Gradient Descent Efficiency Index

Aviral Dhingra
aviral.dhingra.2008@gmail.com

Abstract

Gradient descent is a widely used iterative algorithm for finding local minima in multivariate functions. However, the final iterations often either overshoot the minima or make minimal progress, making it challenging to determine an optimal stopping point. This study introduces a new efficiency metric, E_k, designed to quantify the effectiveness of each iteration. The proposed metric accounts for both the relative change in error and the stability of the loss function across iterations. This measure is particularly valuable in resource-constrained environments, where costs are closely tied to training time. Experimental validation across multiple datasets and models demonstrates that E_k provides valuable insights into the convergence behavior of gradient descent, complementing traditional performance metrics. The index has the potential to guide more informed decisions in the selection and tuning of optimization algorithms in machine learning applications and be used to compare the "effectiveness" of models relative to each other.

1 Introduction

In the field of machine learning, optimizing the training process of models is crucial for achieving high performance while minimizing computational resources. Gradient descent [1] is a widely used optimization algorithm due to its simplicity and effectiveness in finding local minima of differentiable functions. However, the efficiency of gradient descent can diminish with large datasets and prolonged training periods, where additional iterations provide negligible improvements. This raises the need for a robust mechanism to identify the optimal stopping point, ensuring efficient use of computational resources.

Fig. 1 is a contour plot that illustrates how each step taken in gradient descent towards a local minimum is smaller than the previous one. This approach quickly returns diminishing results, making the last few steps cost more computationally than they yield in accuracy.

In Fig. 2, notice how the first 100 steps exponentially decrease (or logarithmically increase) the cost. However, if you zoom out to 100,000 steps, the curve effectively flattens out before 10,000 steps in this particular example.

