Proceedings of CONF-MLA 2026 Symposium: Intelligent Systems and Automation: AI, IoT, Robotic Engineering & Algorithm
DOI: 10.54254/2755-2721/2026.LD35719

A Comparative Study of Gradient Descent and Stochastic Gradient Descent in Neural Network Function Approximation

Pu Gong

Faculty of Science, University of British Columbia, Vancouver, Canada
pugong0415@gmail.com

Abstract. Neural networks are often employed in artificial intelligence because they are capable of learning patterns and approximating nonlinear functions from input data. But the performance of a neural network is not only decided by the model structure, but also by the optimization technique utilized in the training process. In this research, we compare full-batch Gradient Descent versus mini-batch Stochastic Gradient Descent in a neural network function approximation problem. A tiny feedforward neural network is trained to approximate the nonlinear function. The experiment is implemented in Python and the comparison is made based on training loss curves, final training means squared error, final test mean squared error and prediction visualization. The results reveal that the full-batch Gradient Descent gives a smoother loss curve since in each update the entire training dataset is used. Mini-batch Stochastic Gradient Descent on the other hand shows more noticeable oscillations but reaches a lower ultimate training MSE and test MSE in this experimental context. This implies that stochastic updates can be less stable at each step but can still assist the model in achieving greater approximation performance in the same number of epochs. The study shows how optimization techniques influence neural network training behavior and it combines mathematical optimization theory with real model performance.

Keywords: Gradient descent, Stochastic gradient descent, Neural network, Function approximation, Mean squared error

1. Introduction

Neural networks are widely used across various domains, particularly in artificial intelligence. They are able to find patterns and approximate functions from data. In supervised learning, this can be seen as adjusting the parameters of the model to bring the predicted output as close as possible to the target output. In the early days of backpropagation, learning in neural networks was characterized by the continuous adjustment of the connection weights to minimize the difference between actual and target outputs [1]. Therefore, training a neural network is not just a computational task, but also an optimization problem.

Mathematical optimization is the core principle of machine learning. According to the study by Bottou, Curtis, and Nocedal, optimization is a core part of machine learning, where the parameters

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2026.LD35719

of a system are tuned to make predictions on unseen data [2]. Among various optimization methods, the predominant technique for training neural networks is Gradient Descent, commonly referred to as GD [3]. However, there are different kinds of gradient-based optimization that can have different training behaviors. Full-batch Gradient Descent calculates the update based on the full training dataset, and Stochastic Gradient Descent calculates the update based on estimates from individual random samples. This leads to a trade-off between update stability and computational efficiency.

This paper aims to compare the efficiency and performance of Gradient Descent and Stochastic Gradient Descent in the context of function approximation using a neural network. The work contains a brief mathematical explanation with a simulated experiment implemented in Python. A small feedforward neural network is trained to approximate the nonlinear trigonometric function. The comparison is based on training loss, prediction accuracy, convergence behavior, and final mean squared error. This study is important as it seeks to demonstrate how the selection of optimization algorithms influences the practical efficacy of neural network learning by integrating the mathematical concept of gradient-based optimization with empirical training outcomes.

2. Mathematical background

2.1. Neural network approximation

A neural network can be expressed as a parameterized function. In this paper, the model can be written as $f_\theta(x)$, where θ represents the model's weights and biases. In a supervised function approximation task, the objective of training is to select parameters that align the model's predictions with the target output. This view connects neural networks with approximation theory, because the network not only makes predictions but also learns functions from the given data. LeCun et al. also describe multilayer neural networks trained with back-propagation as a successful example of gradient-based learning [4].

Theoretically, universal approximation results justify the use of neural networks for function approximation. Cybenko's result indicates that neural networks with sufficient sigmoidal units can approximate continuous functions on compact domains [5]. When the number of hidden units is large enough, Hornik, Stinchcombe and White further show the strong approximation ability of multilayer feedforward networks [6]. The results support the feed-forward neural network in this experiment. However, approximation ability only means an appropriate network can represent the target function under certain conditions. This does not guarantee that the training process will identify the optimal parameters. Thus, some optimization strategies are still needed to transform the theoretical approximation ability of a network into actual learning performance.

