sciendo

Műszaki Tudományos Közlemények vol. 13. (2020) 174–177.
DOI: Magyar: https://doi.org/10.33895/mtk-2020.13.33
Angol: https://doi.org/10.33894/mtk-2020.13.33

EME MTSZ MTK

TANULÁSIRÁTA-MÓDSZER FULL-BATCH GRADIENS-TANULÁSRA

A LEARNING RATE METHOD FOR FULL-BATCH GRADIENT DESCENT

Asadi Soodabeh,[1] Vogel Manfred[2]

Institute for Data Science, University of Applied Sciences and Arts, Northwestern Switzerland

[1] *soodabeh.asadidezaki@fhnw.ch*

[2] *manfred.vogel@fhnw.ch*

Abstract

In this paper, we present a learning rate method for gradient descent using only first order information. This method requires no manual tuning of the learning rate. We applied this method on a linear neural network built from scratch, along with the full-batch gradient descent, where we calculated the gradients for the whole dataset to perform one parameter update. We tested the method on a moderate sized dataset of housing information and compared the result with that of the Adam optimizer used with a sequential neural network model from Keras. The comparison shows that our method finds the minimum in a much fewer number of epochs than does Adam.

Keywords*: machine learning, gradient descent, learning rate.*

Összefoglalás

Cikkünkben egy olyan tanulásiráta-módszert mutatunk be gradienstanulásra, amely kizárólag elsőrendű információkat használ fel. Ezen módszer esetében nem szükséges a tanulási ráta manuális beállítása. Az algoritmust alkalmaztuk egy nulláról felépített lineáris neurális hálóra full-batch gradiens-módszer esetén, mikor a gradienst a teljes adathalmazra kiszámoljuk egy paraméter-aktualizálási lépésben. A módszert egy közepes méretű, szállásinformációkkal kapcsolatos adathalmazon teszteltük, a kapott eredményeket pedig összevetettük a Keras-beli Adam algoritmus által szolgáltatottakkal egy szekvenciális neurális háló esetén. Az eredmények azt mutatják, hogy az Adam algoritmushoz képest a mi módszerünk sokkal kevesebb epoch alatt megtalálja a minimumot.

Kulcsszavak*: gépi tanulás, gradiens módszer, tanulási ráta.*

1. Introduction

Machine learning methods aim at updating a set of parameters W in order to optimize an objective function $f(W)$. They iteratively perform a procedure which applies changes to the parameters. Gradient Descent (GD) is one of the most popular and widely used algorithms for training machine learning models such as deep neural networks. GD attempts to optimize the objective function by following the steepest descent direction given by the negative of the gradient. There are many modifications to the GD algorithm. In fact, Keras [1] offers several other optimizers for deep learning which are actually improvements of the GD method. The GD algorithm requires the learning rate hyperparameter to be chosen. Setting the learning rate typically involves a tuning procedure. Usually, the highest possible learning rate is chosen manually. Choosing higher than this rate can cause the objective function to diverge. On the other hand, choosing it too low results in slow learning. Determining a good learning rate

could be more of an art than a science for many problems. There have been several attempts at estimating a good learning rate at each iteration of GD. These either try to speed up learning when suitable, far from the minima, or to slow down learning near a local minima. Learning rate schedules have been proposed [2] to automatically decrease the learning rate based on how many epochs through the data have been done. These approaches typically add additional hyperparameters to control how quickly the learning rate decays. One method to accelerate the training procedure is the *Momentum* method [3]. This can be considered as the simplest extension to GD. Momentum strategy is based on saving the past gradient information and using it in the current iteration to update the parameters. In [4], *Adagrad* is introduced and significantly good results are obtained on large scale learning. The update rule in Adagrad is

$$\Delta W^{(i)} = -\frac{\alpha}{\sqrt{\sum_{j=1}^{i} g^{(i)^2}}} \cdot g^{(i)} \qquad (1)$$

when we update the parameters as
$W^{(i+1)} = W^{(i)} + \Delta W^{(i)}$.
Here:
α - is a global learning rate,
$g(i)$ – is the gradient of the cost function with respect to the parameters.
i – superindex that indicates the i-th iteration.

