Faster gradient descent convergence via an adaptive learning rate schedule

Satya Krishna Gorti*
University of Toronto
27 King's College Circle
Toronto, ON M5S
satyag@cs.toronto.edu

Mathieu Ravaut*
University of Toronto
27 King's College Circle
Toronto, ON M5S
mravox@cs.toronto.edu

Abstract

Any gradient descent requires to choose a learning rate. With deeper and deeper models, tuning that learning rate can easily become tedious and does not necesarily lead to an ideal convergence. We propose a variation of the gradient descent algorithm in the which the learning rate η is not fixed. Instead, we learn η itself, either by another gradient descent (first-order method), or by Newton's method (second-order). That way, gradient descent for any machine learning algorithm can be optimized.

1 Introduction

In the past decades, gradient descent has been widely adopted to optimize the loss function in machine learning algorithms *ref*. Lately, the machine learning community has also used the stochastic gradient descent alternative *ref*. Gradient descent can be used in any dimension, and presents the advantage of being easy to understand and inexpensive to compute. Under certain assumptions on the loss function (such as convexity), gradient descent is guaranteed to converge to the minimum of the function. However, stochastic gradient descent has proven to be very efficient even in situations where the loss functions is not convex, as is mostly the case with modern deep neural networks *ref*. Other methods such as Newton's method guarantee a much faster convergence, but are typically very expensive. Newton's method for instance requires to compute the inverse of the Hessian matrix of the loss functions with regards to all parameters, which is impossible with today's hardware and today's deep networks with millions of parameters *ref*.

The quality of a gradient descent heavily depends on the choice of the learning rate η. A too high learning rate will see the loss function jumping around the direction of steepest descent, and eventually diverge. While a very low learning rate guarantees non-divergence, convergence will be very slow, and the loss function might get stuck in a local minimum. Choosing an ideal learning rate requires an intuition of the problem. Typically, researchers would start by performing a line-search over a set of different orders of magnitude of learning rates, but this is long and costly. Besides, a line-search usually assumes a fixed learning rate over time, as doing one line-search per iteration would require exponential computation cost. Usually, we see researchers setting the learning rate to an initial value then decrasing it one or a few items after training has progressed *ref*.

In this paper, we propose to automatically find the learning rate. We still need to input an initial value, but at each iteration, our model will find a learning that optimizes best the loss function at this point of learning. We explore a first-order method and a second-order method to do so, with a strong emphasis on the latter. Our method could be applied to any machine learning algorithm using gradient descent. We show faster convergence on a variety of tasks and models.

* indicates equal contribution

2 Related work

3 Adaptive learning rate

An adaptive learning gradient descent has the following schedule:

$$w(t+1) = w(t) - \eta(t)g(t) \tag{1}$$

where $g(t) = \nabla L(w(t))$, L is the loss function, and w(t) represents the state of the model's weights at time t. In the following we present several ways to update $\eta(t)$.

3.1 First-order method

The first-order method consists in doing gradient descent on the learning rate. Let's introduce a function that will be useful in the following:

$$f : R^n \to R \tag{2}$$
$$\eta \to L(w(t) - \eta g(t)) \tag{3}$$

n corresponds to the number of learnable parameters in the model. f represents how the loss would be if we were to perform a gradient descent update with the given η.
The first-order method is written:

$$w(t+1) = w(t) - \eta(t)g(t) \tag{4}$$
$$\eta(t+1) = \eta(t) - \alpha f'(t) \tag{5}$$

This method introduces a new "meta" learning rate α. It has the advantage of begin light in cost as we only need gradients of f. Indeed:

$$\forall \eta, f'(\eta) = -g(t)^T.\nabla L(w(t) - \eta g(t)) \text{ where . is the dot product in dimension n} \tag{6}$$

In particular, at the value of η used to get w(t+1), we get:

$$f'(\eta(t)) = -g(t)^T.g(t+1) \tag{7}$$

3.2 Second-order method

The previous method transfers the problem of choosing an ideal learning rate for weights to choosing an ideal learning rate for the learning rate itself. To avoid that problem, the second-order method uses a Newton-Raphson algorithm:

$$w(t+1) = w(t) - \eta(t)g(t) \tag{8}$$
$$\eta(t+1) = \eta(t) - \frac{f'(\eta(t))}{f''(\eta(t))} \tag{9}$$

