Proceedings of the 2nd International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/104/20241140

Analysis of the Hyperparameter Selection in Machine Learning

Defu Qian

School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China

andy-q@sjtu.edu.cn

Abstract. A Machine learning is now playing a role in various fields. With the development of the internet, the volume of data is increasing, and the algorithms of machine learning are becoming more complex. Adjusting hyperparameters is a challenge for beginners who are new to machine learning. This research implemented a simple neural network model through code. Moreover, this study has changed some of the most basic parameters of the model and trained different models. This study has used visualization methods to show how the model training time and performance change with the alteration of hyperparameters, and derived their respective accuracy rates. This article provides entry-level data and code for newcomers to the field of machine learning. Since this experiment uses a federated learning model, employs local_update SGD, and also utilizes parallel methods, it is quite comprehensive, covering many algorithm codes for machine learning beginners. Testing the basic parameters of the model can help newcomers to machine learning understand more clearly the impact of hyperparameters on the model. It also aids in the development of more algorithms for adjusting hyperparameters.

Keywords: Machine learning, hyperparameters, visualization analysis.

1. Introduction

Machine learning (ML) algorithms are already playing a big role in various fields, including user behavior analytic, image recognition, natural language processing (NLP), etc. [1]. They are used to handle the data more efficiently while the data can sometimes be difficult for a human to analyze [2]. People can categorize ML algorithms based on feedback, purpose, and method [3]. Distributed machine learning inherently involves learning across different machines. Since the 21st century, the Internet has developed rapidly. Video-sharing websites like YouTube receive hundreds of hours of video uploads every minute, and social networking sites with over a billion users, such as Facebook, handle petabytes of social media data. The surge in big data has led to a growing need for high-dimensional, sophisticated ML models with parameters ranging from billions to trillions to manage the increasing data complexity and to enhance predictive accuracy for smarter tasks [4]. Training large-scale ML models on vast datasets exceeds the storage and computational capacity of a single machine, and this limitation has driven the development of distributed machine learning, where machine learning tasks are carried out across various platforms such as research clusters, data centers, cloud services and tens to thousands of machines [5]. Nevertheless, conventional centralized machine learning methods often consolidate various raw data sets, originating from diverse devices or entities, into a unified server or data center.

DOI: 10.54254/2755-2721/104/20241140

This practice, however, raises significant issues regarding data confidentiality and security. [6]. Federated Learning (FL) is a sophisticated collaborative framework that facilitates the training of machine learning models across various distributed resources. In FL, numerous participants contribute to the development of a shared model without the need to centralize their sensitive data on a single server or within a data center [7]. This approach ensures compliance with stringent legal data regulations, enabling a collective effort to enhance model capabilities while preserving privacy and data integrity.

Within the realm of machine learning, parameters are categorized into two primary groups: one category consists of parameters that can be automatically adjusted based on data during the training process, such as the weights connecting nodes in a neural network, which are referred to as model parameters; the other category includes parameters that must be preset before training, determining the model's configuration and behavior, and these are referred to as hyperparameters. The setting of hyperparameters has a crucial impact on the model's performance, but they do not participate in the automatic adjustment process during model training [8]. Different models have different hyperparameter Settings. While certain machine learning models like non-negative matrix factorization possess a limited set of hyperparameters, recent deep learning techniques typically feature a much larger number of adjustable parameters [9]. For a beginner who want to train a simple machine learning model like local-update SGD model, it can be puzzling. However, the setting of the hyperparameters can sometimes be even more important than choosing a model [10].

In order to show beginners how different hyperparameters can affect the training and effect of the model, this study has written this article to give beginners a deeper understanding of hyperparameters. Although there are already some methods to automatically tune hyperparameters, e.g., Hyperparameters and tuning strategies for random forest [11], or Meta-learning for symbolic hyperparameter defaults [12], they are still based on a basic understanding of hyperparameters. This study is going to implement a neural network training process based on Federated Learning. Specifically, the script is implemented by the PyTorch framework to train and test a model for recognizing MNIST handwritten digit datasets. For each client, local-update SGD is implied to do model update. In addition, this research uses the matplotlib library to draw a graph of training losses over rounds to visualize the training process. By the graphs, one can easily find out how the hyperparameters affect the model. Finally, this study will give the suggestions about how to tune the hyperparameters.