2.2. MSE loss function

To train a neural network, a loss function is needed to measure the difference between the model prediction and target output. This paper uses the Mean Squared Error, which is commonly used in regression and function approximation. The MSE loss is defined as follows:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(f_\theta(x_i) - y_i\right)^2 \tag{1}$$

In this equation, n is the number of training samples, $f_\theta(x_i)$ is the predicted value by the neural network and y_i is the target value. Lower MSE indicates that the model prediction is closer to the desired output on average. Back-propagation allows the computation of the influence of each weight

DOI: 10.54254/2755-2721/2026.LD35719

on the loss, which in turn enables the parameters to be updated through gradient-based optimization [1]. Thus, the loss function acts as a bridge between the prediction error and the optimization method.

2.3. Gradient optimization algorithms

Gradient Descent is a straightforward method for optimizing a differentiable loss function. It updates the model parameters in the direction that reduces the loss. In each iteration, parameters are updated along the negative gradient direction:

$$\theta_{k+1} = \theta_k - \alpha \nabla L(\theta_k) \tag{2}$$

Here, α is the learning rate, and $\nabla L(\theta_k)$ is the gradient of the loss function at the current value. The gradient points to the direction where loss rises fastest; therefore, updating parameters along the negative gradient can reduce the loss value. Gradient Descent is one of the most popular strategies for optimizing neural networks [3]. In full-batch of gradient descent, the gradient is computed for the entire training set before each update. This guarantees a stable update direction but can be computationally prohibitive for large data sets.

Stochastic Gradient Descent modifies this process by computing the gradient using a single training example or a mini batch. The update can be written as

$$\theta_{k+1} = \theta_k - \alpha \nabla L_B(\theta_k) \tag{3}$$

Here B is a randomly sampled mini batch. The theory of stochastic approximation justifies this idea theoretically. R. Robbins and D. Monro considered the problem of estimating an unknown target from a sequence of noisy observations [7]. In machine learning, this principle is realized as stochastic gradient descent.. We approximate the full gradient by a random sample of the data. Stochastic gradient descent is usually more noisy than full-batch gradient descent, but it is often more efficient and scalable.

3. Methodology

3.1. Experimental setup

This paper conducts a small-scale Python experiment to compare full-batch Gradient Descent with mini-batch Stochastic Gradient Descent in a neural network function approximation task. The objective of the experiment is not to develop a substantial artificial intelligence system, but to establish a controlled environment where the training behaviors of the two optimization methods can be directly monitored. The target function used in the experiment is defined as follows:

$$f(x) = \sin(x) + 0.3\cos(3x) \tag{4}$$

The input values are selected from the interval $[-2\pi, 2\pi]$. The function involves trigonometric functions. We calculate the corresponding output values using the target function and add a small amount of Gaussian noise to the training outputs. Noise is added to make the data set similar to real supervised learning problems in which the observations often contain noise. The test data are generated from the same target function without adding any additional noise, in order to test the accuracy of the final prediction on the true function. Figure 1 visualizes the generated target function

DOI: 10.54254/2755-2721/2026.LD35719

and the noisy training observations. The smooth curve represents the true function, while the scattered points represent the noisy samples used for training.

