International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Adaptation to Best Fit Learning Rate in Batch Gradient Descent

Mohan Pandey[1], B. L. Pal[2]

[1]IntegreatedM.Tech. Scholar, Mewar University, Rajasthan, India

[2]Assistant Professor, Mewar University, Rajasthan, India

Abstract: ***The efficient training of a supervised machine learning system is commonly viewed as minimization of a cost function that depends on the parameters of the proposed hypothesis. This perspective gives some advantages to the development of better training algorithm, as the problem of minimization of a function is well known in many fields. Typically many learning system use gradient descent to minimize the cost function but has a problem of slow convergence and exploding on choosing learning rate beyond a threshold. We often make mistake on choosing learning rate as there is no known generic way of choosing learning rate and we need to do it manually. In this paper a method for the adaptation of learning rate is presented and also solving the problem of slow convergence and exploding of the algorithm. The main feature of proposed algorithm is that it speeds up the convergence rate and does not affect the result or number of epochs requires to converge on changing the learning rate.***

Keywords: Batch training, On-line training, Epoch, Learning Rate, Gradient Descent, Hypothesis, Cost Function.

1. Introduction

In the field of machine learning many a times we encounter with the learning problem in which we have given data set to algorithms which have "right answers" and the aim is to develop a model such that it can fit the given set of data according to given "right answers" and this domain is famously known as "supervised machine learning (SML) [1]". In this research field Linear Regression [2], Logistic Regression [3], Neural Networks[4], Back Propagation Neural Networks(BPNNs)[4, 5] are the most popular models. The efficient supervised training of these models is a subject of considerable ongoing research and many algorithms have been proposed to this end. In SML we have actual output vector which are the "right answers" in the given data set and an desired output vector(supervised learning) a common training approach is to minimize the network learning error which is the difference between the actual output vector and desired output vector. The rapid computation of a set of weights that minimize the error function is quite a difficult task since, generally the number of input feature(input vector) is high so as to number of weights to be computed is also high and the error function(which is to minimize) generates a complicated surface in weight space, possessing many local minima and having broad flat regions adjoined to narrow steep ones that need to be searched to locate an "optimal" weight set.

Broadly, SML is divided into two categories: stochastic (also called on–line) [6,7] and batch (also called off–line) [8] learning. On-line training may be chosen either because of the very large data set(or may be redundant) training set or may be want to model a time-variant(slowly) system. On one hand Batch training is fast for small training set and on the other we have on-line which is fast on large set of data helps escaping local minima and provides more natural way to for learning non-stationary task [9].

Batch supervised learning is classical approach in machine learning: a set of examples is obtained and train the model with these examples before exposing it to the real problem set, on other hand in on-line learning, data gathered in several operation of the system is continuously fed to the model and used to adapt the learned function [10,11,12,13]. Batch training trained the model(i.e. finding the weights) by minimizing the given set of data, so it can be viewed as the minimization of an cost function J; that is to find

Θ^*(set of weight parameter)=(Θ_1^*, Θ_2^*, Θ_3^* ..... Θ_n^*) R^n , such that:

$$\Theta^* = \min J(\Theta)$$

Where, Θ is the set of weight parameter of the proposed hypothesis. J is the cost function defined as the sum-of-squared differences error function.

$$J(\Theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\Theta(x^i) - y^{(i)})^2 \quad (1)$$

Note that, m is the total number of training examples, $h_\Theta(x)$ is the hypothesis function, that need to be trained and y is the output vector.

Although, it may seems that on-line training [14] is better and modern technique for large data, but batch training is quite faster for small set of data which is nearly acceptable to most of the problem in machine learning, this makes it a very important training model to study on.

In this paper we focused on the adaptation of the best fit learning parameter of batch gradient descent.

2. Pure Gradient Descent (Fixed Learning Rate)

Outline on working of Gradient descent [15]:

- Start with some set of Θ.
- Keep changing Θ, to reduce $J(\Theta)$ until we hopefully end up at a minimum.

Fig.1 shows a cost function $J(\Theta_0, \Theta_1)$, with two parameters Θ_0, Θ_1, our aim here is to minimize the cost function, and this is done by updating the value of Θ_0 and Θ_1.There are two main issues (i) how to update the Θ_0, Θ_1 and (ii) the step length towards the minima

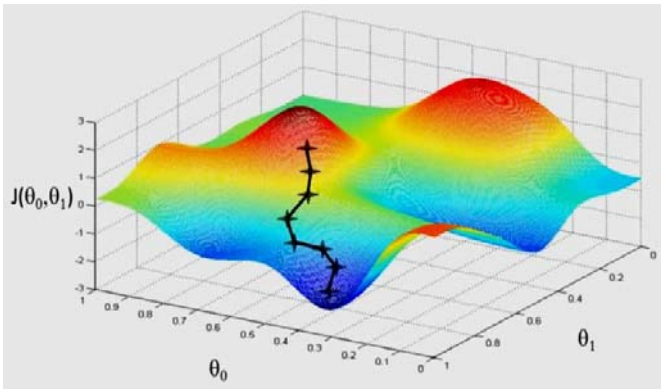

Figure 1: Reaching Minima

For resolving first issue we use partial derivative of cost function over Θ_0, Θ_1 and we use a parameter "learning rate(α)" to decide the "steplength" the α will decide how fast or slow the function will converge [16] to the minima.

