Gradient-Based Optimization Algorithms

A. Ho

August 24, 2017

1 Gradient-Based Optimization

Gradient-based optimization methods are a class of numerical methods used to solve problems of the form:

$$\min_{\mathbf{x}\in\mathbb{R}} f(\mathbf{x}) \qquad \text{or} \qquad \max_{\mathbf{x}\in\mathbb{R}} f(\mathbf{x}) \tag{1}$$

where $f(\mathbf{x})$ is a sufficiently smooth function for which the location of the minima or maxima, $\mathbf{x}_*$, must be found and $\mathbf{x}$ contains the set of input parameters of the function. By taking advantage of the fact that the gradient of a function is zero at a minimum or maximum, the solutions to this optimization problem can be defined as follows:

$$\nabla_{\mathbf{x}} f(\mathbf{x}_*) = \mathbf{0} \tag{2}$$

However, even if an analytical expression for the gradient exists, it may not be possible to decouple the input parameters to form a system of linear equations. In these cases, it would be impossible to directly solve for the input combinations which satisfy Equation (1) by simply reforming it into $\mathbf{A}\mathbf{x} = \mathbf{b}$ and employing well-known inversion algorithms. Thus, these cases must typically be solved in an iterative fashion, in which an initial guess, $\mathbf{x}_0$, is provided and consequently updated with *steps*, denoted as $\Delta\mathbf{x}_i$, to converge onto the desired solution, $\mathbf{x}_*$. The generic recursive update operation can be expressed as follows:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}_i \ , \qquad i = 0, 1, 2, ... \tag{3}$$

where $\Delta\mathbf{x}_i$ is determined via a method-specific *step estimator*. These step estimators typically contain one or more externally-tunable parameters which allow the method to be adapted to a greater variety of problems.

In gradient-based optimization methods, the step estimator is dependent on the gradient of the function evaluated at the current set of inputs, $\nabla_{\mathbf{x}} f(\mathbf{x}_i)$, as the gradient always points in the direction of the greatest increase of f per unit step in the space defined by $\mathbf{x}$. Due to this property, any maximization problem can be converted into a minimization problem, and vice versa, simply by making the substitution $f(\mathbf{x}) \rightarrow -f(\mathbf{x})$ within this class of methods. For simplicity of notation and improved readability, this document assumes that the objective function, $f(\mathbf{x})$, needs to be **maximized** with respect to $\mathbf{x}$.

1.1 Gradient Method

The most basic gradient-based optimization method is called the *gradient ascent* method for a maximization problem, or consequently the *gradient descent* method for a minimization problem. The step estimator used in this method is as follows:

$$\Delta\mathbf{x}_i = \gamma\,\mathbf{G}_i \tag{4}$$

where γ is typically less than unity and

$$\mathbf{G}_i \equiv \nabla_{\mathbf{x}} f(\mathbf{x}_i) \tag{5}$$

Tunable parameters: γ

The parameter γ is usually called the *learning rate*, as it can be interpreted as how much weight the search algorithm puts on the new information provided by the gradient calculation. Due to its extreme simplicity, this method is the most robust gradient-based optimization method, meaning it is almost always capable of finding a solution provided that one exists. However, it is also one of the slowest to converge on said solution due to the fact that $\mathbf{G}_i \to \mathbf{0}$ as $\mathbf{x}_i \to \mathbf{x}_*$. For this reason, other methods have been developed which sacrifice a bit of robustness for convergence speed.

1.2 Gradient Method with Momentum

The first modification is to provide the algorithm with some memory of its previous update steps, such that consecutive updates in the same direction with respect to any input parameter cause the method to accelerate in that direction. This can be expressed mathematically as follows:

$$\Delta\mathbf{x}_i = \gamma\,\mathbf{G}_i + \mu\,\Delta\mathbf{x}_{i-1} \tag{6}$$

where both γ and μ are typically less than unity and $\mathbf{G}_i$ is given by Equation (5).

Tunable parameters: γ, μ

In the framework of a ball rolling into a valley, it can be seen as analogous to the inertia (or momentum) of the ball, which allows the ball to continue moving at a high speed even though the ground flattens out near the bottom of the valley, hence the name "with momentum". Thus, the parameter μ can be seen to control the degree of inertia of the optimization algorithm and is also known as the *momentum factor*.