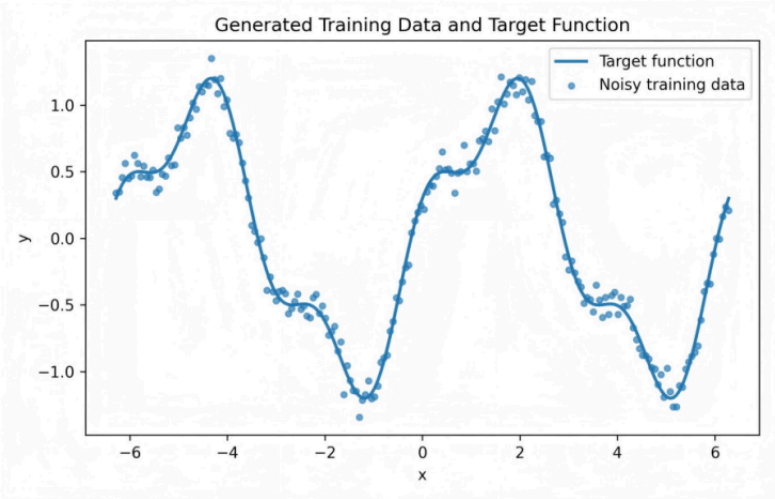

Figure 1. Generated training data and target function

3.2. Model and parameters

The experimental model is a small feedforward neural network with one input layer, two hidden layers, and one output layer. Each hidden layer contains 32 neurons, and the hyperbolic tangent function is used as an activation function because it can represent nonlinear patterns and is suitable for this simple one-dimensional function approximation task. The input values will be normalized before training to improve stability.

Both GD and SGD use the same model architecture and initial parameters. This design guarantees that the comparison is based on the optimization approach and not on variations in the structure of the neural network. The learning rate of full-batch GD is set to 0.03 and the learning rate of mini-batch SGD is set to 0.01. The learning rate for SGD is smaller because the mini-batch updates are noisier, and a smaller step size helps keep the training stable. We set the mini-batch size to 20, so that SGD can update the model based on a tiny sample of training data each time. The following Table 1 summarizes the main experimental settings used in Python simulation.

Table 1. Experimental setting

Setting	Value
Target Function	f(x)=sin(x)+0.3cos(3x)
Input Range	[-2π, 2π]
Training Size	200
Test Size	200
Noise Standard Deviation	0.08
Model	Feedforward Neural Network
Hidden Layers	2
Neurons per layer	32
Loss Function	Mean Squared Error
GD learning rate	0.03

DOI: 10.54254/2755-2721/2026.LD35719

Table 1. (continued）

SGD learning rate	0.01
SGD batch size	20
Epochs	2500

3.3. Metrics and procedures

Both models are trained for 2500 epochs with the same training and testing data. Gradient Descent utilizes the full batch,wherethe whole training set is used to compute the updates to the parameters. For mini-batch Stochastic Gradient Descent, the training data are randomly mixed at each epoch and the parameter updates are performed with mini-batches of 20 samples. This means that GD conducts more consistent full-dataset updates, whereas SGD does more frequent but noisier mini-batch updates.

The training loss at each epoch is recorded and the final model predictions on the test data are evaluated. The evaluation is based on three basic measures. First, the training MSE at the end of training reflects the fit of the model to the noisy training data. Second, the final test MSE is a measure of how closely the trained model approximates the true function at unseen test points, and this is important because training performance alone does not fully describe whether a model can generalize beyond the training data. Hardt, Recht, and Singer connect stochastic gradient methods with generalization behavior and argue that stochastic gradient methods can have small generalization error under suitable training conditions [8]. Third, the standard deviation of the last 200 training losses is used as a simple measure of training stability. A smaller value means the loss curve is more stable at the conclusion of training while a bigger value means there is more variability.

These parameters are utilized to compare the convergence behavior, training stability and prediction accuracy of full-batch GD with mini-batch SGD. In the next section the results are studied using the loss curves of the training, the final MSE values, and the visualization of the predictions.

4. Result and analysis

4.1. Loss curve comparison

The experimental results show that there is a significant difference between full-batch and mini-batch Stochastic Gradient Descent. The same feedforward neural network is trained on the same function approximation task using both methods, but the loss curves and the final predictions behave differently. The difference mainly comes from how the gradient is computed. The full-batch GD calculates each update by using the entire training dataset, while mini-batch SGD estimates each update from smaller random subsets of the data.

