Author: Dr. Shi Guo Email: guoshi1984@hotmail.com

1 Gradient Based Optimization Method

1.1 Practical use of gradient descent: Dealing with large samples

Batch gradient descent vs. stochastic gradient descent vs. mini batch gradient descent
a. Definition
1) **Batch gradient**
Batch gradient means using all the data point to calculate the gradient.
$cost = \sum_{i=1}^{N}$ -loglikelihood of ith sample
$grad = \frac{\partial(cost)}{\partial \mathbf{w}}$
update all parameter based on gradient

2) **Stochastic gradient descent**
The cost function used in batch gradient descent uses is the summation over all the data points. In stochastic descent the cost function we use only contains one data point, we use one data point to update parameters, iterate over all data points.
For m = 1 : N
cost = - loglikelihood of the ith sample
$grad = \frac{\partial(cost)}{\partial \mathbf{w}}$
update all parameter based on gradient

3) **Mini-Batch gradient descent**
We divide N samples into G = N/k groups so that each group contains k data points
For n = 1 : G
cost = C $\sum_{(n-1)k}^{nk}$ loglikelihood
grad = $\frac{\partial(cost)}{\partial \mathbf{w}}$
update all parameter based on gradient

b. Comparison

	Time per iteration	Convergence time for large data	Sensitivity to parameters	Smoothness
Batch Gradient	Slow for large data	Slower	Moderate	Smooth
Stochastic Gradient	Always fast	Faster	High	Very noisy

c. Practical usage
Shuffle the data before running the stochastic gradient descent

1.2 Adam Optimization

(1) Weighted avarage of gradient
A common pratice to avoid value fluctuations during the gradient descent update is using a weighted average. For each step, we average the gradients in the previous steps and do the update. If the cost function is $f(w)$ and its gradient is $g(w) = \nabla f(w)$, We define at step t

$$m^{(t)} = \beta m^{(t-1)} + (1-\beta)g^{(t)}$$
$$w^t = w^{(t-1)} - \alpha m^{(t)}$$

Where beta is a hyperparameter. We usually choose 0.9 or 0.99. Using this weighted average, the gradient we use for update at each step is

$$m^{(0)} = 0$$
$$m^{(1)} = \beta m^{(0)} + (1-\beta)g^{(1)} = (1-\beta)g^{(1)}$$
$$m^{(2)} = \beta m^{(1)} + (1-\beta)g^{(2)} = \beta(1-\beta)g^{(1)} + (1-\beta)g^{(2)}$$
$$m^{(3)} = \beta m^{(2)} + (1-\beta)g^{(3)} = \beta^2(1-\beta)g^{(1)} + \beta(1-\beta)g^{(2)} + (1-\beta)g^{(3)}$$

We can write above as

$$m^{(t)} = (1-\beta)\sum_{i=0}^{t} \beta^{t-i} g^{(i)}$$

(2) First step bias correction
The above weighted average causes a bias on the first step. Because the term $M^{(0)}$ is not defined and we arbitrarily set to zero. This leads to a bias on the first term, so we correct $m^{(t)}$ using $\tilde{m}^{(t)}$

$$\tilde{m}^{(t)} = \frac{m_t}{1-\beta^t} = \frac{1}{1-\beta^t}(\beta m^{(t-1)} + (1-\beta)g^t)$$

$$\tilde{m}^{(0)} = m^0 = 0$$
$$\tilde{m}^{(1)} = \frac{m^{(1)}}{1-\beta} = \frac{\beta}{1-\beta}m^{(0)} + g^{(1)} = g^{(1)}$$
$$\tilde{m}^{(2)} = \frac{m^{(2)}}{1-\beta^2} = \frac{\beta}{1-\beta^2}m^{(1)} + \frac{1-\beta}{1-\beta^2}g^{(2)} = \frac{1}{1-\beta^2}(\beta(1-\beta)g^{(1)} + (1-\beta)g^{(2)})$$
$$\tilde{m}^{(3)} = \frac{m^{(3)}}{1-\beta^3} = \frac{\beta}{1-\beta^3}m^{(2)} + \frac{(1-\beta)}{1-\beta^3}g^{(3)} = \frac{1}{1-\beta^3}(\beta^2(1-\beta)g^{(1)} + \beta(1-\beta)g^{(2)} + (1-\beta)g^{(3)})$$

$$\tilde{m}^{(t)} = \frac{1-\beta}{1-\beta^t}\sum_{i=0}^{t} \beta^{t-i} g^{(i)}$$