As a result of this interpretation, it is sufficient to use the following value for the first step, ie. $i = 0$:

$$\Delta\mathbf{x}_{-1} = \mathbf{0} \tag{7}$$

1.3 Nesterov-Accelerated Gradient with Momentum

As the momentum modification effectively increases the step size for consecutive steps in the same direction, an additional modification would be to simply use the accelerated update step as a predictor and correcting it based on the gradient calculated at the predicted location. This algorithm, often shortened to *NAG*, was developed and proven to have a higher convergence speed by Yurii Nesterov, hence the name, and can be expressed as follows:

$$\Delta\mathbf{x}_i = \gamma\,\mathbf{N}_i + \mu\,\Delta\mathbf{x}_{i-1} \tag{8}$$

where both γ and μ are typically less than unity and

$$\mathbf{N}_i \equiv \nabla_{\mathbf{x}} f(\mathbf{x}_i + \mu\,\Delta\mathbf{x}_{i-1}) \tag{9}$$

Tunable parameters: γ, μ

Due to the predictor-corrector nature of the algorithm, it is recommended to use the following value for the first step, ie. $i = 0$, for reasons of self-consistency:

$$\Delta\mathbf{x}_{-1} = \nabla_{\mathbf{x}} f(\mathbf{x}_0) \tag{10}$$

which requires an additional evaluation of the gradient at the location of the initial guess, $\mathbf{x}_0$, that is not strictly defined by the algorithm.

1.4 Adaptive Gradient Method

Another modification is to provide the algorithm with a way to autonomously adjust the learning rate, γ, such that a more intelligent approach path to the solution can be determined. One way to provide this capability, resulting in the method known as *AdaGrad*, is by using the following:

$$\Delta\mathbf{x}_i = \gamma \circ (\mathbf{\Gamma}_i + \varepsilon)^{\circ -1/2} \circ \mathbf{G}_i \tag{11}$$

where γ is typically less than unity, $\mathbf{G}_i$ is given by Equation (5), $\varepsilon \sim 10^{-8}$ is provided simply to avoid division by zero errors in the algorithm, $\circ$ is the *Hadamard product* denoting an element-wise operation on vector quantities and

$$\mathbf{\Gamma}_i = \sum_{j=0}^{i} \mathbf{G}_j^{\circ 2} \tag{12}$$

Tunable parameters: γ

This method assumes that the solution will be approached if a sufficient number of steps are taken and effectively reduces the learning rate, γ, as the number of steps increases. In practice, this allows for a higher initial learning rate, which is advantageous as larger steps can be taken in the beginning of the search, where the current guess is furthest from the solution. However, this algorithm suffers from the fact that $\boldsymbol{\Gamma}_i$ is completely unconstrained and can quickly hit the overflow limits of a machine if the gradients are large or if a sufficiently large number of steps have already been taken, due to the squaring operation in Equation (12). It also suffers a performance penalty if the region in which the solution resides is not approached quickly enough, as $\gamma/\boldsymbol{\Gamma}_i \to \mathbf{0}$ as $i \to \infty$.

1.5 Adaptive Gradient Method with Decay

In order to constrain this rampantly accumulating factor, an additional modification would be to limit the ability of the algorithm to remember its previous updates, such that the gradients from substantially older updates no longer impact the current step estimation. This method, also known as *RMSProp*, can be expressed mathematically as follows:

$$\Delta \mathbf{x}_i = \gamma \circ \left[\mathbf{E}(\mathbf{G}_i) + \varepsilon\right]^{\circ -1/2} \circ \mathbf{G}_i \tag{13}$$

where γ is typically less than unity, $\varepsilon \sim 10^{-8}$ is provided simply to avoid division by zero errors in the algorithm, $\mathbf{G}_i$ is given by Equation (5), $\circ$ is the Hadamard product and

$$\mathbf{E}(\mathbf{Z}_i) = \beta\, \mathbf{E}(\mathbf{Z}_{i-1}) + (1-\beta)\, \mathbf{Z}_i^{\circ 2} \tag{14}$$

where $\beta \in [0,1]$ is known as the *forgetting factor* due to its influence on how much previous information is retained by the algorithm.

