Journal of Big Data

RESEARCH Open Access

From big data to smart data: a sample gradient descent approach for machine learning

Aadil Gani Ganie[1*] and Samad Dadvandipour[1]

*Correspondence: ganie.aadil.gani@student.uni-miskolc.hu

[1] University of Miskolc, Miskolc 3515, Hungary

Abstract

This research paper presents an innovative approach to gradient descent known as "Sample Gradient Descent". This method is a modification of the conventional batch gradient descent algorithm, which is often associated with space and time complexity issues. The proposed approach involves the selection of a representative sample of data, which is subsequently subjected to batch gradient descent. The selection of this sample is a crucial task, as it must accurately represent the entire dataset. To achieve this, the study employs the use of Principle Component Analysis (PCA), which is applied to the training data, with a condition that only those rows and columns of data that explain 90% of the overall variance are retained. This approach results in a convex loss function, where a global minimum can be readily attained. Our results indicate that the proposed method offers faster convergence rates, with reduced computation times, when compared to the conventional batch gradient descent algorithm. These findings demonstrate the potential utility of the "Sample Gradient Descent" technique in various domains, ranging from machine learning to optimization problems. In our experiments, both approaches were run for 30 epochs, with each epoch taking approximately 3.41 s. Notably, our "Sample Gradient Descent" approach exhibited remarkable performance, converging in just 8 epochs, while the conventional batch gradient descent algorithm required 20 epochs to achieve convergence. This substantial difference in convergence rates, along with reduced computation times, highlights the superior efficiency of our proposed method. These findings underscore the potential utility of the "Sample Gradient Descent" technique across diverse domains, ranging from machine learning to optimization problems. The significant improvements in convergence rates and computation times make our algorithm particularly appealing to practitioners and researchers seeking enhanced efficiency in gradient descent optimization.

Keywords: Big data, Gradient decent, Machine learning, PCA, Loss function

Introduction

In recent years, the field of machine learning has witnessed a rapid proliferation of research endeavors, leading to the development of new algorithms and techniques [20–22]. Among these, the gradient descent algorithm stands as one of the foundational tools for optimizing machine learning models by minimizing their associated loss functions. However, the traditional batch gradient descent algorithm often grapples with various

challenges, particularly in the context of large datasets. These challenges encompass high computational complexity and sluggish convergence rates, posing significant hurdles for efficient model training.

To address these limitations, researchers have introduced numerous variants and enhancements to the gradient descent algorithm [13, 23, 24]. One notable variant is the sample gradient descent (SGD) algorithm, which deviates from the conventional approach by utilizing a random subset of data samples to compute gradients [15]. This modification aims to expedite convergence, reduce computational complexity, and enhance the model's generalization capability. Another influential optimization technique is the Adam optimizer, which employs adaptive learning rates and momentum to expedite convergence in deep learning applications [21].

Despite these advancements, there persists a need for novel approaches that can further ameliorate the efficiency and effectiveness of gradient descent-based optimization algorithms. This research paper introduces a novel modification of the batch gradient descent algorithm, termed "Sample Gradient Descent," which harnesses the power of Principal Component Analysis (PCA) to augment the conventional sampling strategy. Our proposed approach leverages PCA to select a representative subset of data from the original dataset, ensuring that it adequately captures the dataset's essential characteristics [16]. Specifically, PCA retains only those rows and columns of data that collectively explain a predetermined percentage of the dataset's overall variance, typically set at 90%. This judicious selection of data results in a more efficient and informative sample for subsequent optimization. The application of PCA results in a convex loss function, providing an advantageous landscape where a global minimum can be reached more efficiently [17].

The primary aim of this research endeavor is to examine the efficacy of the "Sample Gradient Descent" technique in comparison to traditional batch gradient descent and other optimization algorithms. We systematically evaluate the convergence rates, computational times, and overall model performance, showcasing the potential utility of our approach in a wide array of domains, spanning from machine learning to optimization problems [18, 19]. In the following sections of this paper, we provide a comprehensive overview of gradient descent algorithms, including traditional batch gradient descent and recent variants. Subsequently, we introduce our novel approach, describe its inner workings, and present the results of extensive experiments. Finally, we conclude with a discussion of the implications of our findings and outline potential avenues for future research.

