Adaptive Learning Rate Method

Adaptive learning rate methods are an optimization of gradient descent methods with the goal of minimizing the objective function of a network by using the gradient of the function and the parameters of the network.

1. Gradient descent
2. Adaptive Learning Rate Method
3. Literature
4. Weblinks

Gradient descent

Before the adaptive learning rate methods were introduced, the gradient descent algorithms including Batch Gradient Descent (BGD), Stochastic Gradient Descent (SGD) and mini-Batch Gradient Descent (mini-BGD, the mixture of BGD and SGD) were state-of-the-art. In essence, these methods try to update the weights of the network, with the help of a learning rate \( \eta \), the objective function \( J(\theta) \) and the gradient of it, \( \nabla J(\theta) \). What all gradient descent algorithms and its improvements have in common, is the goal of minimizing \( J(\theta) \) in order to find the optimal weights \( \hat{\theta} \).

The simplest of the three is the BGD.

\[
\theta = \theta - \eta \cdot \nabla \theta J(\theta)
\]

It tries to reach the minimum of \( J(\theta) \), by subtracting from \( \theta \) the gradient of \( J(\theta) \) (refer to Figure 3 for a visualization). The algorithm always computes over the whole set of data, for each update. This makes the BGD the slowest and causes it to be unable to update online. Additionally, it performs redundant operations for big sets of data, computing similar examples at each update and it converges to the closest minimum depending on the given data, resulting in potential suboptimal results.

An often used algorithm is the SGD.

\[
\theta = \theta - \eta \cdot \nabla \theta J(\theta; x^{(i)}; y^{(i)})
\]

Contrary to BGD, SGD updates for each training example \((x^{(i)}; y^{(i)})\), thus updating according to a single example step. Furthermore, this fluctuation enables the SGD to jump to minima farther away, potentially reaching a better minimum. But thanks to this fluctuation, SGD is also able to overshoot. This can be counteracted by slowly decreasing the learning rate. In the exemplary code shown in Figure 2, a shuffle function is additionally used in the SGD and mini-BGD algorithm, compared to the BGD. This is done, as it is often preferable to avoid meaningful order of the data and thereby avoid bias of optimization algorithm, although sometimes better results can be achieved with data in order. In this case the shuffle operation is to be removed.

Lastly, there is the mini-BGD.

\[
\theta = \theta - \eta \cdot \nabla \theta J(\theta; x^{(i+n)}; y^{(i+n)})
\]

The mini-BGD updates for every mini-batch of \( n \) training examples. This leads to a more stable convergence, by reducing the variance of the parameters. When people talk about a SGD algorithm, they often refer to this version.

Author: Johannes Kuhn

Figure 1: Local minima may occur in \( J(\theta) \) (here \( J(w) \)), which may result in suboptimal solutions for some gradient descent methods.
### Fast Learning Methods

<table>
<thead>
<tr>
<th>batch gradient descent (BGD)</th>
<th>stochastic gradient descent (SGD)</th>
<th>mini-batch gradient descent (mini-BGD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>for i in range (nb_epochs):</td>
<td>for i in range(np_epochs):</td>
<td>for i in range(np_epochs):</td>
</tr>
<tr>
<td>params_grad = evaluate_grad(loss_function, data, params)</td>
<td>np.random.shuffle(data)</td>
<td>np.random.shuffle(data)</td>
</tr>
<tr>
<td>params = params - learning_rate * params_grad</td>
<td>for example in data:</td>
<td>for batch in get_batches (data, batch_size=50):</td>
</tr>
<tr>
<td></td>
<td>params_grad = evaluate_grad(loss_function, example, params)</td>
<td>params_grad = evaluate_grad(loss_function, batch, params)</td>
</tr>
<tr>
<td></td>
<td>params = params - learning_rate * params_grad</td>
<td>params = params - learning_rate * params_grad</td>
</tr>
</tbody>
</table>

**Figure 2:** (1) Pseudo code of the three gradient descent algorithms

### Adaptive Learning Rate Method

As an improvement to traditional gradient descent algorithms, the adaptive gradient descent optimization algorithms or adaptive learning rate methods can be utilized. Several versions of these algorithms are described below.

**Momentum** can be seen as an evolution of the SGD.

\[ v_t = \gamma v_{t-1} + \eta \nabla \theta J(\theta) \]
\[ \theta = \theta - v_t \]

While SGD has problems with data having steep curves in one direction of the gradient, Momentum circumvents that by adding the update vector of the time step before multiplying it with a \( \gamma \), usually around 0.9 (1). As an analogy, one can think of a ball rolling down the gradient, gathering momentum (hence the name), while still being affected by the wind resistance (0 < \( \gamma \) < 1).

**Nesterov accelerated gradient** can be seen as a further enhancement to momentum.

\[ v_t = \gamma v_{t-1} + \eta \nabla \theta J(\theta - \gamma v_{t-1}) \]
\[ \theta = \theta - v_t \]

This algorithm adds a guess of the next step, in the form of the term \( \gamma v_{t-1} \). A comparison for the first two steps of Momentum and Nesterov accelerated gradient can be found in Figure 3. The additional term results in a consideration of the error of the previous step, accelerating the progress in comparison to momentum.

Contrary to the nesterov accelerated gradient, **Adagrad** adapts its learning rate \( \eta \) during its run-time and it updates its parameters \( \theta_i \) separately during each time step \( t \). It has to do that, since \( \eta \) adapts for every \( \theta_i \) on its own.

\[ \theta_{i+1} = \theta_{i} - \frac{\eta}{\sqrt{G_{i+1}} + \varepsilon} \cdot \nabla \theta J(\theta_i) \]

\( G_i \) is a matrix containing the squared sum of the past gradients with regards to all \( \theta \) along its diagonal.
\( \epsilon \) is correction term which is utilized to avoid dividing by 0 and is generally insignificantly small (~\(10^{-8}\)).

Due to the accumulation of the squared gradients in \( G_t \), the learning rate \( \frac{\|G_t\|}{\sqrt{G_t^2 + \epsilon}} \) gets smaller over time, finally leading to a significantly small rate, which causes the algorithm to obtain no new knowledge.

**Literature**


**Weblinks**


(5) [http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html](http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html)