Tunable parameters: γ, β

It should be noted that another similar development of this method, called *AdaDelta*, replaces the learning rate parameter, γ, with the following normalization factor:

$$\gamma \longrightarrow \left[\mathbf{E}(\Delta \mathbf{x}_{i-1}) + \varepsilon\right]^{\circ 1/2} \tag{15}$$

where $\varepsilon \sim 10^{-8}$ and $\mathbf{E}$ is the function provided in Equation (14). This replacement was justified as an attempt to reconcile the problem of mismatched units between $\mathbf{x}_i$ and $\Delta \mathbf{x}_i$, but, in practice, it simply removes one tunable parameter from the method and serves to speed up the process of optimizing the method for a given application.

1.6 Adaptive Moment Estimation

It is also possible to formulate a method which combines the retention of both the gradients, as in Section 1.2, and the square of the gradients, as in Section 1.4. By doing so, the following method, known as *Adam*, can be found:

$$\Delta \mathbf{x}_i = \gamma \circ \left[\widehat{\mathbf{V}}_i^{\circ 1/2} + \varepsilon\right]^{\circ -1} \circ \widehat{\mathbf{M}}_i \tag{16}$$

where γ is typically less than unity, $\varepsilon \sim 10^{-8}$ is provided simply to avoid division by zero errors in the algorithm, $\circ$ is the Hadamard product and

$$\begin{aligned} \widehat{\mathbf{M}}_i &= \frac{1}{1-\beta_1^i}\mathbf{M}_i \quad , \qquad \mathbf{M}_i = \beta_1 \mathbf{M}_{i-1} + (1-\beta_1)\,\mathbf{G}_i \\ \widehat{\mathbf{V}}_i &= \frac{1}{1-\beta_2^i}\mathbf{V}_i \quad , \qquad \mathbf{V}_i = \beta_2 \mathbf{V}_{i-1} + (1-\beta_2)\,\mathbf{G}_i^{\circ 2} \end{aligned} \tag{17}$$

where $\beta_1 \in [0,1]$ and $\beta_2 \in [0,1]$ are also known as forgetting factors and $\mathbf{G}_i$ is given by Equation (5).

Tunable parameters: γ, β_1, β_2

This method is called "moment estimation", as it is possible to interpret this algorithm as an estimation of the probability distribution of the gradient, $\mathbf{G}_i$, via the *method of moments*, based on the "samples" acquired in previous steps. Once the distribution is estimated, the step size, $\Delta\mathbf{x}_i$, is then made to be proportional to the weighted mean of this distribution or the first moment of the gradient, represented by $\mathbf{M}_i$. Additionally, the learning rate, γ, is adjusted with the inverse square root of the weighted variance of this distribution or the second moment of the gradient, represented by $\mathbf{V}_i$, as it provides a measure of how quickly the gradients are changing with each consecutive step. As the method of moments estimation procedure tends to introduce a bias between the "sampled" moments and the true moments of the distribution, a *bias-correction factor* is introduced to the moments, via the calculation of $\widehat{\mathbf{M}}_i$ and $\widehat{\mathbf{V}}_i$, in order to ensure that the algorithm will converge to the correct answer.

Another interpretation of this algorithm can be made by introducing the *l_p-norm*, or *p^{th}-order vector norm*, defined as follows:

$$|\mathbf{z}|_p = \frac{1}{n}\left(\sum_{i=0}^{n} z_i^p\right)^{1/p} \tag{18}$$

where n is the number of entries in the vector, $\mathbf{z}$.

From this definition, the algorithm can be seen as attempting to pick steps which minimize a weighted *l_2-norm*, represented by $\mathbf{V}_i$, where the $1/n$-factor in Equation (18) is replaced by the forgetting factor, β. As the terms in the sum are raised to a power, a strong penalty is applied to steps in directions which dramatically increase this value. Thus, the algorithm tends to move towards the regions where the gradients are zero with fewer iterations. This conveniently turns out to be the desired behaviour as the solutions to the optimization problem have a gradient of zero, as defined by Equation (2).

