Gradient descent (GD)

Let $J(\boldsymbol{\theta}, D)$ be a cost function, where $\boldsymbol{\theta}$ is an n_θ-length parameter vector and D a dataset of n_d samples. Moreover, let $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, D)$ be the gradient of cost function $J(\boldsymbol{\theta}) := J(\boldsymbol{\theta}, D)$ with respect to $\boldsymbol{\theta}$. The approximation $\boldsymbol{\theta}^{(k)}$ of $\boldsymbol{\theta}$ at step k of GD given the approximation $\boldsymbol{\theta}^{(k-1)}$ of $\boldsymbol{\theta}$ at step $k-1$ is defined as

$$\boldsymbol{\theta}^{(k)} := \boldsymbol{\theta}^{(k-1)} - a\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(k-1)}, D), \tag{1}$$

where the learning rate a is a positive real number.

In the case of a linear regression model, the data are set to $D := \{X, y\}$, where X is the $n_d \cdot n_\theta$ design matrix and y is the n_d-length output vector associated with linear regression.

Stochastic gradient descent (SGD)

For a big data set D consisting of a large number n_d of data points, the evaluation of the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, D)$ can be computationally expensive. To reduce the computational complexity of GD, stochastic gradient descent (SGD) samples an s-size subset $D_s^{(k)}$ of the data D at step k and evaluates the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, D_s^{(k)})$ using the subset $D_s^{(k)}$. Thus, the approximation $\boldsymbol{\theta}^{(k)}$ of $\boldsymbol{\theta}$ at step k of SGD given the approximation $\boldsymbol{\theta}^{(k-1)}$ of $\boldsymbol{\theta}$ at step $k-1$ is given by

$$\boldsymbol{\theta}^{(k)} := \boldsymbol{\theta}^{(k-1)} - a\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(k-1)}, D_s^{(k)}). \tag{2}$$

SGD is more computationally efficient than GD in large-scale machine learning problems entailing large n_d or large n_θ.

As an implementation hint, use the `sample()` function in R to sample s out of n_d samples at step k without replacement. Subsequently, make sure that you index the design matrix X and output variable y appropriately.

The rest of stochastic optimization methods of this project also use a random subset $D_s^{(k)}$ of the data D at step k.

SGD with momentum (MSGD)

In SGD with momentum (MSGD), a 'momentum' term $\mathbf{m}$ is defined, which is a moving average of the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, D_s^{(k)})$, and the parameters are then updated using the momentum $\mathbf{m}$ instead of the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, D_s^{(k)})$.

The momentum $\mathbf{m}^{(k)}$ and parameter approximation $\boldsymbol{\theta}^{(k)}$ at the k-th step of MSGD are defined recursively according to

$$\mathbf{m}^{(k)} := b\mathbf{m}^{(k-1)} + (1-b)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(k-1)}, D_s^{(k)}), \tag{3}$$

$$\boldsymbol{\theta}^{(k)} := \boldsymbol{\theta}^{(k-1)} - a\mathbf{m}^{(k)}. \tag{4}$$

MSGD has two hyper-parameters, a momentum memory factor b with values in $(0, 1)$ and a positive learning rate a. Furthermore, an initial momentum value $\mathbf{m}^{(0)}$ is required by equation (3) at step $k = 1$.

For values of b closer to one, the moving average equation (3) retains stronger memory of momentum history. For values of b closer to zero, the moving average equation (3) remembers less any past momentum history and places more emphasis on the current gradient of the cost function.

Variant representations of MSGD are available in the literature. Equations (3)-(4) have been preferred due to providing a clearer understanding of MSGD.

SGD with Nesterov accelerated gradient (NAGSGD)

SGD with Nesterov accelerated gradient (NAGSGD) differs from MSGD only in the way it calculates the gradient of the cost function in the moving average equation. More specifically, NAGSGD calculates the gradient of the cost function with respect to approximate future position $\boldsymbol{\theta}^{(k-1)} - ab\mathbf{m}^{(k-1)}$ rather than with respect to the current parameter approximation $\boldsymbol{\theta}^{(k-1)}$.

Thus, step k of NAGSGD is defined as

$$\mathbf{m}^{(k)} := b\mathbf{m}^{(k-1)} + (1-b)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(k-1)} - ab\mathbf{m}^{(k-1)}, D_s^{(k)}), \tag{5}$$

$$\boldsymbol{\theta}^{(k)} := \boldsymbol{\theta}^{(k-1)} - a\mathbf{m}^{(k)}. \tag{6}$$

Alternative expressions for NAGSGD can be found in the literature. Equations (5)-(6) have been chosen for the purpose of communicating the moving average model for momentum and the resulting parameter update clearly.

AdaGrad

The learning rate a of SGD can be too small for one parameter and too large for another one. This problem tends to manifest itself for large number n_θ of parameters. AdaGrad mitigates such a problem by scaling a different learning rate for each parameter adaptively and automatically.

