac{d}{\rho}\right)$
  - Second-order Stochastic Descent: $T(\rho) = O\left(\frac{d^2}{\rho}\right)$

Outline

Learning

- Measure of quality f ("cost function")
- Get training samples $x_1, \ldots, x_n$ from p
- Choose a model $\mathcal{F}$ with parameters θ
- Minimize $E_p[f(\theta, x)]$
- Time budget T

Small-scale vs. Large-scale Learning

- Small-scale learning problem: the active budget constraint is the number of examples n.
- Large-scale learning problem: the active budget constraint is the computing time T.

Which algorithm should we use?

$T(\rho)$ for various algorithms:

- Gradient Descent: $T(\rho) = O\left(nd\kappa \log \frac{1}{\rho}\right)$
- Second Order Gradient Descent: $T(\rho) = O\left(d^2 \log \log \frac{1}{\rho}\right)$
- Stochastic Gradient Descent: $T(\rho) = O\left(\frac{d}{\rho}\right)$
- Second-order Stochastic Descent: $T(\rho) = O\left(\frac{d^2}{\rho}\right)$

Second Order Gradient Descent seems a good choice!

Large-scale learning

- We are limited by the time T
- We can choose between ρ and n
- Better optimization means fewer examples

Generalization error

$$
\begin{aligned}
E(\tilde{f}_n) - E(f^*) &= E(f_F^*) - E(f^*) \text{ Approximation error} \\
&+ E(f_n) - E(f_F^*) \text{ Estimation error} \\
&+ E(\tilde{f}_n) - E(f_n) \text{ Optimization error}
\end{aligned}
$$

There is no need to optimize thoroughly if we cannot process enough data points!

Which algorithm should we use?

Time to reach $E(\tilde{f}_n) - E(f^*) < \epsilon$:

- Gradient Descent: $O\left(\frac{d^2\kappa}{\epsilon^{1/\alpha}} \log^2 \frac{1}{\epsilon}\right)$
- Second Order Gradient Descent: $O\left(\frac{d^2\kappa}{\epsilon^{1/\alpha}} \log \frac{1}{\epsilon} \log\log \frac{1}{\epsilon}\right)$
- Stochastic Gradient Descent: $O\left(\frac{d\kappa^2}{\epsilon}\right)$
- Second-order Stochastic Descent: $O\left(\frac{d^2}{\epsilon}\right)$

with $\frac{1}{2} \leq \alpha \leq 1$ (statistical estimation rate).

In a nutshell

- Simple stochastic gradient descent is extremely efficient
- Fast second-order stochastic gradient descent can win us a constant factor
- Are there other possible factors of improvement?

What we did not talk about (yet)

- we have access to $x_1, \ldots, x_n \sim p$
- we wish to minimize $E_{x \sim p}[f(\theta, x)]$
- can we use the uncertainty in the dataset to get information about p?

Outline

Cost functions

Empirical cost function

$$f(\theta) = \frac{1}{n}\sum_i f(\theta, x_i)$$

True cost function

$$f^*(\theta) = E_{x\sim p}[f(\theta, x)]$$

Gradients

Empirical gradient

$$g = \frac{1}{n} \sum_i \nabla_\theta f(\theta, x_i)$$

True gradient

$$g^* = E_{x \sim p}[\nabla_\theta f(\theta, x)]$$

Central-limit theorem:

$$g | g^* \sim \mathcal{N}\left(g^*, \frac{C}{n}\right)$$

Posterior over g^*

Using

$$g^* \sim \mathcal{N}(0, \sigma^2 I)$$

we have

$$g^*|g \sim \mathcal{N}\left(\left(I + \frac{C}{n\sigma^2}\right)^{-1} g, \left(\frac{I}{\sigma^2} + nC^{-1}\right)^{-1}\right)$$

and then

$$\varepsilon^T g^*|g \sim \mathcal{N}\left(\varepsilon^T \left(I + \frac{C}{n\sigma^2}\right)^{-1} g, \varepsilon^T \left(\frac{I}{\sigma^2} + nC^{-1}\right)^{-1} \varepsilon\right)$$

Aggressive strategy

$$\varepsilon^T g^* | g \sim \mathcal{N}\left(\varepsilon^T \left(I + \frac{C}{n\sigma^2}\right)^{-1} g, \varepsilon^T \left(\frac{I}{\sigma^2} + nC^{-1}\right)^{-1} \varepsilon\right)$$

- we want to minimize $E_{g^*}[\varepsilon^T g^*]$
- Solution:

$$\varepsilon = -\eta \left(I + \frac{C}{n\sigma^2}\right)^{-1} g$$

Aggressive strategy

$$\varepsilon^T g^* | g \sim \mathcal{N}\left(\varepsilon^T \left(I + \frac{C}{n\sigma^2}\right)^{-1} g, \varepsilon^T \left(\frac{I}{\sigma^2} + nC^{-1}\right)^{-1} \varepsilon\right)$$

- we want to minimize $E_{g^*}[\varepsilon^T g^*]$
- Solution:

$$\varepsilon = -\eta \left(I + \frac{C}{n\sigma^2}\right)^{-1} g$$

This is the regularized natural gradient.