2. Data and method

A straightforward neural network training script utilizing the PyTorch library creates a three-layer feedforward neural network named Net. This network comprises three dense layers (fc1, fc2, fc3) and utilizes a ReLU activation function. It processes a 28x28 pixel handwritten digit image by converting it into a 784-dimensional vector, subsequently using the layers to feature-extract and classify the data, culminating in the prediction of one of 10 classes, representing the digits from 0 to 9. The code implements distributed training primarily through the use of Python's multiprocessing module, specifically the Pool class, which allows to create a pool of processes and execute multiple tasks in parallel within these processes. Here are the key steps for implementing distributed training in the code:

- Create Model Copies. Employ the duplicate_model function to create multiple replicas of the original model. Each replica will undergo separate training processes, utilizing distinct subsets of the available data.
- Create a List of Optimizers. For each model copy, create a corresponding optimizer instance for subsequent gradient updates.
- Create a DataLoader. Use a RandomSampler to create a random sampler for the training dataset, ensuring that each model copy sees a different subset of data during training.
- Parallel Training. Use the Pool class to create a process pool and use the pool.map method to apply the train_model function to each model copy and its corresponding optimizer, loss function, random sampler, and training dataset. The pool.map will automatically distribute tasks to processes in the pool and execute training in parallel.

DOI: 10.54254/2755-2721/104/20241140

- Collect Training Results. The pool.map returns a list containing models and losses. Extract all models from this list and aggregate their loss values.
- Average Model Weights. Use the average_models function to average the weights of all trained model copies to form an ensemble model.
- Update the Main Model. Load the averaged weights into the main model, completing one round of distributed training.
- Iterative Process. Repeat the above process in each round of communication, continuously updating the main model.

The evaluation of the model in this code is carried out through the following steps:
Training Loss Assessment: During each round of federated learning communication, each model is trained independently on the local dataset, and then the total loss is calculated. This total loss is the sum of the losses of all models during the local training steps, which is used to monitor the performance of the model during the training process.

- Visualization of Loss. The loss values are stored in the losses_array array, which records the average loss after each communication as the training rounds progress.The matplotlib library is used to plot the loss values as a function of the training rounds, helping to visually assess the progress and convergence of the model training.
- Testing Accuracy Assessment. Once all communication cycles have concluded, the ensemble model undergoes assessment utilizing the test dataset. This dataset comprises data that was not included in the training phase, serving as a means to gauge the model's capacity to generalize to new, unseen information. Throughout the testing phase, the model is configured to evaluation mode (model.eval()), which deactivates the learning mechanisms of Dropout and Batch Normalization layers. The torch.no_grad() context manager is implemented to cease gradient calculations, enhancing the evaluation process by speeding it up and diminishing memory usage. For every image within the test dataset, the model generates a prediction, which is subsequently matched against the actual label. A tally of accurate predictions is maintained. Ultimately, the mean accuracy for the test dataset is determined, representing the ratio of correct predictions to the overall number of predictions made.
- Printing Test Accuracy. At the end of the testing process, the code prints out the test accuracy, providing a quantitative indicator of the model's performance.
- Saving the Visualization Chart. The chart of the loss curve is saved as an image file using plt.savefig('model.png'), which is convenient for later viewing or as part of a report.

For the data, the MNIST dataset, a cornerstone in machine learning and computer vision, was developed by NIST and USC. It comprises numerous images of handwritten numbers, serving as a benchmark for training diverse image recognition algorithms. The dataset structure is as follows:

- Image Size. Each image is 28x28 pixels, meaning each image consists of 784 pixels.
- Image Type. The images are grayscale, with pixel values ranging from 0 to 255.
- Categories. The dataset contains 10 categories, corresponding to the digits 0 through 9.

For dataset division.

- Training Set. The training set contains 60,000 images for model training.
- Test Set. The test set contains 10,000 images for evaluating model performance.