We use the following algorithm for gradient descent:

. repeat until convergence {

$$\Theta_j = \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta) \; for \; all \; j \; (1 \ldots n)$$

}

Simultaneous update:

$$temp0 = \Theta_0 - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta) \; for \; \mathbf{all} \; j \; (1 \ldots n)$$

$$temp1 = \Theta_1 - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta) \; for \; \mathbf{all} \; j \; (1 \ldots n)$$

.
.
.

$$tempn = \Theta_n - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta) \; for \; \mathbf{all} \; j \; (1 \ldots n)$$

$$\Theta_0 = temp0$$
$$\Theta_1 = temp1$$

.
.

$$\Theta_n = tempn$$

Here, α is the learning rate, $\alpha \frac{\partial}{\partial \theta_j} J(\theta^*)$ is the partial derivative of cost function $J(\Theta_0, \Theta_1)$.

α decides the rate at which function will converge and partial derivative gives us the update term through which Θ^* is updated.

There is enormous research work is going on to find various methods to adapt the learning rate for on-line training ([1,3,17]), and there are other approaches also present for on-line training such as Conjugate gradient, Quasi-Newton, using peak value for learning rate, adaptation based on the evolution of error are some of the examples. Given the inherent efficiency of stochastic gradient descent, various schemes have been proposed recently [14, 10, 11, 17, 18]. As, most of the research is going on online-training, in this paper we approaches for batch gradient learning rate adaptation and try to improve the performance of gradient descent.

2.1 Fixed Learning Rate

Batch gradient descent usually focuses on fixed learning rate adaptation (also known as pure gradient descent) following strategy is mostly used: select a range based on previous experience, say

$$0.01 \leq \alpha \leq 2 \; (2)$$

This method proceed by selecting a initial value for α, running gradient descent and observing the cost function, and adjusting the learning rate accordingly.

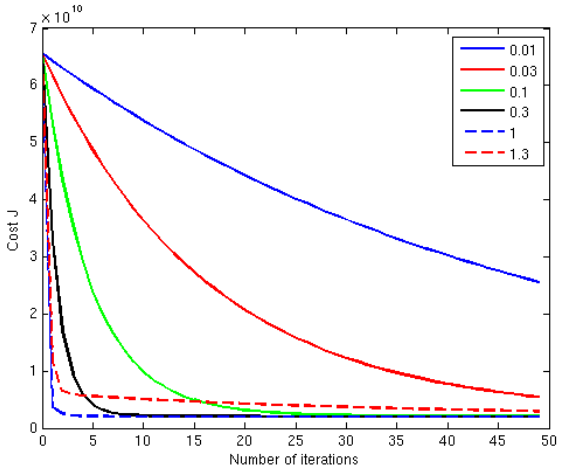

Figure 2: Comparing different learning rate

Notice that for a small alpha like 0.01, the cost function decreases slowly, which means slow convergence during gradient descent. Also, notice that while alpha=1.3 is the largest learning rate, alpha=1.0 has a faster convergence. This shows that after a certain point, increasing the learning rate will no longer increase the speed of convergence.

In fact, if your learning rate is too large, gradient descent will not converge at all, and cost function might blow up like in the following graph for alpha = 1.4

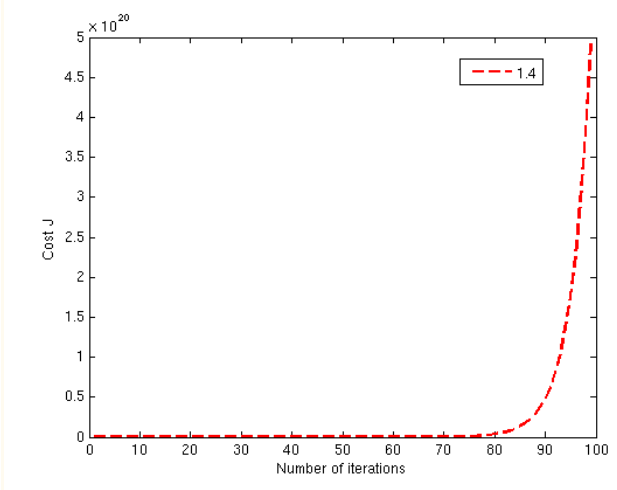

Figure 3: Gradient Descent Explodes

First we select a small value for learning rate, and it takes nearly 350 epochs to converge than on increasing the learning rate number of epochs also decreases but, after $\alpha = 1.4$ convergence is not taking place at all and gradient descent is exploded. So, using pure gradient descent for converging function is not recommended as we can analyze that its very problematic and tedious work to find a optimal value for learning rate just by selecting a range and try to find which suits best. In the following section we will explore a new method to automatically find the optimal value for learning rate.