Conservative strategy

$$\varepsilon^T g^* | g \sim \mathcal{N}\left(\varepsilon^T \left(I + \frac{C}{\sigma^2}\right)^{-1} g, \varepsilon^T \left(\frac{I}{\sigma^2} + nC^{-1}\right)^{-1} \varepsilon\right)$$

- we want to minimize $Pr(\varepsilon^T g^* > 0)$
- Solution:

$$\varepsilon = -\eta \left(\frac{C}{n}\right)^{-1} g$$

Conservative strategy

$$\varepsilon^T g^* | g \sim \mathcal{N}\left(\varepsilon^T \left(I + \frac{C}{\sigma^2}\right)^{-1} g, \varepsilon^T \left(\frac{I}{\sigma^2} + nC^{-1}\right)^{-1} \varepsilon\right)$$

- we want to minimize $Pr(\varepsilon^T g^* > 0)$
- Solution:

$$\varepsilon = -\eta \left(\frac{C}{n}\right)^{-1} g$$

This is the natural gradient.

Online Natural Gradient

- Natural gradient has nice properties
- Its cost per iteration is in d^2
- Can we make it faster?

Goals

$$\varepsilon = -\eta \left(I + \frac{C}{n\sigma^2} \right)^{-1} g$$

We must be able to update and invert C whenever a new gradient arrives.

Plan of action

Updating (uncentered) C:

$$C_t \propto \gamma C_{t-1} + (1-\gamma) g_t g_t^T$$

Plan of action

Updating (uncentered) C:

$$C_t \propto \gamma C_{t-1} + (1-\gamma) g_t g_t^T$$

Inverting C:

- maintain low-rank estimates of C using its eigendecomposition.

Eigendecomposition of C

- Computing k eigenvectors of $C = GG^T$ is $O(kd^2)$
- But computing k eigenvectors of G^TG is $O(kp^2)$!
- Still too expensive to compute for each new sample (and p must not grow)

Eigendecomposition of C

- Computing k eigenvectors of $C = GG^T$ is $O(kd^2)$
- But computing k eigenvectors of $G^T G$ is $O(kp^2)$!
- Still too expensive to compute for each new sample (and p must not grow)
- Done only every b steps

To prove I'm not cheating

- eigendecomposition every b steps

$$
\begin{aligned}
C_t &= \gamma^t C + \sum_{k=1}^{t} \gamma^{t-k} g_k g_k^T + \lambda t \qquad t = 1, \ldots, b \\
v_t &= C_t^{-1} g_t \\
X_t &= \left[\gamma^{\frac{t}{2}} U \quad \gamma^{\frac{t-1}{2}} g_1 \quad \ldots \quad \gamma^{\frac{1}{2}} g_{t-1} \quad g_t \right] \\
C_t &= X_t X_t^T + \lambda t \qquad v_t = X_t \alpha_t \qquad g_t = X_t y_t \\
\alpha_t &= (X_t^T X_t + \lambda t)^{-1} y_t \\
v_t &= X_t (X_t^T X_t + \lambda t)^{-1} y_t
\end{aligned}
$$

Computation complexity

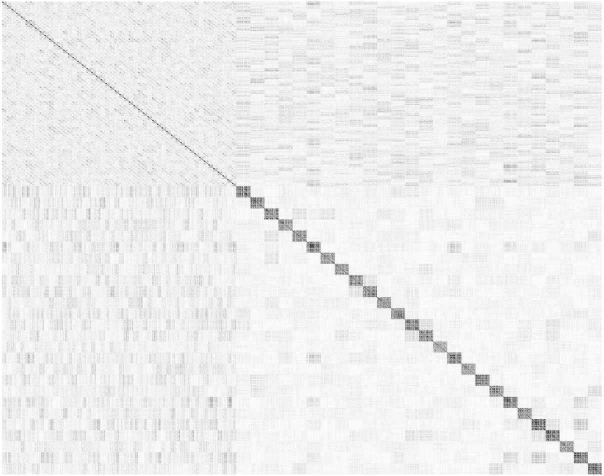
- d dimensions
- k eigenvectors
- b steps between two updates