In this code, the data preprocessing steps include:

- Conversion to Tensors: Transforms. ToTensor() is used to convert images from PIL format or NumPy arrays to PyTorch tensors. This conversion scales pixel values from [0, 255] to [0.0, 1.0].

DOI: 10.54254/2755-2721/104/20241140

- Data Augmentation. Although not explicitly used in this code, data augmentation techniques such as rotation, scaling, and cropping can be applied in practice to increase the diversity of the dataset and improve the model's generalization capabilities.

RandomSampler is used to randomly select data during the training process, ensuring that the samples each model sees during local training are random. This will help prevent the model from becoming too dependent on data in a specific order, thereby improving the model's generalization capabilities. DataLoader is a class in PyTorch used for loading datasets. It provides a convenient way to iterate over datasets, supporting features like automatic shuffling of data order and multi-threaded loading. During training, data is typically processed in small batches. In this code, each batch contains 128 images. Due to its simplicity and wide application, the MNIST dataset has become a standard dataset in machine learning and deep learning tutorials. It is not only used for teaching but also for benchmarking to evaluate the performance of new algorithms.

3. Results and discussion

When trying to tune some basic hyperparameters, things as follows should be considered. Based on the code mentioned above, this study gets the accuracy under different hyperparameters settings.

3.1. Training Results

During model training, one might encounter various time-consuming issues that can affect the efficiency and speed of model training. It can be known from the definition of a hyperparameter to see whether it have effect on the time consuming. Hperparameters like learning_rate, batch_size and num_communications can greatly affect the efficiency of training. With different learning rate setting, one ontain the results shown in Fig. 1. One should pay attention to the numbers on the y-axis. The model of figure1 get accuracy of 88.74% while the other get accuracy of 67% both after about 30 epochs. Actually, it can be seen from the figure that the loss in figure1 is lower than 17 after 3 epochs while the other make it after 30 epochs. It means if one adjusts the learning rate, one can reach high accuracy in less time. The problem is if the learning rate is high, it will be difficult for the model to converge. Thus, one can see the loss bounces ups and downs. If the learning rate is too high, the loss can even diverge. Num_communication, which is the number of communication rounds that occur during the training process can affect the training time, too. The following two figures have different num_communication and the training time are 2.3 min and 6.8 min on my computer. The accuracies of them are 88.74% and 91.64% as depicted in Fig. 2.

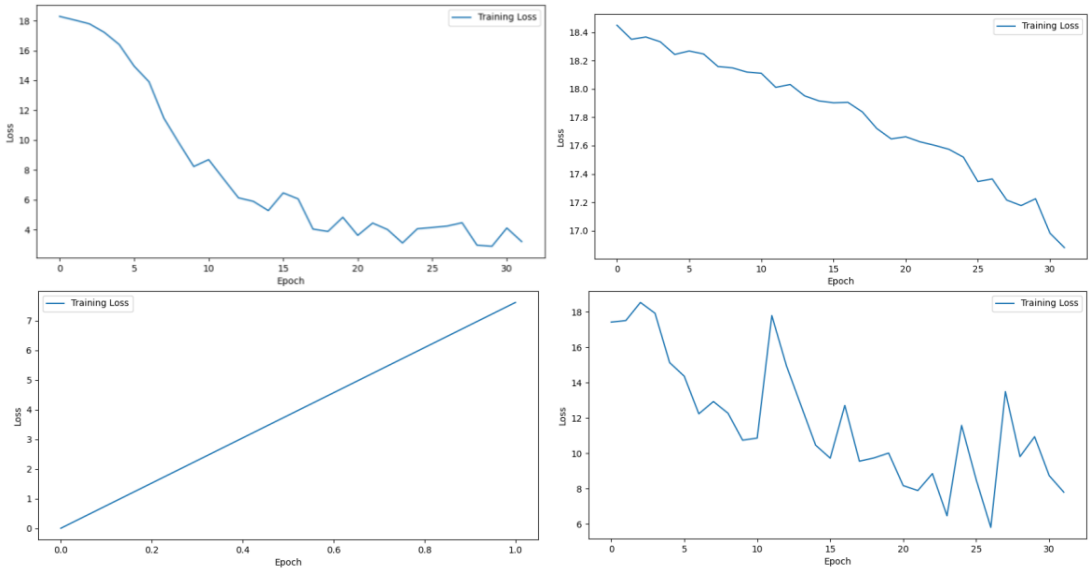

Figure 1. Loss functions for learning rate of 0.1 (left upper), 0.01 (right upper), 1 (left lower) and 5 (right lower) (Photo/Picture credit: Original).

