No more learning rate tuning in gradient descent

Introduction to Machine Learning CSC2515 Project Proposal

Mathieu Ravaut, Satya Gorti

November 2, 2017

Abstract

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 Context

Gradient descent is massively used for optimization tasks which assume that the loss function is differentiable, especially in machine learning. However, to be efficient, it requires an optimal learning rate. A too small learning rate could lead to a very slow convergence, or the loss function stuck in a local minimum ; while a too large one could mean divergence. For a generic gradient descent problem with a cost function L that we want to minimize, let's introduce the following notations:

- $\omega(t)$ represents the weights at instant t
- $g(t) = \nabla L(\omega(t))$ the gradient of the cost function at instant t
- $\eta(t)$ is the learning rate at instant t

The gradient descent problem with an adaptive learning rate can be written:

$$\omega(t+1) = \omega(t) - \eta(t).g(t) \tag{1}$$
$$\eta(t+1) = \eta(t) + \delta(t) \tag{2}$$
$$\tag{3}$$

with $\delta(t)$ to be carefully chosen.

2 1^{st} order method

In the first order, the update on the learning rate is written as:

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

where α is a meta-learning rate and f is the real function defined by:

$$f : \eta \to L(\omega(t) - \eta.g(t))$$

Getting the first derivative of f is straightforward:

$$f'(\eta) = -g(t)^T.\nabla L(\omega(t) - \eta g(t)) \text{ for any } \eta \tag{6}$$

Let's note that:

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

The intuition behind this gradient is easy: if we continue in a similar direction then we increase the learning rate, if we backtrack then we decrease it. One problem is that the algorithm is **not scale invariant** anymore - it will behave differently for $L'(\omega) = \lambda L(\omega)$.

3 2^{nd} order method

The second order method gets rid of this scale dependency, as η is now written as:

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

No meta-rate α anymore, but the second derivative of f is tricky to get.

3.1 Analytical formula

The analytical formula for the second derivative is given by:

$$f''(\eta) = g(t)^T.H_{\omega(t)-\eta.g(t)}(-g(t)) \tag{10}$$

However, with n parameters in the model, the Hessian has dimension $n \times n$, so it is very expensive to compute. We would then need a cost-effective way to approximate this second derivative.

3.2 Finite differences

We propose to approximate the second derivative of f via the **finite differences** method. Applying it to f' then f" leads to an update on η depending only on the cost function at time t. We use **Laplacian smoothing** to avoid overflowing issues. For any ϵ:

$$\eta(t+1) \approx \eta(t) - 2\epsilon \frac{f(\eta(t)+\epsilon) - f(\eta(t)-\epsilon) + \epsilon}{f(\eta(t)+2\epsilon) + f(\eta(t)-2\epsilon) - 2f(\eta(t)) + \epsilon} \tag{11}$$

Note that we use ϵ as the Laplacian smoothing constant, so that we only have one parameter to set. Note also that we could apply finite differentiation only once, and get η as a function of f' (e.g. a function of ∇L).

4 Experiments

We plan on exploring the effects of both first-order and second-order methods on the quality of convergence. Quality means both **speed of convergence**, and avoidance of **local minima** to reach the global minimum. This general method for automatic learning rate setting can be applied to any machine learning task involving gradient descent. Besides, if successful, this adaptive learning schedule would mean that the user should **not worry about tuning the learning rate** to find an optimal convergence anymore. With our method, the only choice is the initial learning rate. We will compare traditional gradient descent with a fixed learning rate with our method on the following examples:

- Linear Regression. The proposed dataset is the Boston House Prices dataset studied in Assignment 1. First experiments show a much faster cost function convergence using the second-order method. The first-order shows signs of instability.
- Logistic Regression.
- Image classification via neural networks. Application to simple datasets such as MNIST or CIFAR-10.

5 Further exploration.

We are thinking of combining an adaptive learning rate method with other optimization algorithms, such as **momentum**.

Besides, the two methods we mentionned could possibly lead to unstable learning rates, perhaps even taking negative values. We will explore constraining learning rates ranges, by using an **activation function** σ for instance:

$$\eta(t+1) = \sigma(\eta(t) + \delta(t)) \tag{12}$$