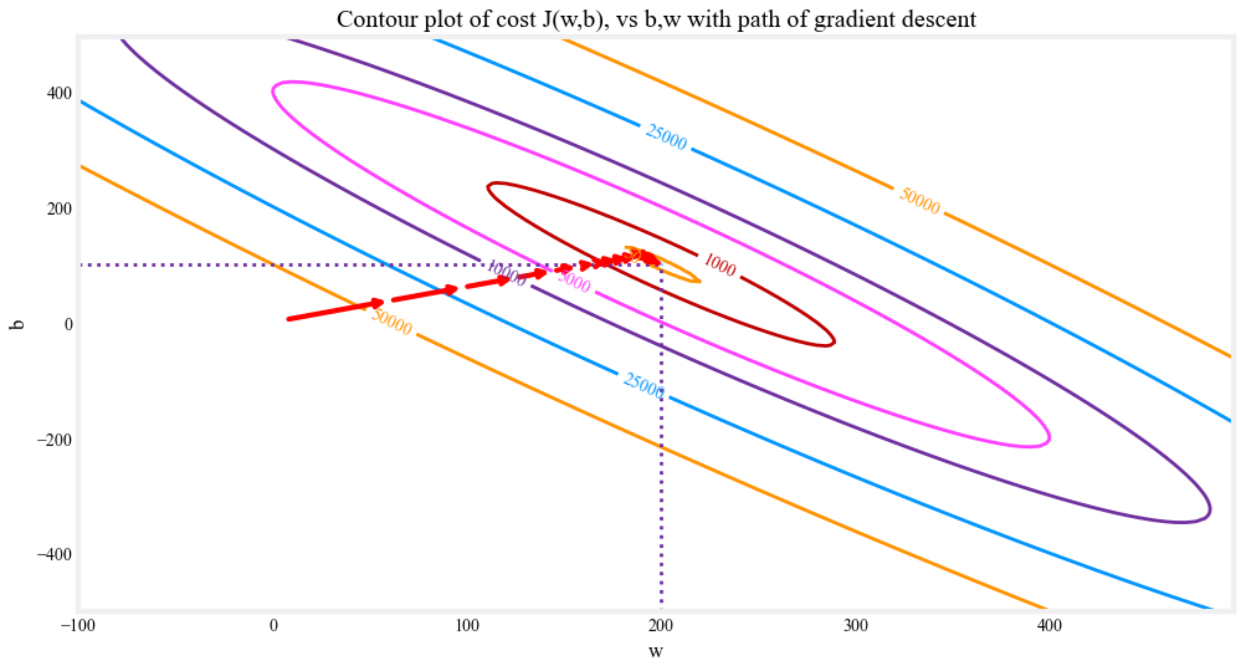

Fig. 1. Contour plot of cost b,w with path of gradient descent $J(w,b)$.

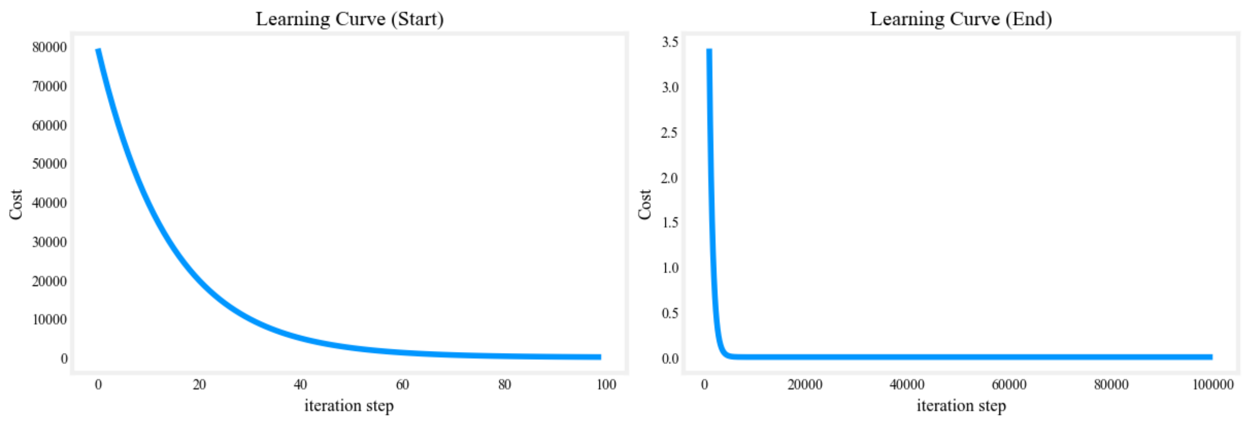

Fig. 2. Comparison of cost with different domain restrictions.

The “Gradient Descent Efficiency Index” is a novel ratio between training parameters that includes the relative change in gradient norm, the initial learning rate, the learning decay rate, absolute change in coefficients, and the number of iterations.

Note that throughout this paper, I will use the name “Gradient Descent Efficiency Index”, the short form “GDEI”, and the function E_k interchangeably.

2 Related Work

2.1 Momentum

Momentum [2] helps accelerate gradient vectors in the right directions, thus leading to faster convergence.

$$v_t = \beta v_{t-1} + (1-\beta) g_t$$
$$\theta_{t+1} = \theta_t - \alpha v_t$$

where:

- v_t is the velocity vector.
- β is the momentum hyperparameter.
- α is the learning rate.
- g_t is the gradient at time step t.

2.2 AdaGrad

AdaGrad [3] adapts the learning rate for each parameter based on the past gradients.

$$G_t = \sum_{\tau=1}^{t} g_\tau^2$$
$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} g_t$$

where:

- G_t is the sum of the squares of the past gradients.
- α is the global learning rate.
- ϵ is a small constant to prevent division by zero.
- g_t is the gradient at time step t.

2.3 RMSProp

RMSProp [4] (Root Mean Square Propagation) adjusts the learning rate for each parameter.