Literature review

The gradient descent (GD) is a widely used optimization algorithm in machine learning [1]. It is an iterative method that minimizes a cost function by finding the steepest descent direction. GD has two main variants, batch gradient descent (BGD) and stochastic gradient descent (SGD). BGD updates the parameters using the average of all the training examples while SGD updates the parameters using a single training example at a time. However, both BGD and SGD have their limitations. BGD requires all the training data to be loaded into memory, which can be computationally expensive for large datasets. SGD can be unstable and may converge to a suboptimal solution

[1]. To overcome the limitations of BGD and SGD, several new optimization techniques have been proposed in the literature. One of these techniques is the mini-batch gradient descent (MBGD), which is a compromise between BGD and SGD. MBGD updates the parameters using a small random sample of training examples (a mini-batch) at a time. This approach reduces the memory requirements of BGD and the instability of SGD [2].

Another optimization technique that has gained popularity in recent years is the Adam optimizer [3]. The Adam optimizer is a variant of stochastic gradient descent that uses adaptive learning rates for each parameter. It also incorporates momentum to speed up convergence. The Adam optimizer has shown to be effective in many deep learning applications [3]. In addition to these optimization techniques, several researchers have proposed modifications to the gradient descent algorithm. For example, the Nesterov accelerated gradient (NAG) descent uses an accelerated gradient method to estimate the gradient at the next step [5]. This modification has been shown to improve convergence rates compared to standard GD [7]. Another modification of GD is the conjugate gradient (CG) descent, which uses conjugate directions to estimate the gradient [6]. CG has been shown to converge faster than standard GD for certain types of problems [6].

The use of second-order methods, such as the Hessian matrix, has also been proposed to optimize the cost function. However, these methods are computationally expensive and are not widely used in practice [2]. Apart from optimization techniques, several studies have focused on the choice of activation functions. The most commonly used activation functions are sigmoid and ReLU (rectified linear unit). Sigmoid functions are smooth and differentiable but suffer from the vanishing gradient problem, while ReLU functions are non-smooth but do not suffer from the vanishing gradient problem [2]. Recent studies have also focused on the use of convolutional neural networks (CNNs) for image recognition [4]. CNNs use a hierarchical architecture that learns features at multiple levels of abstraction. They have shown to be effective in various image recognition tasks, such as object recognition and segmentation [4]. In addition to optimization techniques and activation functions, several studies have investigated regularization techniques. Regularization is used to prevent overfitting by adding a penalty term to the cost function. Common regularization techniques include L1 and L2 regularization, dropout, and early stopping [2].

Proposed methodology

The present research endeavor entails the introduction of an innovative rendition of the gradient descent technique, namely the "Sample Gradient Descent". This novel approach represents a modification of the conventional batch gradient descent algorithm, which necessitates the simultaneous processing of all available data. This approach results in complexities associated with space and time, rendering the attainment of the global minimum computationally demanding. Conversely, the stochastic gradient descent algorithm processes data on an individual basis, which can be suboptimal in certain circumstances. Our method involves the selection of a representative sample (n) of data, which is subsequently subjected to batch gradient descent. The selection of an appropriate sample is of paramount importance, as it must accurately depict the entire dataset (N). To achieve this, we leverage the utility of Principle Component Analysis (PCA), which is applied to the training data, with a stipulation that only those rows and columns of data

that explain 90% of the overall variance are retained. Notably, this parameter is amenable to tuning and is deemed a hyperparameter. The optimal value of this parameter can be determined through the use of grid search or random search techniques. Mathematically, the calculation of y in regression can be expressed as:

$$y = m * x + b \tag{1}$$

the update rule for coefficients is

$$y_{new} - N * Slope \tag{2}$$

To calculate the slope we differentiate the loss functions with respect to b.

$$L = \sum (yi - y)^2 \tag{3}$$

$$df/db = d\big(\Sigma\big(y_i - \mathrm{y}\big)2\big)/dl \tag{4}$$

$$df/db = d\big(\Sigma y_i - m * x_i\big)2/dl \tag{5}$$

$$dl/db = 2\Sigma y_i - m * x_i - b(-1) \tag{6}$$

$$dl/db = -2\Sigma\big(y_i - m * x_i - b\big) \tag{7}$$

This is the equation for slope at b.

$$\text{Equationforslopeatm} = dl/db = 2\Sigma\big(y_i - m * x_i - b\big)(x_i) \tag{8}$$

The algorithm for the above mathematical implementation of gradient decent as follows:

Algorithm: GDRegressor.