3. Adaptive Learning Algorithm

Our main focus in learning rate algorithms is to find the set of parameters that can fit the proposed model by converging the cost function to a minimum value and thus applying the parameters to new data and predict the result. We often want to minimize our cost function in less number of epochs. As, pure gradient descent is not practical to use we propose our algorithm that can gain the benefit of local gain adaptation.

3.1 Description

Pure gradient descent uses a fixed learning rate for converging the cost function, the updation step consist derivative of cost function and learning rate as the step length, on selecting smaller step length it will take many epochs to converge, but we can also select a larger step length as it will not affect the end result also it converge in much less number of epochs, on other hand selecting larger step length (learning rate) cost function may overshoot and not converge at all. As, there is no mathematical way to find the optimal learning rate that can fit a given model. During gradient descent we could use large learning rate when we start the algorithm but require small learning rate when we are close to minimum value of cost function. In the proposed algorithm we were using the previous state of system and compare it with the current state to determine the learning rate.

The adaptation rule work as follows: On every epoch we update the parameters of the model and we calculate the cost of the state, then we compare the previous cost with the current cost, if the previous cost is larger than the current cost we increase the learning rate by 10% to15% to accelerate convergence, if the previous cost is smaller than the current cost, which indicates that previous update was too big, so we decrease the learning rate by 50% and if the both cost are same we will not affect the learning rate. By, increasing learning rate we get larger step length and we converge in less number of epoch which is very crucial.

$$\alpha = \begin{cases} \alpha + 0.1\alpha \,, if\ J(previous) > J(current) \\ \alpha - 0.5\alpha \,, if\ J(previous) < J(current) \\ \alpha \,, if\ J(previous) = J(current) \end{cases} (3)$$

Through our proposed algorithm we could converge cost function in much less epochs then it will take in pure gradient descent, thus improving the performance of the algorithm.

3.2 Algorithm

The following pseudo-code fragment depicts the kernel of the proposed algorithm learning and adaptation process.

Number of epochs : e{

if(e > 1){

if (J(e-1) > J(e)){

$\alpha = \alpha + (10 * \alpha)/100$

}

else if (J(e-1) < J(e)){

$\alpha = \alpha - (50 * \alpha)/100$

}

else{

$\alpha = \alpha$

}

}

}

The core reason for the success of proposed algorithm lies in the concept of the 'direct adaptation' of the cost of previous and current system states. As, we work only on the sign of the cost not on magnitude thus it spreads the learning in the whole network.

4. Experimental Result

We tested our modified algorithm with pure gradient descent. We, implemented linear regression to predict the price of a house according to its size and number of bedroom. We already have data for previous sales. With the help of data we predict the price according to the size of house and number of bedroom, for predicting price of house we define a hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \ (4)$$

and the cost function as:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \ (5)$$

We, first use pure gradient descent and then our modified algorithm, we compare both algorithm on the basis of number of epochs required to minimize the cost function for different learning rate. Below, table shows the comparison.

Table 1: Comparing Algorithms

Learning Rate(α)	Number of epochs	
	Pure Gradient Descent	Modified Gradient Descent
0.01	300	50
0.03	120	40
0.1	40	20
0.3	15	10
1	7	10
1.3	200	5
1.4	Exploding(Not Converging)	7
3	Exploding(Not Converging)	8

As, we have found that changing learning rate can drastically alter the number of epoch required to minimize the cost function in case of "Pure Gradient Descent" but it not the case with "Modified Gradient Descent".

5. Advantages and Limitations

5.1 Advantages

- On changing the value of learning rate, number of epochs required to minimize the function does not change by many.
- It speeds-up the convergence of cost function.
- After a threshold value for learning rate, pure gradient descent will not converge at all and explode, but our modified version will minimize the cost function.

5.2 Limitations

- Learning rate cannot be negative i.e. learning rate must be greater than zero ($\alpha>0$).
- If we take learning rate very small (close to zero), then it will take large number of epochs but again very fast then pure gradient descent.