DOI: 10.54254/2755-2721/104/20241140

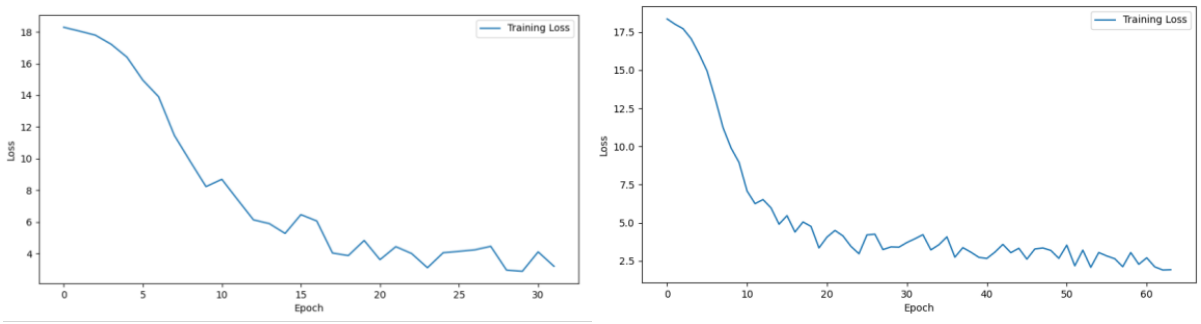

Figure 2. Results for num_communication=32 (left panel) and num_communication=32 (right panel) (Photo/Picture credit: Original).

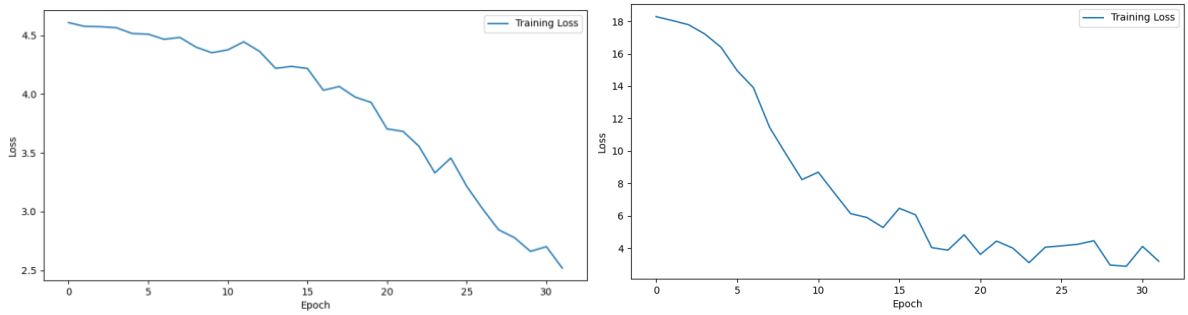

Figure 3. Results for num_local_steps = 2 (left panel) and num_local_steps = 8 (right panel) (Photo/Picture credit: Original).