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

where:

- $E[g^2]_t$ is the exponentially decaying average of past squared gradients.
- β is the decay rate.
- α is the learning rate.
- ϵ is a small constant to prevent division by zero.
- g_t is the gradient at time step t.

2.4 Adam

Adam (Adap) [5] combines the advantages of both AdaGrad and RMSProp.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
$$\theta_{t+1} = \theta_t - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where:

- m_t and v_t are the first and second moment estimates, respectively.
- β_1 and β_2 are hyperparameters for the decay rates.
- α is the learning rate.
- ϵ is a small constant to prevent division by zero.
- g_t is the gradient at time step t.

2.5 Nesterov Accelerated Gradient (NAG)

Nesterov Accelerated Gradient (NAG) [6] is a variation of the momentum method that anticipates the future position of the parameters. Unlike standard momentum, which calculates the gradient at the current position, NAG first makes a big jump in the direction of the accumulated gradient and then measures the gradient. The update rules are:

$$v_{k+1} = \gamma v_k + \eta \nabla L(\theta_k - \gamma v_k)$$
$$\theta_{k+1} = \theta_k - v_{k+1}$$

NAG often results in faster convergence compared to standard momentum, especially in settings with high curvature.

2.6 Other Algorithms and Variants

In addition to the widely used methods mentioned above, numerous other algorithms and variants have been proposed to address specific challenges in gradient-based optimization[7]. These include:

- **SGD with Warm Restarts:** A variant of stochastic gradient descent (SGD) that periodically restarts with a large learning rate to escape local minima.
- **AdaMax:** An extension of Adam that uses the infinity norm instead of the second moment, making it more robust in certain applications.
- **AMSGrad:** A modification of Adam that aims to improve its convergence properties by fixing a flaw in the original algorithm.
- **Nadam:** Combines the Nesterov Accelerated Gradient with Adam, offering a blend of adaptive learning rates and look-ahead gradient calculation.

Each of these algorithms contributes uniquely to the field of optimization, providing various trade-offs in terms of convergence speed, stability, and computational cost. However, a common challenge across all these methods is the lack of a standardized metric to measure the efficiency of each iteration. The efficiency score E_k proposed in this paper aims to address this gap, offering a more detailed perspective on the performance of gradient descent.

3 Derivation of GDEI

3.1 External Parameters

3.1.1 Mean Squared Error (MSE)

The error metric used is the Mean Squared Error (MSE) [8]. It is defined as:

$$E = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where:

- n is the number of data points.
- y_i is the actual value of the i-th data point.
- $\hat{y}_i$ is the predicted value of the i-th data point.

This formula calculates the average of the squared differences between the actual and predicted values, providing a measure of the quality of the model's predictions.

3.1.2 Proportion of Initial Loss Reduced (P_k)

The parameter P_k represents the proportion of the initial error that has been reduced by the k-th iteration. It is defined as:

$$P_k = \frac{L_{\text{initial}} - L_k}{L_{\text{initial}}}$$

where:

- L_{initial} is the initial mean squared error at the beginning of the optimization process.
- L_k is the current mean squared error at the k-th iteration.

This term quantifies the effectiveness of each iteration in reducing the error, with a higher P_k indicating greater progress towards minimizing the loss function.

3.1.3 Absolute Change in Loss Function (Δ_k)

The parameter Δ_k denotes the absolute change in the loss function, which is the MSE, between consecutive iterations. It is defined as:

$$\Delta_k = |L_{k-1} - L_k|$$

where:

- L_{k-1} is the mean squared error at the $(k-1)$-th iteration.
- L_k is the mean squared error at the k-th iteration.

This term captures the stability of the optimization process. Large values of Δ_k suggest instability, which can indicate inefficient steps or overly aggressive learning rates.

