Approximate Second Order Methods

Sargur N. Srihari
srihari@cedar.buffalo.edu

Topics in Optimization for Deep Models

- Importance of Optimization in machine learning
- How learning differs from optimization
- Challenges in neural network optimization
- Basic Optimization Algorithms
- Parameter initialization strategies
- Algorithms with adaptive learning rates
- Approximate second-order methods
- Optimization strategies and meta-algorithms

Topics in Second Order Methods

1. Overview
2. Newton's Method
3. Conjugate Gradients
   – Nonlinear Conjugate Gradients
4. BFGS
   – Limited Memory BFGS

Overview

- We discuss here second order methods of training deep networks
- The only objective function examined is empirical risk:
  - Empirical risk, with m training examples, is

$$\boxed{J(\boldsymbol{\theta}) = E_{(\boldsymbol{x},y)\sim\hat{p}_{data}}\Big(L(f(\boldsymbol{x};\boldsymbol{\theta}),y)\Big) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),y^{(i)})}$$

$f(\boldsymbol{x}\,;\boldsymbol{\theta})$ is the predicted output when the input is $\boldsymbol{x}$
y is target output
L is the per-example loss function

  - Methods extend readily to other objective functions such as those that include parameter regularization

Newton's Method

- In contrast to first order gradient methods, second order methods make use of second derivatives to improve optimization
- Most widely used second order method is Newton's method
- It is described in more detail here emphasizing neural network training
- It is based on Taylor's series expansion to approximate $J(\theta)$ near some point θ_0 ignoring derivatives of higher order

Newton Update Rule

- Taylor's series to approximate $J(\theta)$ near θ_0

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\theta - \theta_0)^T \nabla_\theta J(\boldsymbol{\theta}_0) + \frac{1}{2}(\theta - \theta_0)^T H(\theta - \theta_0)$$

  - where H is the Hessian of J wrt θ evaluated at θ_0
- Solving for the critical point of this function we obtain the Newton parameter update rule

$$\theta^* = \theta_0 - H^{-1}\nabla_\theta J(\theta_0)$$

  – Thus for a quadratic function (with positive definite H) by rescaling the gradient by H^{-1} Newton's method directly jumps to the minimum

  – If objective function is convex but not quadratic (there are higher-order terms) this update can be iterated yielding the training algorithm given next

Training Algorithm associated with Newton's Method

Algorithm: Newton's method with objective:

$$\boxed{J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})}$$

Require: Initial parameter $\boldsymbol{\theta}_0$

Require: Training set of m examples

while stopping criterion not met **do**

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Compute Hessian: $\boldsymbol{H} \leftarrow \frac{1}{m}\nabla^2_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Compute Hessian inverse: $\boldsymbol{H}^{-1}$

Compute update: $\Delta\boldsymbol{\theta} = -\boldsymbol{H}^{-1}\boldsymbol{g}$

Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

Positive Definite Hessian

- For surfaces that are not quadratic, as long as the Hessian remains positive definite, Newton's method can be applied iteratively
- This implies a two-step procedure:
  - First update or compute the inverse Hessian (by updating the qudratic approximation)
  - Second, update the parameters according to

$$\boxed{\theta^* = \theta_0 - H^{-1}\nabla_\theta J(\theta_0)}$$

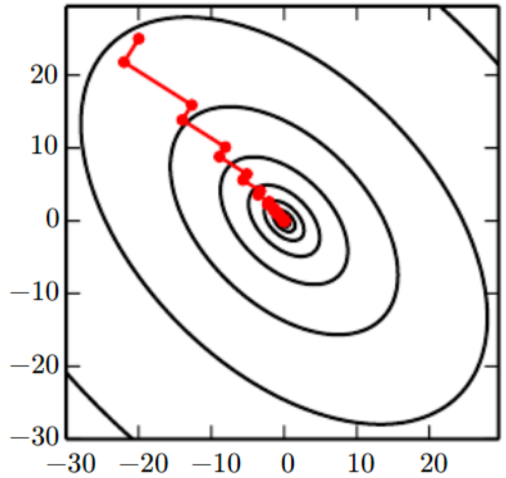
Regularizing the Hessian

- Newton's method is appropriate only when the Hessian is positive definite
  - In deep learning the surface of the objective function is nonconvex
    - Many saddle points: problematic for Newton's method
- Can be avoided by regularizing the Hessian
  - Adding a constant α along the Hessian diagonal

$$\theta^* = \theta_0 - \left[H(f(\theta_0)) + \alpha I\right]^{-1} \nabla_\theta f(\theta_0)$$