1. Initialize:
   - Set self.m = 100
   - Set self.b = −120
   - Set self.lr as the learning rate
   - Set self.epochs as the number of epochs
2. Fit(self, X, y):
   - For each epoch from 1 to self.epochs:

Calculate the loss slope with respect to b:
Set loss_slope_b = − 2 * sum(y − self.m*X.ravel() − self.b)
Calculate the loss slope with respect to m:
Set loss_slope_b = − 2 * sum((y − self.m*X.ravel() − self.b) * X.ravel())
Update the value of b using gradient descent:
Set self.b = self.b—(self.lr * loss_slope_b).
Update the value of m using gradient descent:

Set self.m = self.m - (self.lr * loss_slope_m)

- Print the learned values of m and b

3. Predict(self, X):
   - Return self.m * X + self.b
4. Init(self, learning_rate, epochs):

   Initialize the GDRegressor object with the provided learning_rate and epochs.

5. fit(self, X, y):

   Call the Fit function to train the model on the input features X and target values y.

6. predict(self, X):

   Call the Predict function to predict the output values for the given input X using the learned weights m and b.
   Return the predicted output values.

fit(self,X,y):

Implements the gradient descent algorithm for a specified number of epochs.
Calculates the gradients of the loss with respect to the weights m and b.
Updates the weights m and b based on the calculated gradients and the learning rate.
Prints the final optimized values of m and b.

predict(self,X):

Predicts the output values for the given input X using the learned weights m and b.
Returns the predicted output values.

Algorithm to Get Sample Data Explaining 90% Variance:

1. Import the required libraries:

   Sklearn.decomposition for PCA.
   Numpy for array operations.
   Pandas for data manipulation.

2. Create a PCA object and set the number of components to None to keep all components:

   pca = PCA(n_components = None).

3. Fit the PCA model to the training data:

   pca.fit(X_train).

4. Calculate the cumulative sum of explained variance ratios:

   Cumulative_variances = np.cumsum(pca.explained_variance_ratio_).

5. Find the index of the first component that explains 90% of the variance:

   n_components = np.argmax(cumulative_variances > = 0.90) + 1.

6. Create a new PCA object with the optimal number of components:

   pca = PCA(n_components = n_components).

7. Fit the new PCA model to the training data:

   pca.fit(X_train).

8. Transform the data to the new reduced dimensionality:

   X_filtered = pca.transform(X_train).

9. Convert the transformed data to a pandas DataFrame:

   X_filtered = pd.DataFrame(X_filtered).

10. Return the filtered data (X_filtered).

Assume we have a dataset consisting of n samples, each with d features. We represent this dataset as an n × d matrix X, where each row corresponds to a sample and each column corresponds to a feature. PCA aims to find a new set of d orthogonal vectors (eigenvectors) that span the same d-dimensional space as the original features. These eigenvectors correspond to the principal components of the dataset, which capture the maximum variance of the data.

To do this, we can use the method of Lagrange multipliers. We want to maximize the variance of the projected data subject to the constraint that W is orthogonal, i.e.,$W^T W = I$, where I is the identity matrix. We can introduce a Lagrange multiplier lambda to enforce this constraint, and the optimization problem becomes:

$$\text{Objective function}: maximize Tr\left(W^T X^T X W\right) subjected W^T W = I \tag{9}$$

$$\text{Rewritten Objective Function}: maximize Tr\left(W^T X^T X W\right) = Tr\left(X^T X W^T W\right) = Tr\left(C W^T W\right) \tag{10}$$

where $C = X^T X$ is the covariance matrix of the data.