3.2 Formula & Theoretical Explanation

The efficiency score E_k for the k-th iteration of gradient descent is a metric designed to evaluate the effectiveness of each iteration in reducing the loss function while also considering the stability of the optimization process. The score is given by:

$$E_k = 100 - 1 \min\left(100, \max\left(1, \frac{100 \times P_k}{1 + \log\left(1 + \Delta_k^2\right)}\right)\right)$$

Let's break down and explain the components of this formula:

3.2.1 Logarithmic Term $\log\left(1 + \Delta_k^2\right)$

The term $\log\left(1 + \Delta_k^2\right)$ serves to dampen the impact of large changes in the loss function.

- The logarithm $\log(x)$ grows slowly as x increases, so applying it to $1 + \Delta_k^2$ helps moderate large values of Δ_k^2. This means that even if Δ_k^2 is large, its effect on the efficiency score E_k will not be excessively amplified.
- Adding 1 inside the logarithm ensures that the term $\log\left(1 + \Delta_k^2\right)$ remains positive, preventing any undefined or negative logarithmic values.
- Δ_k^2 (the squared difference between successive errors) reflects the stability of the optimization. If the error changes drastically between iterations (i.e., Δ_k is large), the logarithmic term will increase, thereby reducing the efficiency score.

3.2.2 The Term $1 + \log\left(1 + \Delta_k^2\right)$ in the Denominator

The purpose of placing $1 + \log\left(1 + \Delta_k^2\right)$ in the denominator is to penalize instability in the optimization process.

- When the change in the loss function (Δ_k) is small, $\log\left(1 + \Delta_k^2\right)$ will also be small, meaning the efficiency score E_k will not be heavily penalized.
- Conversely, if the loss function change is large, the logarithmic term will increase, leading to a larger denominator and thus a lower efficiency score E_k. This reduction reflects the inefficiency introduced by instability in the optimization process.

3.2.3 Proportional Term $100 \times P_k$ in the Numerator

The term $100 \times P_k$ in the numerator represents the proportion of the initial error that has been reduced.

- P_k is the fraction of the initial error that has been reduced by the k-th iteration, and multiplying it by 100 converts this fraction into a percentage.

- This term rewards the optimization process for effectively reducing the error. The larger the error reduction P_k, the higher the numerator, and consequently, the higher the efficiency score E_k.

3.2.4 The Role of $\min(100, \cdot)$ and $\max(1, \cdot)$

The $\min(100, \cdot)$ and $\max(1, \cdot)$ functions ensure that the efficiency score E_k stays within a reasonable and interpretable range.

- $\max(1, \cdot)$ ensures that the efficiency score does not drop below 1, which prevents the score from becoming too punitive, especially when the change in loss function Δ_k is very large.
- $\min(100, \cdot)$ caps the efficiency score at 100, indicating that a score of 100 is the maximum achievable, representing an ideal iteration where error reduction is perfect and stability is maintained.

3.3 Final Function

The efficiency score E_k for the k-th iteration of gradient descent is defined to quantify the effectiveness of each iteration in reducing the loss function while accounting for the stability of the optimization process. The score is given by:

$$E_k = 100 - \min\left(100, \max\left(1, \frac{100 \times P_k}{1 + \log\left(1 + \Delta_k^2\right)}\right)\right)$$

Substituting the definitions of P_k and Δ_k into the formula:

$$E_k = 100 - \min\left(100, \max\left(1, \frac{100 \times \frac{L_{\text{initial}} - L_k}{L_{\text{initial}}}}{1 + \log\left(1 + (L_{k-1} - L_k)^2\right)}\right)\right)$$

Next, simplifying the fraction:

$$E_k = 100 - \min\left(100, \max\left(1, \frac{100 \times (L_{\text{initial}} - L_k)}{L_{\text{initial}} \times \left(1 + \log\left(1 + (L_{k-1} - L_k)^2\right)\right)}\right)\right)$$

4 Experimental Validation

4.1 Assumptions and Standards

The experiments were conducted under the following assumptions:

- The loss function being used is "Mean Squared Error"

- The learning rate and other hyperparameters are fixed during each individual experiment and between experiments if relative efficiency is to be taken into account
- The datasets used are representative of common machine learning tasks
- Standard practices in machine learning, such as using a consistent validation set to monitor performance and ensuring reproducibility through fixed random seeds

4.2 Assumptions and Standards

The `generate_data` function used in this paper produces a synthetic dataset suitable for testing linear regression models and gradient descent optimization techniques.