Motivating Conjugate Gradients

- Method to efficiently avoid calculating H^{-1}
  - By iteratively descending conjugate directions
- Arises from steepest descent for quadratic bowl has an ineffective zig-zag pattern
  - Since each line direction is orthogonal to previous
    - Let previous search direction be $\boldsymbol{d}_t$
    - Then $\nabla_\theta J(\theta)\boldsymbol{d}_{t-1} = 0$
    - Current search direction will have no contribution in direction $\boldsymbol{d}_{t-1}$
    - Thus $\boldsymbol{d}_t$ is orthogonal to $\boldsymbol{d}_{t-1}$
    - Method of conjugate gradients addresses this problem



Imposing Conjugate Directions

- We seek to find a search direction that is conjugate to the previous line search direction
- At iteration t the next search direction $\boldsymbol{d}_t$ takes the form $\boldsymbol{d}_t = \nabla_\theta J(\theta) + \beta_t \boldsymbol{d}_{t-1}$
- Directions $\boldsymbol{d}_t$ and $\boldsymbol{d}_{t\text{-}1}$ are conjugate if $\boldsymbol{d}_t H \boldsymbol{d}_{t\text{-}1} = 0$
- Methods for imposing conjugacy
  - Fletcher-Reeves $\boxed{\beta_t = \dfrac{\nabla_\theta J(\theta_t)^T \nabla_\theta J(\theta_t)}{\nabla_\theta J(\theta_{t-1})^T \nabla_\theta J(\theta_{t-1})}}$
  - Polak-Ribiere $\boxed{\beta_t = \dfrac{(\nabla_\theta J(\theta_t) - \nabla_\theta J(\theta_{t-1}))^T \nabla_\theta J(\theta_t)}{\nabla_\theta J(\theta_{t-1})^T \nabla_\theta J(\theta_{t-1})}}$

Conjugate gradient algorithm

Algorithm The conjugate gradient method

Require: Initial parameters $\boldsymbol{\theta}_0$
Require: Training set of m examples
  Initialize $\boldsymbol{\rho}_0 = \mathbf{0}$
  Initialize $g_0 = 0$
  Initialize $t = 1$
  while stopping criterion not met **do**
    Initialize the gradient $\boldsymbol{g}_t = \mathbf{0}$
    Compute gradient: $\boldsymbol{g}_t \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute $\beta_t = \frac{(\boldsymbol{g}_t - \boldsymbol{g}_{t-1})^\top \boldsymbol{g}_t}{\boldsymbol{g}_{t-1}^\top \boldsymbol{g}_{t-1}}$ (Polak-Ribière)
    (Nonlinear conjugate gradient: optionally reset β_t to zero, for example if t is a multiple of some constant k, such as $k = 5$)
    Compute search direction: $\boldsymbol{\rho}_t = -\boldsymbol{g}_t + \beta_t \boldsymbol{\rho}_{t-1}$
    Perform line search to find: $\epsilon^* = \text{argmin}_\epsilon \frac{1}{m} \sum_{i=1}^m L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_t + \epsilon \boldsymbol{\rho}_t), \boldsymbol{y}^{(i)})$
    (On a truly quadratic cost function, analytically solve for ϵ^* rather than explicitly searching for it)
    Apply update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{\rho}_t$
    $t \leftarrow t + 1$
  end while

The BFGS Algorithm

- Broyden-Fletcher-Goldfarb-Shanno (BFGS)
  - Newton's method without the computational burden
  - It is similar to the conjugate gradient method
    - More direct approach to approximating Newton's update
- Recall Newton's update: $\boxed{\theta^* = \theta_0 - H^{-1}\nabla_\theta J(\theta_0)}$
    - where H is the Hessian of J wrt θ evaluated at θ_0
  - Primary difficulty is computation of H^{-1}
  - BFGS is quasi Newton: approximates H^{-1} by matrix M_t that is iteratively refined by low-rank updates
    - Once the inverse Hessian M_t is updated, the direction of descent ρ_t is determined by $\rho_t = M_t g_t$
    - Final update to parameters is $\theta_{t+1} = \theta_t + \varepsilon^* \rho_t$