OPTIMIZATION TECHNIQUES

A PREPRINT

Kushagra Gupta (Mentor)
Department of Mathematics and Statistics
IIT Kanpur
kushgpt@iitk.ac.in

Mayuri Gedam
Department of Mechanical Engineering
IIT Kanpur
mvgedam@iitk.ac.in

Ayush Jain
Department of Electrical Engineering
IIT Kanpur
ayushj@iitk.ac.in

Ankur Banga
Department of Electrical Engineering
IIT Kanpur
ankurb@iitk.ac.in

Nitin Garg
Department of Mathematics and Statistics
IIT Kanpur
nitingrg@iitk.ac.in

Dhruvil Sangani
Department of Mathematics and Statistics
IIT Kanpur
vikrant@iitk.ac.in

Vikrant Malik
Department of Mechanical Engineering
IIT Kanpur
vikrant@iitk.ac.in

K Nikita
Department of Computer Science and Engineering
IIT Kanpur
nikitak@iitk.ac.in

Kartavya Jain
Department of Physics
IIT Kanpur
kartavya@iitk.ac.in

June 7, 2019

1 Three Parameter Weibull Distribution

We were supposed to maximize the likelihood function for the weibull distribution. The Ideal value of three parameters β, γ, η was estimated using a python script that effectively equated the partial derivative of MLE to 0.

MLE:

$$N\log(\beta) - N\beta\log(\eta) + (\beta-1)\sum_{i=1}^{n}\ln(x) - \sum_{i=1}^{n}(\frac{x}{\eta})^{\beta} \tag{1}$$

If γ approaches minimum value of sample

$$(\beta-1)\log(min(t)-\gamma) \to \infty \tag{2}$$

$$\frac{\partial f}{\partial \beta} = 0 \qquad \frac{\partial f}{\partial \eta} = 0 \tag{3}$$

$$\eta = [\frac{1}{N}\sum x^{\beta}]^{\frac{1}{\beta}} \tag{4}$$

Complete Documentation and Scripts can be found at :

`https://github.com/Ayushjain9501/convexOptim/tree/master/Assignments/Assignment%201`

2 Batch Gradient Descent

Batch Gradient Descent computes the gradient of the cost function w.r.t. the parameters for the entire training set. The update can be represented as follows:

$$\theta = \theta - \eta * \Delta_{\theta} J(\theta) \tag{5}$$

This algorithm can be very slow as it iterates over the whole dataset just to perform a single update. It also can not update our model online where new examples are being added. Batch gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

It can be visualized as:

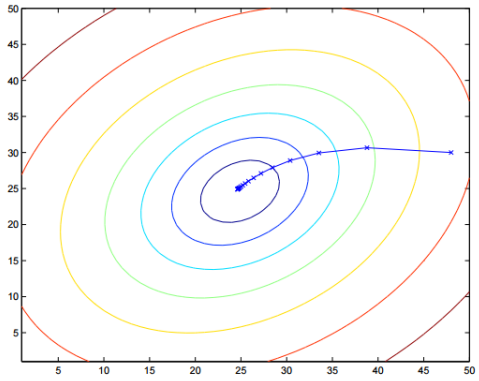

Figure 1: Batch Gradient Descent

Below is an implementation of this algorithm:

`https://github.com/nitingarg1000/Optimization/tree/master/Batch_Gradient_Descent`

3 Stochastic Gradient Descent

Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online.

We try to fit a straight line to some randomly generated dataset using SGD.

Parameter update for a randomly selected training example (x,y):

$$\theta = \theta - \eta * \Delta_{\theta} J(\theta, x, y) \tag{6}$$

Complete Documentation and Scripts can be found at :

`https://github.com/Ayushjain9501/convexOptim/tree/master/Assignments/stochasticGD`

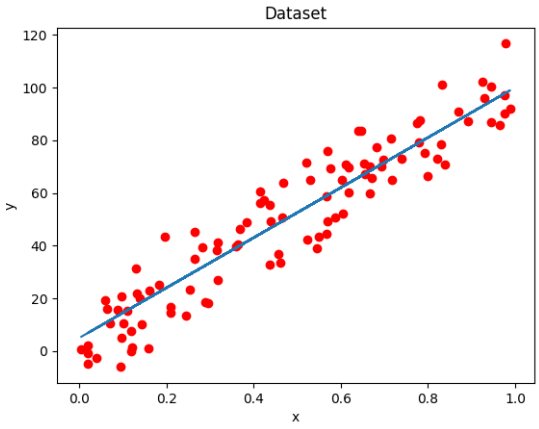

Figure 2: Final Output

4 Mini Batch Gradient Descent

Mini-batch gradient descent performs an update for every mini-batch of n training examples. The update can be represented as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{7}$$