From above, we see that $m^{(1)} = g^{(1)}$, so gradient in the first iteration does not have any bias. Also under this correction, for any t, the sum of coefficients of $g^{(i)}$ is 1.

$$\frac{1-\beta}{1-\beta^t}\sum_{i=0}^{t}\beta^i = \frac{1-\beta}{1-\beta^t}\frac{1-\beta^t}{1-\beta} = 1$$

(3) Learning rate scaling
So far we have a constant learning rate α. This means during one step of update, the change of w is large when the gradient is large and the value of the parameter would also fluctuate. To fix this, we modify the gradient by dividing its magnitute. Similarly to $m^{(t)}$, we define

$$\begin{aligned} v^{(t)} &= \beta_v v^{(t-1)} + (1-\beta_v) g_t^2 \\ \tilde{v}^{(t)} &= \frac{v^{(t)}}{1-\beta_v^t} \\ w^{(t)} &= w^{(t-1)} - \alpha \frac{\tilde{m}^{(t)}}{\sqrt{\tilde{v}^{(t)}} + \epsilon} \end{aligned}$$

Where the ϵ is a small positive number in order to prevent dividing by zero. (4) Summary

$$\begin{aligned} g^{(t)} &= \nabla_w f(w) \\ m^{(t)} &= \beta m^{(t-1)} + (1-\beta) g^{(t)} \\ v^{(t)} &= \beta_v v^{(t-1)} + (1-\beta_v) g_t^2 \\ \tilde{m}^{(t)} &= \frac{m_t}{1-\beta^t} \tilde{v}^{(t)} = \frac{v^{(t)}}{1-\beta_v^t} \\ w^{(t)} &= w^{(t-1)} - \alpha \frac{\tilde{m}^{(t)}}{\sqrt{\tilde{v}^{(t)}} + \epsilon} \end{aligned}$$

1.3 Conjugate Gradient Method

a. The idea of A orthogonality
There exists tremendous materials online explaining conjugate gradient method. However, after reading many versions of explanations, I am still confused that why letting moving directions conjugate to each other eventually leads to a solution. So I will be explaining the intuition first such that the idea of CONJUGATE comes more natural to understand.
1)**The quadratic form and its gradient.** Let us start with a quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A \mathbf{x}$$

Let

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$

and

$$\mathbf{x} = (x_1, x_2)^T$$

Then

$$\begin{aligned} f(x_1, x_2) &= \frac{1}{2} \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ &= \frac{1}{2}(2x_1^2 + x_2^2) \end{aligned}$$

The gradient of $f(\mathbf{x})$ is

$$\begin{aligned} \nabla f(x_1, x_2) &= (\frac{\partial}{\partial x_1} f(x_1, x_2), \frac{\partial}{\partial x_2} f(x_1, x_2)) \\ &= (2x_1, x_2) \\ &= \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ &= A\mathbf{x} \end{aligned}$$

2)**Minimization using gradient descent.** Suppose we arbitrarily choose a starting point $(x^{(0)}, y^{(0)}) = (1, 1)$. Based on the principle of gradient descent, we move point 0 to point 1 along the opposite direction of the gradient. Namely, the direction we move should be

$$\mathbf{d} = A\mathbf{x} = (-2x_1, -x_2)^T = (-2, -1)^T$$

After finding the direction, we need to determine the step α that we need to move, our next point is

$$\begin{aligned} (x_1^{(1)}, x_2^{(1)})^T &= (x_1^{(0)}, x_2^{(0)})^T + \alpha \mathbf{d}^{(0)} \\ &= (x_1^{(0)}, x_2^{(0)})^T + \alpha(-2, -1)^T \\ &= (1 - 2\alpha, 1 - \alpha) \\ f(x_1^{(1)}, x_2^{(1)}) &= 2(1 - 2\alpha)^2 + (1 - \alpha)^2 \\ &= 9\alpha^2 - 10\alpha + 3 \end{aligned}$$

We choose α such that α minimizes $f(x_1^{(1)}, x_2^{(1)})$ and we achieve this by setting $\frac{\partial f}{\partial \alpha} = 0$.