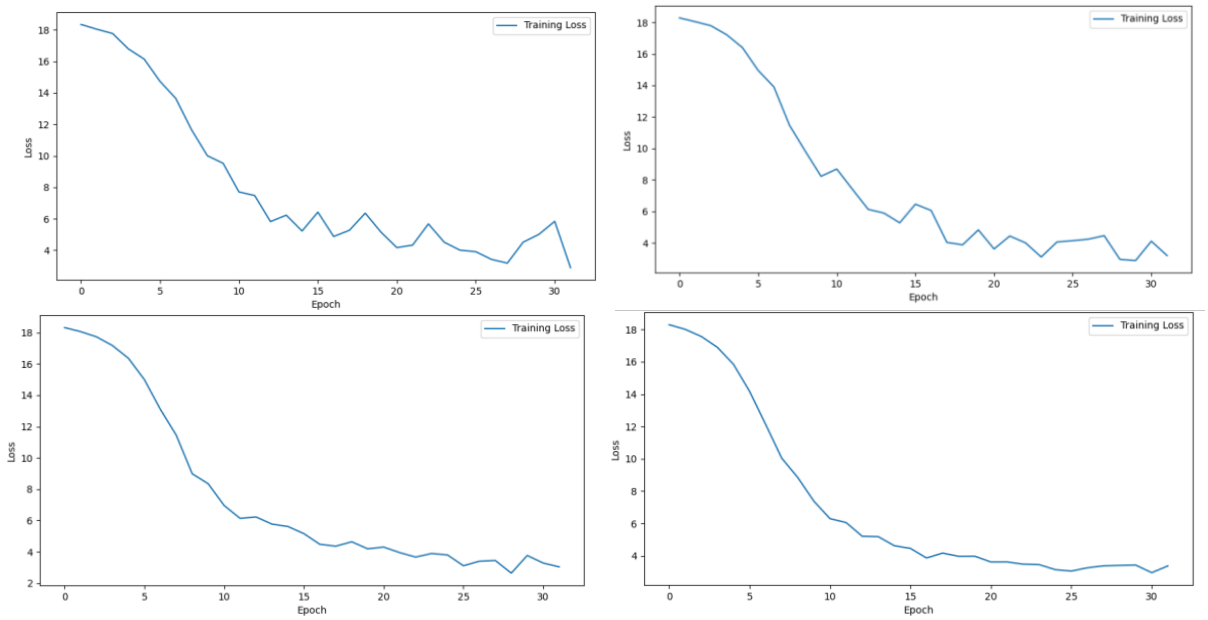

Figure 4. Loss functions for batch_size of 16 (left upper), 32 (right upper), 64 (left lower) and 128 (right lower) (Photo/Picture credit: Original).

3.2. Model performance

Model performance evaluation is an important aspect of machine learning and data science, helping us understand how a model performs in practical applications. Many hyperparameters will affect the performance of the model, e.g., learning_rate, num_local_steps. Sometimes this kind of

DOI: 10.54254/2755-2721/104/20241140

hyperparameters' influence on the model is unpredictable. With different num_local_steps setting, one can obtain the Fig. 3. Bigger num_local_steps make the model converge quicker and have better performance. The accuracies of them are 66.18% and 88.74%. Batch_size is another hyperparameters that will affect the performance of the model. Seen from Fig. 4, the accuracies of them are 87.72%, 88.74%, 89.98% and 89.75%. Actually, bigger batch size is not equal to better performance. There are issues like overfitting, while higher batch_size can make the training process more stable.

3.3. Discussion

There are some things worth attention. Trade-offs in Performance and Efficiency: While pursuing model performance, it is also necessary to consider training efficiency. Sometimes, to achieve higher performance, it may be necessary to sacrifice some training time. In situations with limited resources, a balance must be struck between model complexity and training time. Choosing appropriate hyperparameters based on the complexity of the problem and the characteristics of the data. For instance, deeper networks and more complex models may be required for nonlinear problems. Hyperparameter optimization often requires multiple experiments and iterations. Automated tools can be used to accelerate this process. Ultimately, the choice of hyperparameters should align with business objectives. For example, if real-time prediction is crucial, some accuracy may need to be sacrificed to reduce inference time. By considering these factors comprehensively, hyperparameter tuning can be conducted more effectively to achieve the best model performance and reasonable training time within limited resources.

3.4. Limitations and prospects

Due to time and resource constraints, the experiment may have tested only a few hyperparameters, without covering all possible combinations. This could lead to an incomplete understanding of the impact of hyperparameters. The experiment may have only targeted simple models and not considered more complex model structures, such as deep learning models, which may have different sensitivities to hyperparameters. Using a single dataset (like MNIST) may not fully reflect the impact of hyperparameters on different types of data. Relying solely on simple parameter settings and visualization methods may not provide an in-depth understanding of the intrinsic mechanisms of how hyperparameters affect the model.