4.2.1 Description of the Generated Data

- **Features** (X):
  - An $n \times m$ matrix X, where each element is a random value in the range $[0, 2)$.
  - n is the number of samples, and m is the number of features.
  - The features are generated independently and uniformly at random.
- **Labels** (y):
  - A vector $y \in \mathbb{R}^n$ where each label is generated based on a linear relationship with the first feature of X, plus some Gaussian noise.
  - The label for the i-th sample is computed as:

    $$y_i = 4 + 3 \cdot x_{i1} + \epsilon_i$$

    where:
    - x_{i1} is the first feature of the i-th sample.
    - $\epsilon_i \sim \mathcal{N}(0, 1)$ is a random noise term drawn from a standard normal distribution.

This dataset models a simple linear relationship between the target variable y and the first feature of X, with added noise to simulate real-world data variability. The remaining features in X are irrelevant to y, providing a scenario to test feature selection and model robustness.

4.2.2 Suitability for Gradient Descent Optimization

Using synthetically generated data like this is highly beneficial for testing gradient descent algorithms due to the following reasons:

- **Known Underlying Model**: Since the true relationship between X and y is known, one can easily assess how well the optimization algorithm recovers the underlying parameters (intercept and slope).
- **Controlled Noise**: The addition of Gaussian noise allows testing the algorithm's ability to handle data imperfections, which are common in real datasets.
- **Feature Irrelevance**: Including multiple features where only one is relevant tests the algorithm's capacity to identify and focus on significant predictors.
- **Scalability Testing**: By adjusting n and m, one can examine the performance and scalability of gradient descent under different dataset sizes.

Overall, this synthetic dataset provides a controlled environment to develop, debug, and validate gradient descent optimization techniques before applying them to more complex real-world data.

4.3 Implementation

The following function, presented in pseudocode, serves as an example of how to calculate the GDEI in real-time. The objective is to illustrate practical applications of the index and inspire the reader with potential use cases.

```
Initialize theta (weights and bias) randomly
Add bias term to input features X_b

SET initial_error = None
Initialize cost_history and efficiency_history

FOR iteration FROM 1 TO n_iterations:
    Predict y_pred using X_b and theta
    Calculate error = y_pred - y

    Compute gradients based on error
    Update theta = theta - (learning_rate * gradients)

    Calculate current cost (MSE)
    Store current cost in cost_history

    IF first iteration:
        SET initial_error = current cost
        CONTINUE

    SET prev_cost = cost from previous iteration
    Calculate efficiency using calculate_efficiency(initial_error, cost, prev_cost)
    Store efficiency in efficiency_history

    Update learning_rate = learning_rate * decay_rate
```

4.4 Plotting the Index

Fig. 3 tracks the live efficiency of a randomly generated dataset for linear regression that has an approximate linear curve with noise. The efficiency is between a range of 1 and 100. As show it decreases in each iteration and the rate of change of efficiency (i.e. $\mathrm{d}E_k/\mathrm{dk}$) also retards as the number of iterations progress.

