Adagrad

Adagrad [3] is an algorithm for gradient-based optimization that does just this: It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. For this reason, it is well-suited for dealing with sparse data. Dean et al. [4] have found that Adagrad greatly improved the robustness of SGD and used it for training large-scale neural nets at Google, which -- among other things -- learned to recognize cats in Youtube videos. Moreover, Pennington et al. [5] used Adagrad to train GloVe word embeddings, as infrequent words require much larger updates than frequent ones.

Previously, we performed an update for all parameters θ at once as every parameter θ_i used the same learning rate η. As Adagrad uses a different learning rate for every parameter θ_i at every time step t, we first show Adagrad's per-parameter update, which we then vectorize. For brevity, we set $g_{t,i}$ to be the gradient of the objective function w.r.t. to the parameter θ_i at time step t:

$$g_{t,i} = \nabla_\theta J(\theta_i).$$

The SGD update for every parameter θ_i at each time step t then becomes:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}.$$

In its update rule, Adagrad modifies the general learning rate η at each time step t for every parameter θ_i based on the past gradients that have been computed for θ_i:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}.$$

$G_t \in \mathbb{R}^{d \times d}$ here is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. θ_i up to time step t [25], while ϵ is a smoothing term that avoids division by zero (usually on the order of $1e-8$). Interestingly, without the square root operation, the algorithm performs much worse.

As G_t contains the sum of the squares of the past gradients w.r.t. to all parameters θ along its diagonal, we can now vectorize our implementation by performing an element-wise matrix-vector multiplication $\odot$ between G_t and g_t:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$

One of Adagrad's main benefits is that it eliminates the need to manually tune the learning rate. Most implementations use a default value of 0.01 and leave it at that.

Adagrad's main weakness is its accumulation of the squared gradients in the denominator: Since every added term is positive, the accumulated sum keeps growing during training. This in turn causes the learning rate to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge. The following algorithms aim to resolve this flaw.