Figure 2 illustrates the training loss trajectories of Gradient Descent and mini-batch Stochastic Gradient Descent throughout 2500 epochs. The blue curve is for full-batch GD while the orange curve is for mini-batch SGD. The GD curve goes down more softly, which means a more stable update direction. This is because each GD update relies on the whole training set. In contrast, the SGD curve exhibits a downward trend with more obvious swings. Each mini batch is simply an approximation of the complete gradient; therefore the direction of each update has greater randomness. Ruder states that SGD updates often exhibit significant variation and so might generate

DOI: 10.54254/2755-2721/2026.LD35719

oscillations in the objective function [3]. In this experiment however, the volatility does not prevent SGD from achieving a lower training loss.



Figure 2. Training loss curves of GD and SGD

4.2. MSE quantitative analysis

The numeric results also confirm this observation. As shown in Table 2, full-batch GD achieves a final training MSE of 0.1008 and a final test MSE of 0.1023. Mini-batch SGD ends up with a lower final training MSE, 0.05298, and a lower final test MSE, 0.04827. The values show that SGD provides a better approximation of the target function for the chosen training setting.

The difference can be explained by the rate of updates. In this experiment, the full-batch GD updates the model using all the 200 training samples in each epoch. Mini-batch SGD employs a batch size of 20, which means that it updates the parameters several times in one epoch. It is noisier at each update, but the model gets more frequent opportunities to change its parameters. This could explain the lower MSE after the same number of epochs for SGD. The measure of stability also reveals the predicted trade-off. The standard deviation of the previous 200 training losses is 0.0010 for full-batch GD and 0.0022 for mini-batch SGD. In other words, towards the end of training GD is more stable whereas SGD exhibits higher swings. The lower final training and test MSE of SGD, however, indicates that more volatility does not inevitably lead to inferior final performance.

Table 2. Final performance comparison

Optimizer	Final Training MSE	Final Test MSE	Std. of Last 200 Training Losses	Batch Size	Learning Rate
Full-batch GD	0.1008	0.1023	0.001040	200	0.03
Mini-batch SGD	0.05298	0.04827	0.002175	20	0.01

4.3. Prediction visualization

The final predictions of the GD-trained and SGD-trained models are compared with the target function in Figure 3. Both approaches capture the general non-linearity of the target function. However, in certain locations, especially in the vicinity of some peaks and troughs, the SGD forecast is more in line with the target function. The GD prediction is smoother and misses more of the high variance regions of the function. This means that the full-batch GD underfits the target function to a

DOI: 10.54254/2755-2721/2026.LD35719

larger extent under the given learning rate and epoch setting. This is aligned with the MSE values presented in Table 2. Since the SGD model has a lower final test MSE, its prediction curve is expected to be closer to the true function on the test points. The prediction visualization therefore provides a visual explanation of the numerical results. It shows not only that SGD has a lower error value, but also where this improvement occurs on the function curve.

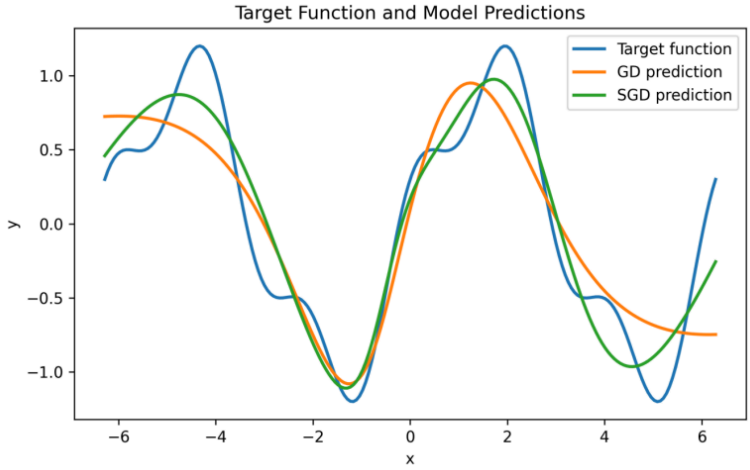

Figure 3. Target function and model prediction

4.4. Discussion

Results clearly demonstrate a compromise between stability and approximation accuracy. Because each update uses the whole data set, full-batch GD has a flatter loss curve. This makes the direction of its gradient more consistent. However, in this experiment GD also converges more slowly and provides a smoother prediction curve which underfits several nonlinear aspects of the objective function.