$$\frac{\partial f}{\partial \alpha} = 18\alpha - 10 = 0$$

We get $\alpha = \frac{5}{9}$, and the corresponding $(x_1^{(1)}, x_2^{(1)})$ is

$$\begin{aligned} (x_1^{(1)}, x_2^{(1)}) &= (x_1^{(0)}, x_2^{(0)}) + \alpha \hat{\mathbf{d}}^{(0)} \\ x_1^{(1)} &= 1 + \frac{5}{9} \times (-2) = -\frac{1}{9} \\ x_2^{(1)} &= 1 - \frac{5}{9} = \frac{4}{9} \end{aligned}$$

A-orthogonality Since we can easily know the minimum point of $f(x_1, x_2)$ is (0,0) without doing any calculation. We can calculate the error term which gives us how far we are still off the minimum point. We define the error term as

$$e^{(1)} = (0,0) - (x_1^{(1)}, x_2^{(1)}) = (\frac{1}{9}, -\frac{4}{9}).$$

Last, let us work on an interest fact by look at a matrix multiplication

$$\begin{aligned}&\mathbf{d^{(0)}} A\mathbf{e^{(1)}}\\ &= \begin{pmatrix} -2 & -1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{9} \\ -\frac{4}{9} \end{pmatrix} = 0\end{aligned}$$

Orthogonal!!! This means the error term and moving direction are A-orthogonal. The fact of orthogonality holds if the $f(\mathbf{x})$ has more than two variables. This is a very interesting point. But why?

b. Proof of A-orthogonality

Now we suppose the $f(\mathbf{x}) = f(x_1, x_2, ..., x_i, ..., x_N)$ take a more general quadratic form

$$\begin{aligned}f(\mathbf{x}) &= \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x} + c\\ &= \frac{1}{2}(\sum_i a_{ii}x_i^2 + 2\sum_{i,j,i<j} a_{ij}x_i x_j) - \sum_i b_i x_i + c\end{aligned}$$

Now taking the derivative,

$$\begin{aligned}&\frac{\partial f(\mathbf{x})}{\partial x_k}\\ =&a_{kk}x_k + \sum_{j\neq k} a_{kj}x_j - b_k\\ =&\sum_j a_{kj}x_j - b_k\end{aligned}$$

The last line is the same as the first row result of matrix Ax -b. So

$$\begin{aligned}&\begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ ... \\ \frac{\partial f(\mathbf{x})}{\partial x_N} \end{pmatrix}\\ =&\begin{pmatrix} \sum_j a_{1j}x_j - b_1 \\ \sum_j a_{2j}x_j - b_2 \\ ... \\ \sum_j a_{Nj}x_j - b_N \end{pmatrix}\\ =&Ax - b\end{aligned}$$

We choose $f(\mathbf{x}^{(1)})$ such that

$$\frac{\partial f(x^{(1)})}{\partial \alpha} = 0$$

So,

$$\frac{\partial f(x^{(1)})}{\partial \alpha}$$
$$= \sum_i \frac{\partial f(x_i^{(1)})}{\partial x_i^{(1)}} \frac{\partial x_i^{(1)}}{\partial \alpha} \qquad = \nabla f(\mathbf{x}^{(1)}) \cdot d^0 = (Ax^{(1)} - b) \cdot d^0$$

While we know that

$$Ae^{(1)} = A(x^{(min)} - x^{(1)}) = b - Ax^{(1)}$$

This leads to

$$d^{(0)^T} Ae^{(1)} = 0$$

So by choosing α that minimize $f(\mathbf{x^{(1)}})$, we automatically guarantee that the moving direction is A-orthogonal to the error term. Proof completed.

c. Steps of convergence

In the example we did in the intuition, the $f(\mathbf{x})$ only has two variables. And we have proved that after moving one step, the error term is A-orthogonal to the moving direction. So if we can enforce our next moving direction is A-orthogonal to the current one, how many steps does the process converge? The answer is by enforcing each moving directions are A-orthogonal to each other, the algorithm can converge using exact n step, where n is the number of variables.