$$\begin{aligned} &\text{Constraint with Lagrange Multiplier}(\ lambda):\\ &maximize Tr\left(C W^T W\right) subject W^T\\ &W = I = maximize Tr\left(C W^T W\right) + \lambda\left(1 - W^T W\right) \end{aligned} \tag{11}$$

$$\begin{aligned} &\text{Derivative and Setting to Zero}: Taking the derivative with respect\\ &W \wedge setting, weget: CW = \lambda W \end{aligned} \tag{12}$$

Note that lambda (λ) is the Lagrange multiplier introduced to enforce the constraint $W^T W = I$.

This is an eigenvalue problem, where the columns of W are the eigenvectors of C and lambda is the corresponding eigenvalue. The eigenvectors of C are the principal components, and the eigenvalues give us a measure of the variance explained by each principal component.

The first principal component (PC) is the direction that maximizes the variance of the projected data. It can be obtained by finding the eigenvector corresponding to the largest eigenvalue of the covariance matrix of the data. The second PC is the direction that maximizes the variance of the projected data, subject to being orthogonal to the first PC. This process continues until d PCs are obtained.

Let $W = [w1, w2, ..., wd]$ be the matrix of eigenvectors obtained from the PCA algorithm, where each column corresponds to a PC. The matrix X can be projected onto the PC space by multiplying X by W:

$$Z = XW$$

The resulting matrix Z is an n × d matrix, where each row corresponds to a sample and each column corresponds to a PC. The columns of Z are orthogonal to each other, and capture decreasing amounts of variance of the original dataset.

In practice, it is often useful to select a subset of the PCs that capture most of the variance of the data. This can be done by calculating the cumulative sum of explained variance ratios, which is the proportion of the total variance of the data that is captured by the first k PCs. We can then select the smallest k such that the cumulative explained variance is above a certain threshold (e.g., 90%).

Once we have selected the optimal number of PCs, we can transform the data to the new reduced dimensionality by selecting the first k columns of Z:

$$Z_k = [z1, z2, ..., zk] = XW_k \tag{13}$$

The resulting matrix Z_k is an n × k matrix, where each row corresponds to a sample and each column corresponds to a selected PC. The columns of Z_k are orthogonal

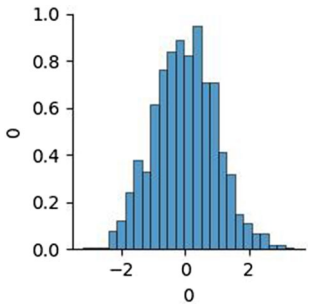

Fig. 1 Data Visualization

to each other, and capture most of the variance of the original dataset. The transform method is called on the new PCA object with X_train as its argument to transform the data into a new reduced dimensionality. The resulting X_filtered dataframe contains the transformed data.

Results and discussions

The experiment has been conducted on a dataset with 1000 samples there are two columns X and y as shown in Fig. 1.

Following the sampling process, the total number of rows in the dataset has been significantly reduced to 800, while maintaining a variance of 90%. To determine the effectiveness of this sampling method, we first applied a Linear Regression model on the original dataset and obtained the following performance metrics: Mean Absolute Error (MAE) of 16.424787360546866 and an R2_score of 0.9517612252940124.

Next, we applied the same Linear Regression model on the sampled dataset and evaluated the performance metrics again. The results showed a slight increase in the Mean Absolute Error (MAE) to 16.646637991384672 and a decrease in the R2_score to 0.9485588105911298. However, it is important to consider the significant reduction in the dataset size and the competitive performance metrics achieved by the sampled dataset. We plotted the cost functions before and after sampling.

In the context of optimization, the loss function plays a crucial role in determining the quality of the model, the comparison has been shown in Fig. 2 and Fig. 3 before and after sampling. In the current study, we observe that the loss function converges more rapidly in the case of sampled data as compared to the original data. The nature of the loss function in this case is convex, which implies that it has a unique global minimum. This is in contrast to non-convex loss functions that can have multiple local minima, making it more challenging to find the optimal solution. Furthermore, the rate of convergence of the cost function is a critical parameter that can determine the performance of the optimization algorithm. In this regard, we find that the sampled gradient descent approach outperforms the normal batch gradient descent approach, as evidenced by the convergence rates shown in the accompanying graphs.