Using the shorthand $\mathbf{g}^{(k)} := \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(k)}, D_s^{(k)})$, consider the outer product $\mathbf{g}^{(k)} \times \mathbf{g}^{(k)}$ at step k, which is an $n_\theta \cdot n_\theta$ matrix. Summing over all steps up to k yields the $n_\theta \cdot n_\theta$ matrix

$$G^{(k)} := \sum_{\ell=0}^{k} \mathbf{g}^{(\ell)} \times \mathbf{g}^{(\ell)}. \tag{7}$$

At step k, AdaGrad performs the parameter update

$$\boldsymbol{\theta}^{(k)} := \boldsymbol{\theta}^{(k-1)} - a(\mathrm{diag}(G^{(k-1)}) + \epsilon)^{\odot(-0.5)} \odot \mathbf{g}^{(k-1)}, \tag{8}$$

where a is a positive learning rate, $\mathrm{diag}(G^{(k-1)})$ is the diagonal of matrix $G^{(k-1)}$, ϵ is a positive smoothing term that avoids division by zero (usually on the order of $1e-8$) and $\odot$ denotes Hadamard (element-wise) operations. Step $k=1$ requires an initial value $G^{(0)}$.

To further clarify AdaGrad, equation (8) will be written below in an alternative form by using element-wise notation instead of matrix notation. Let $g_j^{(k)} := \partial J(\boldsymbol{\theta}^{(k)}, D_s^{(k)})/\theta_j$ be the j-th partial derivative of the cost function at step k, i.e. the j-th coordinate of the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(k)}, D_s^{(k)})$. The AdaGrad update of the j-th parameter $\theta_j^{(k)}$ at step k is

$$\theta_j^{(k)} := \theta_j^{(k-1)} - \frac{a}{\sqrt{G_{jj}^{(k-1)} + \epsilon}} g_j^{(k-1)}, \tag{9}$$

where $G_{jj}^{(k-1)}$ is the (j,j)-th diagonal element of $G^{(k-1)}$.

Equation (9) makes it clear that AdaGrad uses a differently scaled learning rate $a/\sqrt{G_{jj}^{(k-1)} + \epsilon}$ for each parameter θ_j.

As a hint to assist implementation, you may focus on equation (9). You do not need to compute the outer products appearing in equation (8). It suffices to calculate the diagonal of matrix $G^{(k)}$, which means that it suffices to calculate the sum of squared gradients

$$G_{jj}^{(k)} = \sum_{\ell=0}^{k} (g_j^{(\ell)})^2. \tag{10}$$

AdaGrad eliminates the need to manually tune the learning rate, since it tunes adaptively the learning rate for each parameter according to equation (9). However, the adaptive tuning of learning rates in AdaGrad comes with a weakness. Squared gradients accumulate in the denominator of learning rate $a/\sqrt{G_{jj}^{(k-1)} + \epsilon}$, hence learning rates shrink during training and AdaGrad is no longer able to perform parameter updates via equation (9).

RMSProp

RMSProp attempts to alleviate the aforementioned weakness of AdaGrad by applying a moving average model to squared gradients and by then using the output of the moving average model in the AdaGrad parameter update mechanism.

More concretely, the j-th coordinate $v_j^{(k)}$ of the moving average of squared gradients and j-th parameter $\theta_j^{(k)}$ at step k of RMSprop are updated according to

$$v_j^{(k)} := cv_j^{(k-1)} + (1-c)(g_j^{(k-1)})^2, \tag{11}$$

$$\theta_j^{(k)} := \theta_j^{(k-1)} - \frac{a}{\sqrt{v_j^{(k)} + \epsilon}} g_j^{(k-1)}. \tag{12}$$

The hyperparameter c is known as the memory factor for squared gradients, taking values in $c \in (0,1)$. Equation (11) requires an initial value $v_j^{(0)}$ at step $k = 1$. Taking into account all coordinates $j = 1, 2, \ldots, n_\theta$, the initial value $\mathbf{v}^{(0)}$ for the moving average model of RMSProp is an n_θ-length vector.

Adam

Adaptive moment estimation (Adam) is a stochastic optimization method that computes a different learning rate for each parameter adaptively. Apart from retaining a moving average of squared gradients similarly to RMSProp, Adam stores a moving average of gradients additionally.

The Adam update of the j-th parameter $\theta_j^{(k)}$ at step k is given by

$$m_j^{(k)} := bm_j^{(k-1)} + (1-b)g_j^{(k-1)}, \tag{13}$$

$$v_j^{(k)} := cv_j^{(k-1)} + (1-c)(g_j^{(k-1)})^2, \tag{14}$$

$$\hat{m}_j^{(k)} := \frac{m_j^{(k)}}{1-b^k}, \tag{15}$$

$$\hat{v}_j^{(k)} := \frac{v_j^{(k)}}{1-c^k}, \tag{16}$$

$$\theta_j^{(k)} := \theta_j^{(k-1)} - \frac{a}{\sqrt{\hat{v}_j^{(k)} + \epsilon}} \hat{m}_j^{(k)}. \tag{17}$$

Adam has three hyperparameters, namely a positive learning rate a, a memory factor b of gradients with values in $(0,1)$ and a memory factor c of squared gradients with values in $(0,1)$. Morevover, Adam requires the initial values $\mathbf{m}^{(0)}$ and $\mathbf{v}^{(0)}$ at step $k = 1$.

Equations (13) and (14) provide the moving average of gradients and of squared gradients, respectively. Adam takes its name after these first and second order estimators of gradients. Due to $\mathbf{m}^{(0)}$ and $\mathbf{v}^{(0)}$ being initialized typically as vectors of zeros, the moving averages $\mathbf{m}^{(k)}$ and $\mathbf{v}^{(k)}$ are asymptotically biased towards zero, especially when b and c are set to a value close to one. To contouract such biases while estimating the moving averages of gradients and of squared gradients, equations (15) and (16) are introduced.