In the future, one needs to test the same hyperparameters on different types of machine learning models to understand the sensitivity of different models to hyperparameters. Besides, scholars conduct experiments on multiple different datasets to verify the generalizability of hyperparameter settings. One also needs to use more advanced statistical methods and machine learning theories to analyze the impact of hyperparameters on model performance. Moreover, one needs to Improve the experimental design to ensure the reproducibility and reliability of the results. In addition, one needs to investigate how hyperparameters affect the internal mechanisms of the model to increase the explainability of the tuning process. Moreover, applying knowledge from other fields are also crucial (e.g., psychology, cognitive science) to hyperparameter tuning may discover new tuning methods.

4. Conclusion

To sum up, this study has demonstrated the impact of hyperparameter selection on model training. This research implemented a simple neural network model through code and altered fundamental parameters such as learning_rate, batch_size, num_local_upgrade, and num_communication to train different models. This research utilized visualization methods to show how model training time and performance change with the alteration of hyperparameters, and derived their respective accuracy rates. Currently, it is looking forward to using more advanced statistical methods and machine learning theories to analyze the impact of hyperparameters on model performance. It is planned to conduct experiments on multiple different datasets to verify the generalizability of hyperparameter settings, etc. This study provides introductory data and code for newcomers to the field of machine learning. Since this experiment involved a federated learning model, it utilized local upgrade SGD and also employed parallel methods,

DOI: 10.54254/2755-2721/104/20241140

making it quite comprehensive and covering many algorithm codes for machine learning beginners. Testing the basic parameters of the model can help newcomers to machine learning understand the impact of hyperparameters on the model more clearly. It also aids in the development of more algorithms for tuning hyperparameters.

References

[1] Jordan M I and Mitchell T M 2015 Machine learning: perspectives, Trends, and prospects, Science vol 349 pp 255-260
[2] Mahesh B 2019 Machine Learning Algorithms -A Review. International Journal of Science and Research (IJSR) vol 9 p 381
[3] Joost V, Matthijs W, Jonathan K, Jeroen K, Tim V and Jan S R 2020 A Survey on Distributed Machine Learning. ACM Comput. Surv. vol 53(2) p 30.
[4] Yuan J, Gao F, Ho Q, et al. 2015 Lightlda: Big topic models on modest computer clusters Proceedings of the 24th International Conference on World Wide Web pp 1351-1361
[5] Ge M, Bangui H and Buhnova B 2018 Big Data for Internet of Things: A Survey, Future Generation Computer Systems vol 87 p 602
[6] Zinkevich M, Weimer M, Li L and Smola A 2010 Parallelized stochastic gradient descent. In: Advances in neural information processing systems (NeurIPS), vol 4 p 4
[7] Kairouz P, McMahan HB, Avent B, Bellet A, Bennis M, Bhagoji A N, Bonawitz K, Charles Z, Cormode G, Cummings R and D'Oliveira R G 2019 Advances and open problems in federated learning ArXiv preprint arXiv:1912.04977
[8] Kuhn M and Johnson K 2013 Applied predictive modeling. New York: Springer.
[9] Young M T, Hinkle J, Ramanathan A and Kannan R 2018 HyperSpace: Distributed Bayesian Hyperparameter Optimization 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD) pp 339-347.
[10] Niklas L and Paul D 2006 Quantifying the impact of learning algorithm parameter tuning. In Proceedings of the 21st national conference on Artificial intelligence - vol 1 (AAAI'06) p 395–400.
[11] Probst P, Wright M N and Boulesteix A L 2019 Hyperparameters and tuning strategies for random forest. WIREs Data Mining Knowl Discov. vol 9 p e1301.
[12] Pieter G, Florian P, Jan N. van R, Bernd B and Joaquin V 2021 Meta-learning for symbolic hyperparameter defaults Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '21) pp 151–152