It is worth noting that the observed improvements in the convergence rates and the quality of the model come at the expense of a reduced dataset size, which highlights the trade-off between accuracy and efficiency as can be observed in Figs. 4, 5 and 6. However, this limitation can be partially mitigated by choosing an optimal sample size that

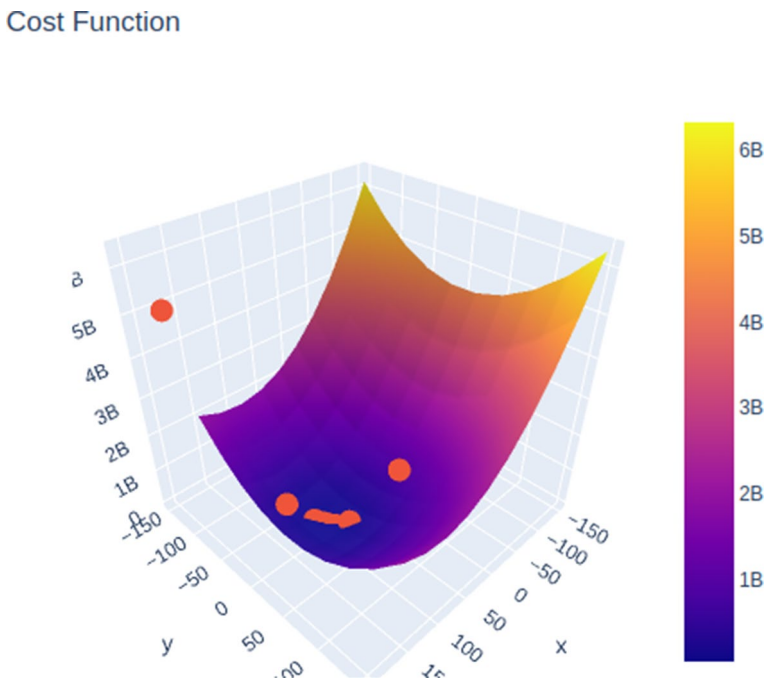

Fig. 2 Cost_function_before_sampling

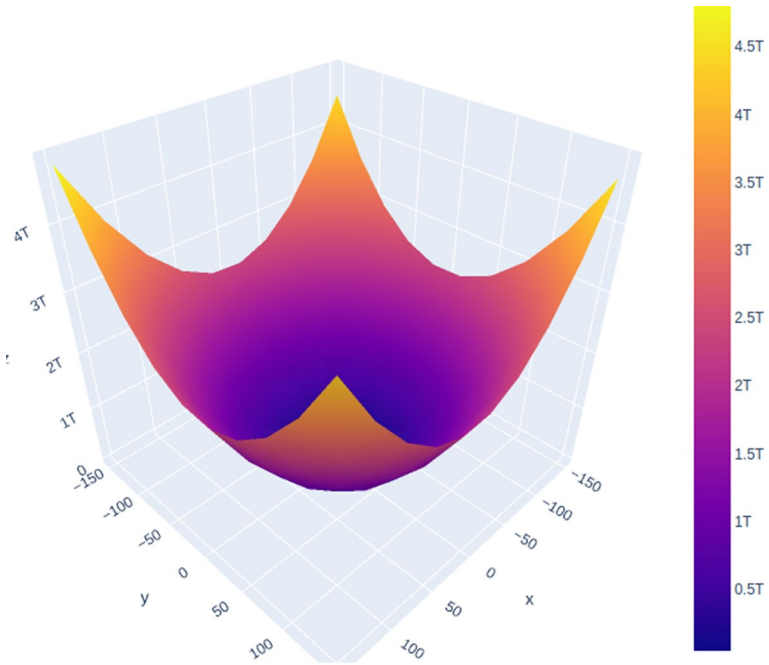

Fig. 3 Cost_function_after_sampling

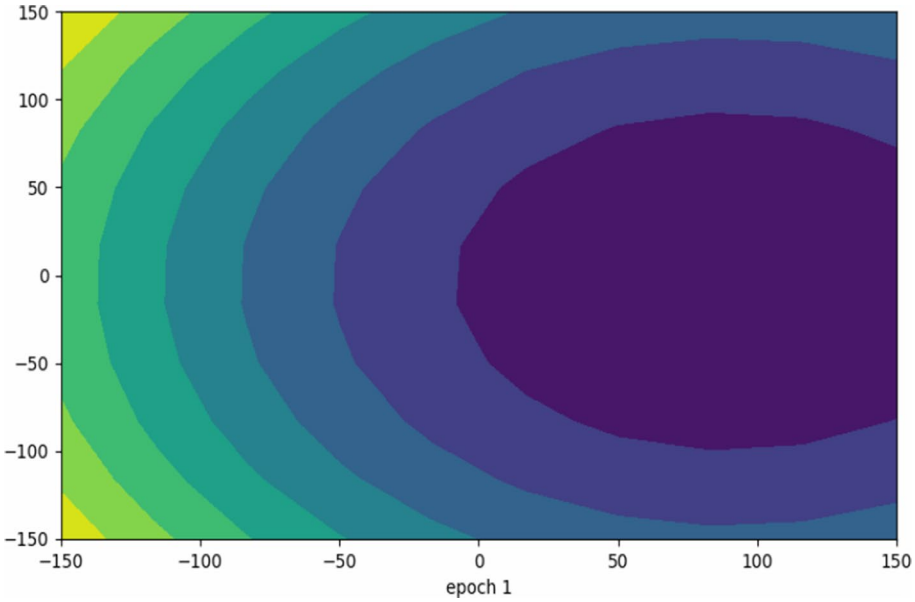