Mini-batch SGD is different. Its loss curve is not as smooth as that of full-batch GD, since each update is based on a randomly picked mini-batch. However, a smaller batch size means more updates of parameters in each epoch. Bottou's analysis helps explain why stochastic gradient approaches are popular in practice: instead of waiting for a full-dataset gradient, SGD can continue updating the model using sampled training cases, thus decreasing the cost of each step [9]. Bottou, Curtis, and Nocedal also refer to stochastic gradient methods as the workhorses for large scale machine learning, especially when the classic batch optimization approaches are not practicable [2].

This conclusion should not be understood as a proof that SGD is always superior to GD under all circumstances. It relies on learning rate, batch size, number of epochs, model architecture, and initialization conditions. In this experiment, GD employs a learning rate of 0.03, whereas SGD utilizes a lesser learning rate of 0.01. The smaller learning rate for SGD is chosen to reduce instability from noisy mini-batch updates. The final output can vary depending on the learning rate and batch size selected. Training and initialization methods have a great impact on the training of neural networks according to Glorot and Bengio [10]. Therefore, this experiment must be seen as a controlled comparison in a certain arrangement. This demonstrates that in this context, SGD can produce superior final approximation performance while GD provides a more stable and smoother training procedure.

DOI: 10.54254/2755-2721/2026.LD35719

5. Conclusion

This research compared full-batch Gradient Descent with mini-batch Stochastic Gradient Descent in a neural network function approximation challenge . The major aim was to study how two gradient-based optimization techniques affect the training behavior while using the same model architecture, dataset and number of epochs. In the python experiment, full-batch Gradient Descent resulted in a smoother loss curve since every update was calculated from the whole training set. However, mini-batch Stochastic Gradient Descent presented more obvious oscillations, because each update was performed on a randomly selected mini-batch. However, in this context, SGD achieved a lower final training MSE and test MSE than GD. The SGD prediction curve was closer to the target function than the GD prediction.

The major conclusion is that stability and final accuracy are not always related. A smoother training curve does not imply a better final approximation. In this experiment, the noisier mini-batch updates of SGD led to more effective progress of the model across the same number of epochs. This conclusion is useful for understanding neural network training, since it reveals that the choice of optimization technique can have a direct impact on both convergence behavior and prediction quality.

There are limitations to this study. The experiment was based on a basic one-dimensional synthetic function. Hence the conclusion cannot be generalized to a complex task such as picture classification or natural language processing. The neural network was also tiny and only one setting of learning rate, batch size and model architecture was explored. Future work could include comparison more learning rates and batch sizes, testing deeper neural networks, and performing the same comparison on real datasets. This comparison may be extended further to more advanced optimization approaches after a more thorough investigation of the mathematical mechanisms.

References

[1] Rumelhart, D. E., et al. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533–536. https://doi.org/10.1038/323533a0.

[2] Bottou, L., et al. (2018). Optimization methods for large-scale machine learning. *SIAM Review, 60*(2), 223–311. JSTOR, http://www.jstor.org/stable/45109416.

[3] Ruder, S. (2016, September). An overview of gradient descent optimization algorithms. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1609.04747.

[4] Lecun, Y., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278–2324. https://doi.org/10.1109/5.726791.

[5] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control Signals and Systems, 2*(4), 303–314. https://doi.org/10.1007/bf02551274.

[6] Hornik, K., et al. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2*(5), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8.

[7] Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics, 22*(3), 400–407. JSTOR, http://www.jstor.org/stable/2236626.

[8] Hardt, M., et al. (2015). Train faster, generalize better: Stability of stochastic gradient descent. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1509.01240.

[9] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. *Proceedings of COMPSTAT'2010*, 177–186. https://doi.org/10.1007/978-3-7908-2604-3_16.

[10] Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *PMLR*. https://proceedings.mlr.press/v9/glorot10a.