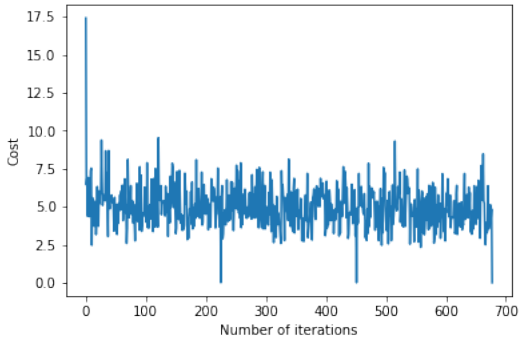

Figure 3: Final Output

This reduces the variance of the parameter updates, which can lead to more stable convergence. Common mini-batch sizes range between 50 and 256, but can vary for different applications. Mini-batch gradient descent is typically the algorithm of choice when training a neural network and the term SGD usually is employed also when mini-batches are used. However it does not guarantee good convergence.

Below is an implementation of this algorithm:

`https://github.com/V1krant/Optimisation-Techhniques.git`

5 Matrix Factorization

We find the solution of Bx=b simply by calculating x=B-1b, but sometimes calculating B-1 is cumbersome or when B is nearly singular. So, we use factorization or decomposition methods to break B into easy to use matrices. Some of the factorisation methods are discussed below... —-dhruvil

5.1 LU and PLU Factorization for basis B

It's simply the gaussian elimination method that we have learnt already in 12th class and also in mth102. We convert B matrix to upper triangular matrix U by performing a series of operations. We multiply a series of matrix to obtain the same. Name those matrices as Gi and permutations Pi after every operation.

(Gr Pr). . . (G2 P2)(G1 P1)B = U

Multiplying this matrices on either side of the equation Bx=b will give Ux = b' Where b'=(Gr Pr). . . (G2 P2)(G1 P1)b If no permutations are performed, Gr . . . .Gi is lower triangular, and denoting its (lower triangular) inverse as L, we have the factored form B = LU for B. Also, if PT is a permutation matrix that is used to a priori rearrange the rows of B and we then apply the Gaussian triangularization operation to derive L-1PTB = U, we can write B = (PT)-1LU = PLU, noting that PT = P-1. Hence, It's also known as PLU decomposition.

5.2 QR and QRP Factorization for a Basis B

This factorization is most suitable and is used frequently for solving dense equation systems. Here the matrix B is reduced to an upper triangular form R by premultiplying it with a sequence of square, symmetric orthogonal matrices Qi. Bi-1 = Qi-1 ....Q1B The matrix Qi is a square, symmetric orthogonal matrix of the form Qi = 1 – yi qi qiT, where qi = (0, ..., 0, qii ,...,qni)' and yi E RT are suitably chosen to perform the foregoing operation. Defining Q = Qn-1. . . ...Q1 , we see that Q is also a symmetric orthogonal matrix and that QB = R, or that B = QR, since Q = QT = Q-1 that is, Q is an involutory matrix. Now, to solve Bx = b, we equivalently solve QRx = b or Rx = Qb by finding b' = Qb first and then solving the upper triangular system Rx = b' via back-substitution. Note that since ||Qv|| = ||v|| for any vector v, we have ||R|| = ||QR|| = ||B||, so that R preserves the relative magnitudes of the elements in B, maintaining stability. This is its principal advantage.

5.3 Cholesky Factorization LLT and LDLT for Symmetric, Positive Definite Matrices B

This method is only applicable for square symmetric matrix B. We will represent this matrix as B=L*LT, where L is a Lower-triangular matrix. So that L*LT becomes a symmetric matrix and then by comparing the elements of both the symmetric matrices, we could figure out the elements of L. The equation system Bx=b, can now be solved via LT(Lx)=b, through the solution of two triangular systems of equations. We first find y to satisfy Ly = b and then compute x via the system LTx = y. Sometimes we write B = LDLT for the sake of accuracy as it avoids calculating the square root which otherwise was to be calculated in finding the elements of B.

References

[1] George Kour and Raid Saabne. Real-time segmentation of on-line handwritten arabic script. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 417–422. IEEE, 2014.

[2] George Kour and Raid Saabne. Fast classification of handwritten on-line arabic characters. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 312–318. IEEE, 2014.

[3] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.

[4] MOKHTAR S. BAZARAA HANIF D. SHERALI C. M. SHETTY . Non-linear programming theory and algorithms. pages 756-759