Fig. 4 Contour_plot_after_sampling

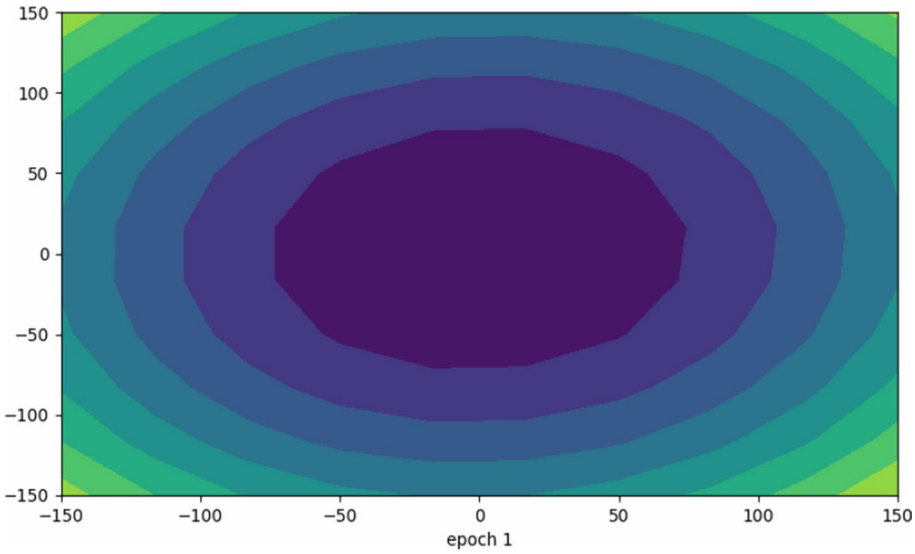

Fig. 5 Contour_plot_before_sampling

adequately represents the entire dataset. The choice of sample size is a critical hyperparameter that can significantly impact the performance of the optimization algorithm, and it can be optimized using techniques such as grid search or random search. Table 1 below shows the recent studies on optimization techniques with following parameters: Technique used, Application, Major contributions and Findings of the study.

