John Lambert blog About

Gauss-Newton Optimization in 10 Minutes

Mar 31, 2018

Unconstrained Optimization

The Gauss-Newton Method

Suppose our residual is no longer affine, but rather nonlinear. We want to minimize $\|r(x)\|^2$. Generally speaking, we cannot solve this problem, but rather can use good heuristics to find local minima.

- Start from initial guess for your solution
- Repeat:
- (1) Linearize $r(x)$ around current guess $x^{(k)}$. This can be accomplished by using a Taylor Series and Calculus (Standard Gauss-Newton), or one can use a least-squares fit to the line.
- (2) Solve least squares for linearized objective, get $x^{(k+1)}$.

The linearized residual $r(x)$ will resemble:

$$r(x) \approx r(x^{(k)}) + Dr(x^{(k)})(x - x^{(k)})$$

where Dr is the Jacobian, meaning $(Dr)_{ij} = \frac{\partial r_i}{\partial x_j}$

Distributing the rightmost product, we obtain

$$r(x) \approx Dr(x^{(k)})x - \left(Dr(x^{(k)})(x^{(k)}) - r(x^{(k)})\right)$$

With a single variable x, we can re-write the above equation as

$$r(x) \approx A^{(k)}x - b^{(k)}$$

Levenberg-Marquardt Algorithm (Trust-Region Gauss-Newton Method)

In Levenberg-Marquardt, we have add a term to the objective function to emphasize that we should not move so far from $\theta^{(k)}$ that we cannot trust the affine approximation. We often refer to this concept as remaining within a "trust region" (TRPO is named after the same concept). Thus, we wish $||\theta - \theta^{(k)}||^2$ to be small. Our new objective is:

$$||A^{(k)}\theta - b^{(k)}||^2 + \lambda^{(k)}||\theta - \theta^{(k)}||^2$$

This objective can be written inside a single ℓ_2-norm, instead using two separate ℓ_2-norms:

$$|| \begin{bmatrix} A^{(k)} \\ \sqrt{\lambda^{(k)}} I \end{bmatrix} \theta - \begin{bmatrix} b^{(k)} \\ \sqrt{\lambda^{(k)}}\theta^{(k)} \end{bmatrix} ||^2$$

Suppose we have some input data x and labels y. Our prediction function could be

$$\hat{f} = x^T\theta_1 + \theta_2$$

Suppose at inference time we use $f(x) = \text{sign}\left(\hat{f}(x)\right)$, where $\text{sign}(a) = +1$ for $a \geq 0$ and -1 for $a < 0$. At training time, we use its smooth (and differentiable) approximation, the hyperbolic tangent, tanh:

$$\phi(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

```
phi = @(x) (exp(x)-exp(-x))./(exp(x)+exp(-x));
```

The gradient of tanh: $\nabla_x \tanh = 1 - \tanh(x)^2$. We call this ϕ' in code:

```
phiprime = @(x) (1-phi(x).^2);
```

Suppose our objective function is the MSE loss, with a regularization term:

$$J = \sum_{i=1}^{N} \left(y_i - \phi\left(x_i^T \theta_1 + \theta_2\right) \right)^2 + \mu ||\theta_1||^2$$

The residual for a single training example i is r_i is

$$y_i - \phi\left(x_i^T \theta_1 + \theta_2\right)$$

For a vector of training examples $\mathbf{X}$ and labels $\mathbf{Y}$, our nonlinear residual function is:

```
r = @(x,y,t1,t2) y-phi(x'*t1+t2);
```

To linearize the residual, we compute its Jacobian $Dr(\theta_1, \theta_2)$ via matrix calculus:

$$\frac{\partial r_i}{\partial \theta_1} = -\phi'(x_i^T \theta_1 + \theta_2) x_i^T$$

```
jacobian_0_entr = -phiprime(X(:,i)'*theta(1:400)+theta(end))* X(:,i)'
```

$\frac{\partial r_i}{\partial \theta_2} = -\phi'(x_i^T \theta_1 + \theta_2)$

```
jacobian_1_entr = -phiprime(X(:,i)'*theta(1:400)+theta(end))
```

The the full Jacobian evaluated at a certain point X_i is just these stacked individual entries:

```
Dr = zeros(length(labels_train),length(theta));
for i = 1:length(labels_train)
        Dr(i,:) = [jacobian_0_entr, jacobian_1_entr];
end
```

Let $\theta = \begin{bmatrix} \theta_1^T & \theta_2 \end{bmatrix}^T \in \mathbb{R}^{401}$. The linearized residual follows the exact form outlined in the Gauss-Newton section above:

$$r(\theta) \approx A^{(k)} \theta - b^{(k)}$$

where

$$b^{(k)} = A^{(k)}\theta^{(k)} - r\left(\theta^{(k)}\right)$$

In code, this term is computed as:

```
A_k_temp = Dr; % computed above
b_k_temp = Dr*theta - r(X, Y,theta(1:400),theta(end));
```

We solve a least-squares problem in every iteration, with a 3-part objective function (penalizing the residual, large step sizes, and also large θ_1-norm weights):

$$||\begin{bmatrix} A^{(k)} \\ \sqrt{\lambda^{(k)}} I_{401} \\ \begin{bmatrix} \sqrt{\mu} I_{400} & 0 \end{bmatrix} \end{bmatrix} \theta - \begin{bmatrix} b^{(k)} \\ \sqrt{\lambda^{(k)}}\theta^{(k)} \\ 0 \end{bmatrix}||^2.$$

We represent the left term by

```
A_k = [A_k_temp; sqrt(lambda(itr))*eye(length(theta)); sqrt(mu)*[eye(lengt
```

and the right term by

```
b_k = [b_k_temp; sqrt(lambda(itr))*theta; zeros(length(theta)-1,1)];
```

We solve for the next iterate of θ with the pseudo-inverse:

```
theta_temp = A_k\b_k;
```

The full algorithm might resemble:

```
mu = 10; %regularization coefficient
lambda(1) = 1; %initial lambda for Levenberg-Marquardt

theta = ones(401,1); %initial value for theta (last entry is theta_2)

for itr = 1:15
        %calculate Jacobian at the current iteration (see code above)

        %linearize the objective function (see code above)

        % stopping condition ...
```

```
end
```

John Lambert blog

johnwlambert
none

Notes on Machine Learning and Optimization.