1.7 Adaptive Moment Estimation with l_∞

By taking the l_2-norm interpretation of the Adam method, an additional modification could be to use a high-order vector norm, such as l_3- or l_4-norm, as the summation tracks higher powers of the gradient and, thus, penalizes steps in poor directions even more harshly. However, it was found that these higher powers also cause the algorithm to become numerically unstable due overflow errors or catastrophic cancellation. Fortunately, a simplification of the l_∞*-norm* eliminates the necessity of using exponents at all, allowing for its use to formulate a new method, also known as *AdaMax*, as follows:

$$\Delta\mathbf{x}_i = \gamma \circ \mathbf{U}_i^{\circ -1} \circ \widehat{\mathbf{M}}_i \tag{19}$$

where γ is typically less than unity, $\circ$ is the Hadamard product, $\widehat{\mathbf{M}}_i$ is given by Equation (17) and

$$\mathbf{U}_i = \max \circ \left(\beta_2 \mathbf{V}_i, |\mathbf{G}_i|\right) \tag{20}$$

where $\beta_1 \in [0,1]$, $\beta_2 \in [0,1]$, $\mathbf{G}_i$ is given by Equation (5), $\mathbf{V}_i$ is given by Equation (17), and the notation $\max \circ (...)$ indicates that the maximum between the bracketed values is determined in an element-wise fashion.

Tunable parameters: γ, β_1, β_2

It should be noted that concrete evidence that this method is superior to the Adam method, discussed in Section 1.6, has not yet been produced.

1.8 Nesterov-Accelerated Adaptive Moment Estimation

In a similar fashion as described in Section 1.3, it is possible to apply the Nesterov acceleration scheme to the Adam method, discussed in Section 1.6. The resulting method, also known as the *Nadam* method, can be expressed as follows:

$$\Delta\mathbf{x}_i = \gamma \circ \left[\widehat{\mathbf{V}}_i^{\circ 1/2} + \varepsilon\right]^{\circ -1} \circ \left[\beta_1 \widehat{\mathbf{M}}_i + \frac{1-\beta_1}{1-\beta_1^{i-1}} \mathbf{G}_i\right] \tag{21}$$

where γ is typically less than unity, $\beta_1 \in [0,1]$, $\beta_2 \in [0,1]$, $\varepsilon \sim 10^{-8}$ is provided simply to avoid division by zero errors in the algorithm, $\circ$ is the Hadamard product, $\mathbf{G}_i$ is given by Equation (5), and $\widehat{\mathbf{M}}_i$ and $\widehat{\mathbf{V}}_i$ are given by Equation (17).

Tunable parameters: γ, β_1, β_2

It should be noted that this formulation of the Nadam method is only mathematically valid when $\beta_2 > 0.9$, due to an assumption made in order to apply the Nesterov acceleration. Although the Nadam method has been shown to perform better than the Adam algorithm, discussed in Section 1.6, on equivalent applications, this restriction on the β_2 parameter makes the Adam method more generally applicable.

1.9 Additional Notes

Overall, it is generally accepted that the Adam method, described in Section 1.6, is the superior choice out of all the methods discussed in this document. However, it should be mentioned that some sources also quote the NAG or Nadam method, described respectively in Section 1.3 and 1.8, to be equal or better than the Adam method, depending on the specific application.

Table 1 shows the recommended parameter values for each optimization method discussed in this document. However, it should be noted that they are recommendations based solely on empirical findings and may not provide useful results for every application. As such, each application should thoroughly test and re-adjust them as necessary.

Method Name	Recommended Parameter Values
Gradient	$\gamma = 10^{-5}$
Momentum	$\gamma = 10^{-4},\ \mu = 0.9$
NAG	$\gamma = 10^{-4},\ \mu = 0.9$
AdaGrad	$\gamma = 10^{-2},\ \varepsilon = 10^{-8}$
RMSProp	$\gamma = 10^{-3},\ \beta = 0.9,\ \varepsilon = 10^{-8}$
AdaDelta	$\beta = 0.9,\ \varepsilon = 10^{-8}$
Adam	$\gamma = 10^{-3},\ \beta_1 = 0.9,\ \beta_2 = 0.999,\ \varepsilon = 10^{-8}$
AdaMax	$\gamma = 10^{-3},\ \beta_1 = 0.9,\ \beta_2 = 0.999$
Nadam	$\gamma = 10^{-3},\ \beta_1 = 0.9,\ \beta_2 = 0.999,\ \varepsilon = 10^{-8}$

Table 1: Recommended parameter values for the various gradient-based optimization methods