While the manually tuned global learning rate α appears here, each iteration has its own specific rate that grows with the inverse of the gradient magnitude. Due to the continual accumulation of squared gradients in the denominator, proportional to the inverse of the gradient, the learning rate will continue to decrease throughout training, eventually decreasing to zero and stopping training completely. In [5], with *Adadelta* optimizer, this deficiency was removed.

In [6], the authors proposed Adam algorithm which is based on adaptive estimates of lower-order momentums. A variant of this algorithm which is based on the infinity norm is proposed as Adamax.

Here, we propose a suitable value for the learning rate at the current iteration (epoch for the full-batch GD) which is based on estimating the scalar function of the learning rate with a parabola and choosing the minimum of the parabola as the learning rate in the current iteration. Details of our method are explained in the next section.

2. Results

Suppose that $X \in \mathbb{R}^{m\times n}, Y \in \mathbb{R}^{m\times 1}$ are the matrix of features and the vector of labels, respectively. We consider the cost function to be the mean squared error as

$$f(W) := \frac{1}{m}\sum_{i=1}^{m}(y^i - \hat{y}^i)^2 \qquad (2)$$

where
y^i is the i-th label corresponding to x^i the i-th row of features matrix X.
$\hat{y}^i = XW + b$ denotes the i-th predicted value,
W, b are the weight and the bias parameters to be optimized.

For simplicity, in the sequel, we omit b and consider W only. Suppose that we are in the current iteration *(i)*. The idea of gradient descent is to step forward from the current position in the direction of the negative gradient. The step size should be chosen to reach the minimal point. Thus, the problem is to choose α such that

$$h(\alpha) := f\left(W^{(i)} - \alpha\nabla f\left(W^{(i)}\right)\right), \qquad (3)$$

is minimal. The new approximate solution $W^{(i+1)}$ is given as:

$$W^{(i+1)} = W^{(i)} - \alpha\nabla f\left(W^{(i)}\right). \qquad (4)$$

In order to find the minimum of h, we should differentiate h and apply some root-finding algorithms. This is often too complicated, additionally, knowing the exact minimum on the negative gradient line is not really helpful, because the optimum almost certainly does not lie on this line. Hence, it suffices to approximate h by a parabola and to take the absolute minimum on this parabola as an approximation for the minimum of h.

Our algorithm works as follows:

– We first define $\beta_1 = 0$ and $\beta_3 = 1$, and we calculate
$h(\beta_3) = f\left(W^{(i)} - \beta_3\nabla f\left(W^{(i)}\right)\right)$.
– While $h(\beta_3) \geq h(\beta_1) = f\left(W^{(i)}\right)$, we divide β_3 by 2. Note that since β_1 is not the minimum of h, there exists such a value for β_3.
– With $\beta_1 = 0$, β_3 and $\beta_2 = \beta_3/2$, and their corresponding function values, we can construct the parabola $P(\beta) := A\beta^2 + B\beta + C$. The minimum α of P in the interval $[\beta_1, \beta_3]$ is either $\alpha = -\frac{B}{2A}$ or the boundary $\alpha = \beta_3$.

This gives a new approximation as the one in (4).

3. Implementation

We tested our optimizer on the set of data of House Sales in King County, USA [7]. This dataset contains 21613 data points which were broken into 17384 points for train- and 4229 for validation sets. We built a linear neural network from scratch and used the above-explained idea for finding the suitable learning rate. We also used the sequential dense layers from Keras and implemented a neural network along with different optimizers from Keras. The best result was obtained for Adam. Therefore, we compared our results with those of Adam optimizer. We made our comparison in terms of the mean squared error (MSE) and the mean absolute percentage error (MAPE) defined as follows:

$$MSE := \frac{1}{m}\sum_{i=1}^{m}(y^i - \hat{y}^i)^2, \tag{5}$$

and

$$MAPE := \frac{1}{m}\sum_{i=1}^{m}\left|\frac{y^i - \hat{y}^i}{y^i}\right|. \tag{6}$$