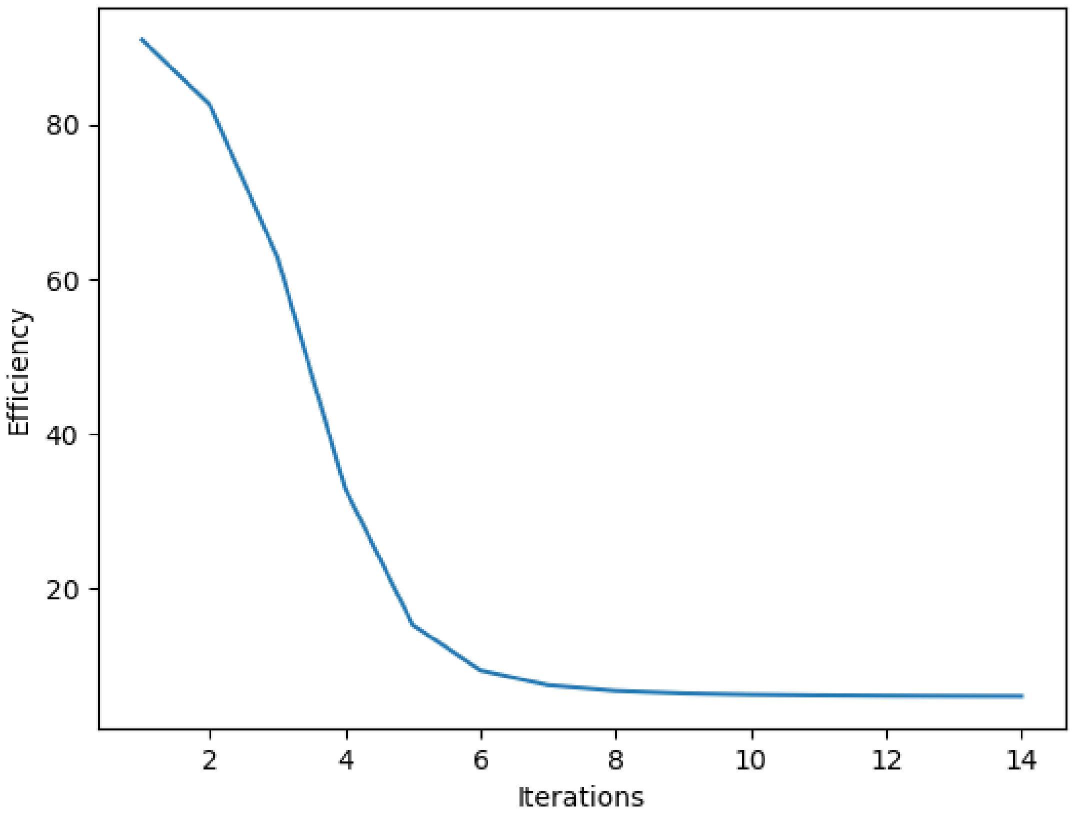

Fig. 3. Line graph of efficiency with respect to iterations in a sample dataset.

5 Conclusion

This paper introduced a novel efficiency score E_k for evaluating the effectiveness of each gradient descent iteration. The proposed index captures both the relative reduction in error and the stability of the loss function, offering a more nuanced assessment of gradient descent performance compared to traditional metrics. By providing a finer-grained measure of efficiency, E_k enables more informed decisions in the selection and adjustment of optimization techniques in machine learning.

The experimental validation demonstrates that E_k is a robust metric that correlates well with final accuracy and can highlight inefficiencies in the optimization process. Future work will focus on extending the experimental validation of E_k across a broader range of optimization algorithms, including those with adaptive learning rates and second-order methods. Additionally, the application of E_k to other optimization problems, such as those involving non-smooth or stochastic loss functions, will be explored. The idea of a "golden number of iterations" or a function that produces the same given a set of parameters can also be explored with this index.

References

[1] Cauchy, A. (1847). Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes Rendus*, *25*(2), 536-538.

[2] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, *4*(5), 1-17.

[3] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*, 2121-2159.

[4] Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, *4*(2), 26-31.

[5] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[6] Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. *Doklady AN SSSR*, 269(3), 543-547.

[7] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[8] Legendre, A. M. (1805). Nouvelles méthodes pour la détermination des orbites des comètes. *Paris: F. Didot*.