Proof:

We express the error term $e^{(0)}$ at step 0 as a linear combinations of n searching directions

$$e^{(0)} = \sum_{j=0}^{n-1} \delta_j d^{(j)}$$

Where δ_j is a scalar. We find δ_j by multiplying $d^{(k)^T} A$ on both sides

$$d^{(k)^T} Ae^{(0)} = \sum_{j=0}^{n-1} \delta_j d^{(k)^T} Ad^{(j)}$$
$$d^{(k)^T} Ae^{(0)} = \delta_k d^{(k)^T} Ad^{(k)}$$

$$\begin{aligned}\delta_k =& \frac{{d^{(k)}}^T A e^{(0)}}{{d^{(k)}}^T A d^{(k)}} \\ =& \frac{{d^{(k)}}^T (A e^{(k)} + \sum_{i=0}^{k-1} \alpha_i d^{(i)})}{{d^{(k)}}^T A d^{(k)}} \\ & ({d^{(k)}}^T d^{(i)} = 0) \\ =& \frac{{d^{(k)}}^T A e^{(k)}}{{d^{(k)}}^T A d^{(k)}}\end{aligned}$$

If we let

$$\alpha_i = \delta_i$$

So

$$e^{(k)} = e^{(0)} - \sum_{i=0}^{k-1} \alpha_i d^{(i)}$$

$$= \sum_{i=0}^{n-1} \delta_i d^{(i)} - \sum_{i=0}^{k-1} \alpha_i d^{(i)} = \sum_{j=k}^{n-1} \delta_i d^{(i)}$$

We see that when i = n, $e^{(n)} = 0$. So we reach the convergence after exact n steps.

d. algorithm

For minimizing

$$f(\mathbf{x}) = \mathbf{x^T A x} - \mathbf{2 b^T x}$$

$$\begin{aligned}x^{(i)} &= x^{(i-1)} + \alpha^{(i-1)} d^{(i-1)} \\ r^{(i)} &= r^{(i-1)} - \alpha^{(i-1)} A d^{(i-1)} \\ d^{(i)} &= r^{(i)} + \beta^{(i-1)} d^{(i-1)}\end{aligned}$$

at each step

$$\alpha^{(i)} = \frac{{r^{(i)}}^T r^{(i)}}{{d^{(i)}}^T A d^{(i)}}$$

$$\beta^i = \frac{{r^{(i+1)}}^T r^{(i+1)}}{{r^{(i)}}^T r^{(i)}}$$

2 Hession Based Method

2.1 Newton Method

a. Newton Method Principles

Based on Taylor's expansion if we are at x_0, we try to find δx so that $x_0 + \delta x$

is closer to the stationary point.

$$f(x_0 + \delta x) = f(x_0) + f^{'}(x_0)\delta x + f^{''}(x_0)(\delta x)^2$$

take the derivative

$$df(x_0 + \delta x)/dx = f^{'}(x_0) + f^{''}(x_0)\delta x$$

therefore

$$\delta x = -\frac{f^{'}(x_0)}{f^{''}(x_0)}$$
$$X^{(t+1)} = X^t - \frac{f^{'}(x_0)}{f^{''}(x_0)}$$

b. Matrix Forms

$$x^{(t+1)} = x^t - H^{-1}(f(x^t))\nabla f(x^t)$$

where H is the Hessian matrix.

c. Connection with Gradient descent The newton method can be reduce to gradient descent method by taking Hessian matrix as Identity matrix

d. Pros Since it utilizes the second order derivative, it converges much faster than gradient descent.
For quadratic function, the equation from the Taylor expansion is exact, therefore the stationary point can be found using only one step.

e. Cons Need to evaluate the inverse of the Hessian Matrix, so it is computationally expensive.

2.2 Quasi Newton

Newton method requires the inverse of the Hessian matrix, which is usually not easy to solve. So we need to find an approximation of the Hessian. Similar to the way we solve for gradient, we can use finite difference method, in which the gradient is

$$gradf(x) = \frac{f(x + \delta x) - f(x)}{\delta x},$$

This is only exact when δx approaches zero. For 2nd order derivative, we can write

$$f^{'}(x) = \frac{f^{'}(x + \delta) - f^{'}(x)}{\delta}$$

Again this is only exact when δ is zero. Based on this idea we replace the Hessian Matrix with an approximation that satisfies the following approximation

$$\nabla f(x + \delta x) = \nabla f(x) + B\delta x$$

This is quasi newton method. Various Quasi Newton methods exist with different choice of B.

3 Levenburg Marquadt

This Method adds a scaled Identity matrix uI to the Hessian, for large u and small Hessian, the method is equivalent to gradient descent with step size 1/u.