Now all the learning only depends on the loss. However, the second derivative of requires building its Hessian matrix:

$$\forall \eta, f''(\eta) = g(t)^T.H_L(w(t) - \eta g(t)) \tag{10}$$

We propose to approximate this Hessian using finite differences. By using on f' then f", we get an update formula of the learning rate depending only on several values of the loss function:

$$f'(\eta + \epsilon) \approx \frac{f(\eta + 2\epsilon) - f(\eta)}{2\epsilon} \tag{11}$$

$$\text{and } f'(\eta - \epsilon) \approx \frac{f(\eta) - f(\eta - 2\epsilon)}{2\epsilon} \tag{12}$$

$$\text{so } f''(\eta) \approx \frac{f(\eta + 2\epsilon) + f(\eta - 2\epsilon) - 2f(\eta)}{4\epsilon^2} \tag{13}$$

Given the finite differences on the first derivative:

$$f'(\eta) \approx \frac{f(\eta + \epsilon) - f(\eta - \epsilon)}{2\epsilon} \tag{14}$$

We get the final simple formula for η:

$$\eta(t+1) = \eta(t) - 2\epsilon \frac{(f(\eta + \epsilon) - f(\eta - \epsilon))}{f(\eta + 2\epsilon) + f(\eta - 2\epsilon) - 2f(\eta)} \tag{15}$$

Assuming a constant sign for the numerator, this formula means the following: when slightly increasing the learning rate corresponds to a lower loss than slightly reducing it, then the numerator is negative. In consequence, the learning rate increases, as pushing in the positive direction for the lerning rate helps reduce the loss. However, reality is more complex as the sign of the denominator changes as well.

3.3 BFGS

4 Experiments

4.1 Practical considerations

With the second-order method, the denominator in the learning rate update might underflow. To avoid such a situation, we add to the denominator a small smoothing value when it reaches zero machine, but only when it does so:

$$\text{if } (f(\eta + 2\epsilon) + f(\eta - 2\epsilon) - 2f(\eta)) \approx 0 \tag{16}$$

$$\text{then } \eta(t+1) = \eta(t) - 2\epsilon \frac{(f(\eta + \epsilon) - f(\eta - \epsilon))}{f(\eta + 2\epsilon) + f(\eta - 2\epsilon) - 2f(\eta) + \delta} \tag{17}$$

A typical value of δ is 10^{-6}.

Besides, when updating the loss and the learning rate, one should be very carefl about the order of the operations. Let's say that we just perform the *k-th* iteration. We have a loss value $L^{(k)}$ and a learning rate value $\eta^{(k)}$. Then the *(k+1)-th* step of our algorithm is written as follows:

- Get the five loss values $f(\eta^{(k)} + \epsilon)$, $f(\eta^{(k)} - \epsilon)$, $f(\eta^{(k)} + 2\epsilon)$, $f(\eta^{(k)} - 2\epsilon)$ and $f(\eta^{(k)})$
- $L^{(k+1)} \leftarrow f(\eta^{(k)})$
- $\eta^{(k+1)} \leftarrow \eta^{(k)} - 2\epsilon...$

4.2 Results

We compare our second-order method to a basic method, which is the same algorithm but with a fixed learning rate schedule. That means, the basic method uses plain stochastic gradient descent as optimizer. The first order method was quite unstable. When relevant, we also include its results. We compare loss and accuracy on both training and test sets. Our goal in these experiments is not to break state of the art on the given problem, but to show an improvement in our method compared to the basic approach.

4.2.1 Linear regression

We first tried our adaptive learning rate schedule on linear regression applied to the Boston Housing dataset *ref.* This dataset of 506 points with 13 features is suitable for a simple linear regression model.

4.2.2 Logistic regression

We also tried our method with multi-class logistic regression on the fashion MNIST dataset *ref*.

4.2.3 Image classification with neural networks

CIFAR-10

A common benchmark in machine learning is to classify images on the CIFAR-10 dataset *ref*. Deep neural networks are most suited for this task *ref*. We use a LeNet with 5 layers and ResNet-18.