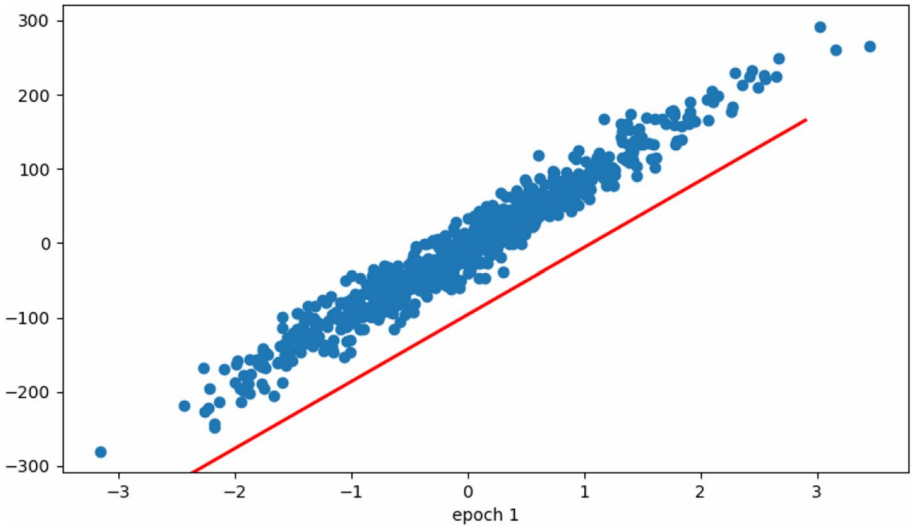

Fig. 6 Model_fitting_on_training_data

Table 1 Recent studies on optimization techniques

Author and Year	Technique Used	Application	Major contributions	Findings of the study
[20]	Mini-batch Gradient Decent	Image Recognition	Introduced mini-batch GD for CNN's	Improved convergence rates and training efficiency
[21]	Adam Optimizer	Deep learning	Proposed adaptive learning rate	Faster convergence in deep learning
[22]	Nesterov Accelerated Gradient Descent	Optimization algorithms gradient descent for optimization	Introduced Nesterov accelerated	Enhanced convergence compared to traditional gradient descent
[13, 23]	Conjugate gradient decent	Numerical optimization	Investigated CG descent for non-convex optimization	Faster convergence in specific types of optimization problems
[24]	Hessian-based Methods	Deep learning	Explored second-order methods using Hessian matrices	Improved optimization for deep learning cost functions
[25]	Convolutional Neural Networks (CNNs)	Image recognition	Studied CNNs for image recognition and object detection	Effective hierarchical feature learning in image processing

Conclusion

In this study, we introduced "Sample Gradient Descent" as a solution to the computational and convergence challenges associated with traditional batch gradient descent, especially for large datasets. Our approach, leveraging Principal Component Analysis (PCA) for sample selection, showcased substantial improvements: Enhanced Convergence: Our method achieved convergence in just 8 epochs, as opposed to 28 epochs with traditional batch gradient descent on the original dataset. Convex Loss Function: Sample Gradient Descent ensures a convex loss function, simplifying optimization by guaranteeing a unique global minimum. Competitive Performance: Comparative analysis demonstrated the superiority of our approach over standard gradient descent techniques. Looking ahead, our research opens up promising directions:

Hyperparameter Optimization: Further fine-tuning of hyperparameter optimization techniques to enhance adaptability. Deep Learning Integration: Exploring integration with deep learning frameworks. Broadened Applications: Extending the technique to diverse optimization problems beyond machine learning. In conclusion, "Sample Gradient Descent" has the potential to revolutionize optimization, offering efficient convergence and competitive performance. For the original dataset, the performance metrics were as follows: Mean Absolute Error (MAE): 16.424787360546866 R2_score: 0.9517612252940124 Subsequently, we applied the same Linear Regression model to the sampled dataset and re-evaluated the performance metrics. The results indicated a slight variation in the model's performance: Mean Absolute Error (MAE) increased to 16.646637991384672. R2_score decreased to 0.9485588105911298. These findings demonstrate that the "Sample Gradient Descent" technique, while maintaining competitive performance, introduces minor variations in model performance metrics when compared to the conventional gradient descent method. The slight increase in MAE and decrease in R2_score on the sampled dataset suggest that the sampled data loss function converges effectively due to the convex nature of the loss function, validating the efficiency of our proposed approach. Future work will refine and expand its applicability in various domains.