6. Conclusion

I want to conclude that the modified version of gradient descent can be used an all type of learning system and it increases the speed of convergence thus saving the precious time as most of the time we are unable to predict the correct hypothesis at a single shot, but we try the same hypothesis with different learning rate and hope for better result, but with the modified version we can be sure of the result at single shot that the hypothesis is correct or not and move to different hypothesis more confidently. Thus, this algorithm gives a more powerful tool to minimize the cost function, predicting the weight parameter and even with different values of learnt can be used an all type of learning system and it increases the speed of convergence thus saving the precious time as most of the time we are unable to predict the correct hypothesis at a single shot, but we try the same hypothesis with different learning rate and hope for better result, but with the modified version we can be sure of the result at single shot that the hypothesis is correct or not and move to different hypothesis more confidently. Thus, this algorithm gives a more powerful tool to minimize the cost function, predicting the weight parameter and even with different values of learning rate it converges the cost function with nearly same and less number of epochs.

References

[1] S. B. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques", Informatica, vol.31, no.3,pp. 249-268, July-2007.
[2] Linear regression. *en.Wikipedia.org*. [Online] 8 5 2014. [Cited: 8 5 2014.] http://en.wikipedia.org/wiki/Linear_regression.
[3] Friedman,Trevor Hastie, Robert Tibhirani, "Additive Logistic Regression: A Statistical View of Boosting", Annals of Statistics Stanford University, Vol. 28, No.2, pp. 237-407, 2007.
[4] Wei Wu, Jian Wang, Mingsong Cheng, Zhengxue Li, "Convergence analysis of online gradient method for BP neural networks", Elsevier, Neural Network,Vol.24, pp. 91-98, September 2010.
[5] L.-W. Chan, F. Fallside, "An adaptive training algorithm for back propagation networks", Computer Speech and Language, Vol. 2, pp. 205-218, 1987.
[6] N. Murata, K.-R. MÄuller, A. Ziehe, and S.-i. Amari, "Adaptive on-line learning in changing environments", Advances in Neural Information Processing Systems, M. C. Mozer, M. I. Jordan, and T. Petsche, vol. 9, pp. 599-605, The MIT Press, Cambridge, MA, 1997.
[7] Leon Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent",NEC Labs America, Princeton NJ 08542, USA.
[8] Y. LeCun, P. Y. Simard, and B. Pearl-mutter, "Automatic learning rate maximization in large adaptive machines", Advances in Neural Information Processing Systems, S. J. Hanson, J. D. Cowan, and C. L. Giles, vol. 5, pp. 156-163, Morgan Kaufmann, San Mateo, CA, 1993.
[9] BORDES. A., BOTTOU, L., and GALLINARI, P: SGD-QN: "Careful Quasi-Newton Stochastic Gradient Descent", Journal of Machine Learning Research, Vol. 10, pp. 1737-1754, 2009.
[10] D. Saad, "On–line Learning in Neural Networks", Cambridge University Press, Vol. 4, pp. 281-285, 1998.
[11] N.N. Schraudolph, "Online Local Gain Adaptation for Multilayer perceptrons, Technical Report", IDSIA–09–98, IDSIA, Lugano, Switzerland, 1998.
[12] F. M. Silva and L. B. Almeida, "Speeding up back-propagation", Advanced Neural Computers, R. Eckmiller, Elsevier, Amsterdam, pp. 151-158, 1990.
[13] T. Tollenaere, \SuperSAB: fast adaptive back propagation with good scaling properties", Neural Networks, Vol. 3, pp. 561-573, 1990.
[14] L.B. Almeida, T. Langlois, J.D. Amaral, A. Plankhov, "Parameter adaptation in Stochastic Optimization, In: On–line Learning in Neural Networks", D. Saad, Cambridge University Press, pp. 111–134, 1998.
[15] S Sra, S Nowozin, SJ Wright. Optimization for machine learning. *Optimization for machine learning*. s.l. : MIT, 2012.
[16] R.Jacobs, "Increased rates of convergence through learning rate adaptation", Neural Networks, Vol. 1, pp. 295-307, 1998.
[17] N.N. Schraudolph, "Local Gain Adaptation in Stochastic Gradient Descend, Technical Report", IDSIA–09–99, IDSIA, Lugano, Switzerland, 1999.

[18] R.S. Sutton, "Adapting Bias by Gradient Descent: an Incremental Version of Delta–Bar–Delta", Proc. of the Tenth National Conference on Artificial Intelligence, MIT Press, pp. 171–176, 1992.

Author Profile

Mohan Pandey is currently pursuing Masters Degree in Computer Science and Engineering in Mewar University, India.

B L Pal received the M.Tech. degree in SIT from School of Electronics, Devi Ahilya University, Indore in 2009 and worked in Reliance Big Entertainment for the year(2009). He is working with Mewar University as a Assistant Professor for five years and currently designated as HOD.