Figure 1 and Figure 2 illustrate how MSE and MAPE decrease over the iterations. As shown in the figures, our optimizer falls around the minimum after very few iterations while Adam does not still capture the minimum after 500 epochs. After this number of epochs, our optimizer reached the MAPE error of 20.14 and 20.69 for the training and the validation sets, respectively, while Adam reached the MAPE of 39.03 and 39.76 for the training and the validation sets, respectively. The hyperparameters selected for Adam in this implementation are as follows:

- learning rate: 0.0003
- beta_1=0.9
- beta_2= 0.99
- decay=10^{-6}

4. Conclusions

We presented a learning strategy for a gradient descent optimizer which estimates the scalar function of the learning rate in a typical iteration by a parabola while the parameters and the gradient with respect to the parameters are fixed. We implemented our idea on a set of housing data and compared the progress of mean squared error as well as the mean absolute percentage error with those of Adam over a certain number of epochs. This is done for a linear neural network with whole data used in the process before the parameters get updated. Under these circumstances, our idea appears to find the minimum much faster than Adam does. We have not been successful yet in obtaining satisfactory results in circumstances other than these. The idea for future work is to apply some slight modifications to enable our method to work well for any neural network and for the case of mini-batch gradient descent to fulfill the memory requirement as well.

Acknowledgment

The work of the first author is supported by the Swiss Government Excellence Scholarships grant number ESKAS-2019.0147. This author would also like to thank the support from the University of Applied Sciences and Arts, Northwestern Switzerland.

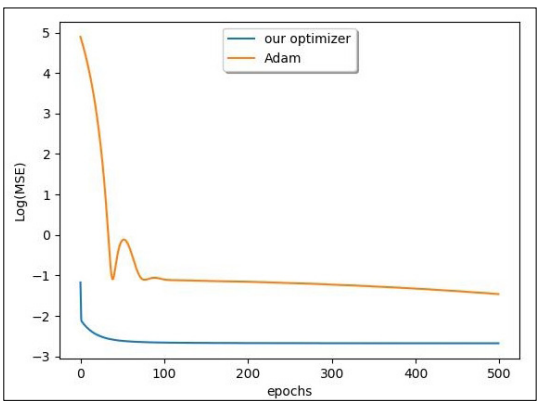

Figure 1. *Logarithm of MSE over 500 epochs for the validation set.*

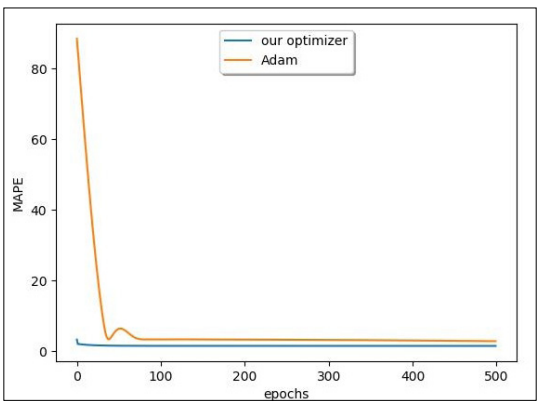

Figure 2. *MAPE over 500 epochs for the validation set.*

References

[1] Keras
https://keras.io/

[2] Robinds H., Monro, H.: *A stochastic approximation method.* Annals of Mathematical Statistics, 22. (1951) 400–407.
https://projecteuclid.org/euclid.aoms/1177729586

[3] Rumelhart D. E., Hinton G. E., Williams R. J.: *Learning representations by back-propagating errors.* Nature, 323. (1986) 533–536.
https://www.nature.com/articles/323533a0

[4] Duchi J., Hazan E., Singer Y.: *Adaptive subgradient methods for online leaning and stochastic optimization*. Journal of Machine Learning Research, 12. (2011) 2121–2159.
http://jmlr.org/papers/v12/duchi11a.html

[5] Zeiler M. D.: *Adadelta: An adaptive learning rate method*. arXiv preprint arXiv:1212.5701, 2012.
https://arxiv.org/abs/1212.5701

[6] Kingma D., Lei Ba J.: *Adam: a method for stochastic optimization*. ICLR 2015.
https://arxiv.org/abs/1412.6980

[7] House Sales in King County, USA: *Predict house price using regression*
https://www.kaggle.com/harlfoxem/housesalesprediction