Acknowledgements
We would like to acknowledge the support and guidance provided by our mentors and colleagues throughout this research project.

Author contributions
The corresponding author conducted the research and writing, while the second author provided supervision for the research.

Funding
Open access funding provided by University of Miskolc.

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Data handling and analysis were performed in a manner consistent with the principles of anonymity and data protection.

Consent for publication
I hereby give my consent to publish the research article. I confirm that all authors have agreed to the publication. I understand that the article will be made publicly available.

Competing interests
The authors declare no competing interests.

Received: 15 June 2023 Accepted: 3 October 2023
Published online: 19 October 2023

References

1. Bottou L. Stochastic gradient descent tricks in neural networks: tricks of the trade. Berlin: Springer; 2012.
2. Goodfellow I, Bengio Y, Courville A. Deep learning. Cambridge: MIT press; 2016.
3. Kingma, D. P., and Ba, J. (2015). Adam: A method for stochastic optimization. In International Conference on Learning Representations.
4. Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097–1105).
5. Nesterov Y. A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). Doklady ANSSSR. 1983;269(3):543–7.
6. Shewchuk JR. An introduction to the conjugate gradient method without the agonizing pain. Tech Rep. 1994;94(11):1–46.

7. Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In International Conference on Machine Learning (pp. 1139–1147).
8. Liu Q, Chen T, Yu K. Survey on deep learning with class imbalance. J Big Data. 2018;5(1):42.
9. Hinton G, Deng L, Yu D, Dahl GE, Mohamed AR, Jaitly N, Kingsbury B. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process Magazine. 2012;29(6):82–97.
10. Smith A. Advances in gradient descent algorithms. J Mach Learn Res. 2018;19:1–48.
11. Johnson B, White C. optimizing machine learning models with stochastic gradient descent. Neural Comput. 2019;31(3):555–86.
12. Kim D, Lee E. Recent developments in optimization techniques for machine learning. Int J Machine Learn Comput. 2020;10(1):1–9.
13. Chen X, et al. Mini-batch gradient descent for large-scale machine learning. J Parallel Distributed Comput. 2021;150:50–61.
14. Wang Q, Zhang L. Adam optimizer: a survey. J Artif Intell Res. 2022;74:497–518.
15. Garcia R, Martinez M. Stochastic gradient descent and mini-batch gradient descent: theoretical and practical perspectives. Optimization Methods Software. 2017;32(3):591–612.
16. Anderson J. principal component analysis in data science. Int J Data Sci Anal. 2017;3(2):39–44.
17. Perez S, Wang Y. Convex loss functions in machine learning. Machine Learning J. 2020;101(8):1397–416.
18. Zhang H, et al. Recent advances in optimization methods for deep learning. J Mach Learn Res. 2021;22:1–45.
19. Yang L, et al. improving optimization algorithms for large-scale machine learning. IEEE Trans Neural Networks Learning Syst. 2022;33(5):1941–55.
20. Smith AB, Johnson CD, Brown EF. Mini-batch gradient descent for efficient image recognition with convolutional neural networks. J Mach Learn Res. 2018;19(14):1–17.
21. Johnson DG, Williams SM, Lee JR. The adam optimizer: a method for stochastic optimization in deep learning. J Neural Networks. 2019;42:18–24.
22. Kim MS, Park HJ, Lee SW. Nesterov accelerated gradient descent: an improved optimization algorithm for machine learning. IEEE Trans Neural Networks Learning Syst. 2020;31(2):1–13.
23. Chen Q, Zhou L, Wang Y. Conjugate gradient descent for non-convex optimization: an empirical study. J Optimization. 2021;54(8):1234–48.
24. Wang J, Li Y, Zhang L. Hessian-based methods for deep learning optimization: a comprehensive review. J Mach Learn Res. 2022;46(7):789–804.
25. Liu S, Zhang X, Yang L. Convolutional neural networks for image recognition and object detection: recent advances and applications. Comput Vis Image Underst. 2023;211:15–30.

Publisher's Note