- Computing the eigendecomposition (every b steps): $k(k+b)^2$
- Computing the natural gradient: $d(k+b)+(k+b)^2$
- if $p \ll (k+b)$, cost per example is $O(d(k+b))$

What next

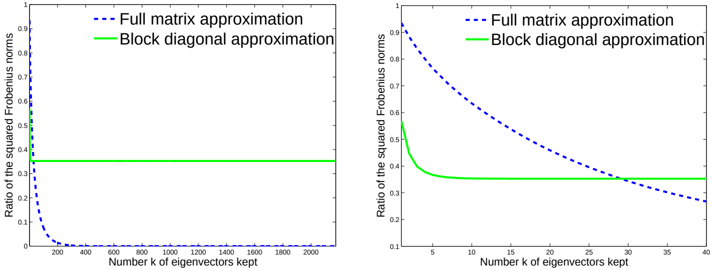
- Complexity in $O(d(k+b))$
- We need a small k
- If $d > 10^6$, how large should k be?

Decomposing C



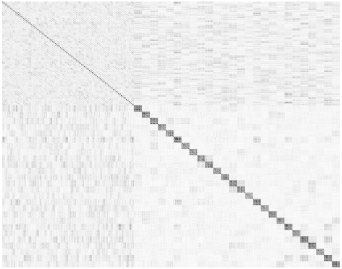
C is almost block-diagonal!

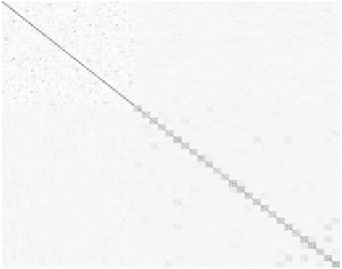
Quality of approximation



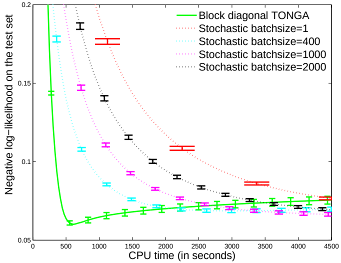
Minimum with $k = 5$

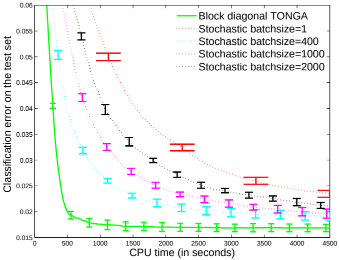
Evolution of C



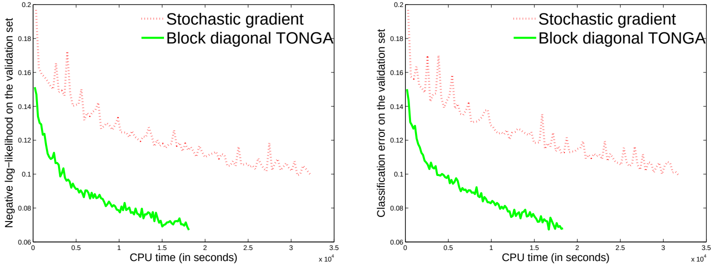


Results - MNIST

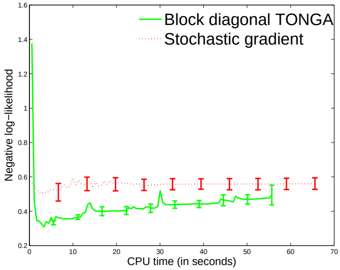


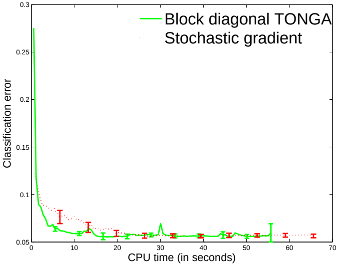


Results - Rectangles



Results - USPS





TONGA - Conclusion

- Introducing uncertainty speeds up training
- There exists a fast implementation of the online natural gradient

Difference between C and H

- H accounts for small changes in the parameter space
- C accounts for small changes in the input space
- C is not just a “cheap cousin” of H

Future research

Much more to do!

- Can we combine the effects of H and C?
- Are there better approximations of C and H?
- Anything you can think of!

Thank you!

Questions?