We first started by optimizing the LeNet model. We trained for different values of the learning rate, and 0.01 appeared to be the best choice. Thus, we compare the basic method with the second-order adaptive one starting at 0.01:

figure: losses, LeNet, starting LR 0.01
figure: accuracies, LeNet, starting LRL 0.01

(comments)

In the ResNet paper by *ref*, ResNet is optimized via the Adam optimizer. As we are using SGD here, we added dropout (on both the basic and adaptive methods of course) at the end of the second and fourth blocks of the network to prevent overfitting. We use a batch size of 256 for training.

figure: losses, ResNet, starting LR 0.1
figure: accuracies, ResNet, starting LRL 0.1

In the adaptive schedule, all training metrics (training and test loss, training and test accuracies) are optimized faster. Typically, in the first epochs, training loss is twice lower in the adaptive version. The second-order method reaches its plateau sooner as well, after around xx epochs versus xx in the basic case, and then starts to overfit. Interestingly, the best performance achieved by the adaptive method is almost exactly equal to the one achieved by the basic method (and not better).

Let's have a look at the variations of the learning rate for both networks:

figure: lr, LeNet, starting LR 0.01
figure: lr, ResNet, starting LR 0.1 1st case
figure: lr, ResNet, starting LR 0.1 2nd case

Experiments on ResNet gave two kinds of results, that we both plotted here. It seems that a starting learning rate of 0.1 for ResNet was a bit too high, as we see it decrease its learning rate in every experiment. Let's see what happens if we set the initial learning rate at 0.01:

figure: losses, ResNet, starting LR 0.01
figure: accuracies, ResNet, starting LR 0.01 figure: lr, ResNet, starting LR 0.01

Once again, the adaptive schedule performs better than the basic one. This time, the initial learning rate seems to be too small, as the network increases its value until stabilizing it around xxx.

CIFAR-100

In this section, we increment difficulty by solving an image classification problem with 10 times more classes. After a line-search, the best learning rate to train ResNet-18 on this dataset seems to be 0.a

figure: losses, ResNet, starting LR 0.a
figure: accuracies, ResNet, starting LR 0.a
figure: lr, ResNet, starting LR 0.a

Once again, we see ... This time the converged value of LR is ...

5 Further exploration

5.1 Learning the learning rate

In the previous experiments, we have seen the learning rate converge to a given value over time. This value seems to depend on both the model and the dataset. Now a question rises: given a dataset and a model, is this value the ideal learning rate ?

We started an adaptive learning rate training with the learning rate value that the ResNet model converged to in section 4.2.2 with 0.01 as a starting learning rate.

figure: losses, ResNet, starting LR 0.xx
figure: accuracies, ResNet, starting LR 0.xx
figure: lr, ResNet, starting LR 0.xx

This time, the learning rate variations are of much weaker amplitude, as the this parameter is already in a ideal zone.

5.2 Momentum

5.3 Getting loss values via a validation set

5.4 Comparison with other optimizers

In the last few years, a popular variation of gradient descent named Adam *ref* has progressively gained consensus among researchers for its efficiency. Thus, we compared our method with Adam and its default learning of 10^{-3}.

figure: losses, ResNet, starting LR 0.001
figure: accuracies, ResNet, starting LR 0.001
figure: lr, ResNet, starting LR 0.001

6 Limitations

6.1 Choice of step ϵ

In finite differences, the choice of the paramater ϵ is a main issue. Indeed, in any deep learning problem, we can expect the loss function to present noisy oscillations locally. On one hand, with a too small value of ϵ, we might not capture meaningful variations of the f function. On the other hand, a too large ϵ would make the learning rate oscillate too much, reaching either too high values or even negative values. Both cases can in turn quickly make the loss function diverge. We have seen empirically that when the learning rate gets negative, training will likely not converge. All our reported experiments were done using an ϵ value of 10^{-5}, which seemed to be a good compromise.

6.2 Choice of the initial learning rate

In our algorithm schedule, learning rate variations are automatic, but we have to choose the initial rate value. This value seems not to matter so much, as with several ranges of initial values, our algorithm still converges (to the same value).

6.3 Cost of loss computations

We have shown faster training in terms of number of epochs. However, at each iteration, we have to perform 5 back-propagations instead of one. Thus, computation time is slowed down compared to the fixed learning rate approach. *show experiments results*. Assuming that the experimenter does not care about time but about finding the best loss function minimization, it is not such a big problem.

6.4 Overfitting

Our version of gradient descent reduces the training loss much faster than in the fixed learning rate approach. The training loss sometimes gets down to very surprisingly low values, while the test loss

does not reduce much more than in the basic model. Thus, our model seems to be more likely to overfit. We have tried to add dropout to the ResNet model, and that proved successful.

7 Conclusion

References

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.