Optimizing the performance of the neural network by using a mini dataset processing method

Jingliang chen (✉️ 2274322195@qq.com)
Hubei University of Technology

chenchen wu
Hubei University of Technology

hao Li
Hubei University of Technology

Research Article

Keywords: mini data process, loss, iteration, deep neural network, optimize

Posted Date: November 4th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2202214/v1

License: ☛ This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License
Optimizing the performance of the neural network by using a mini dataset processing method

First Jingliang Chen\textsuperscript{1,1*}, Second Chenchen Wu\textsuperscript{1,1†} and Third Hao Li\textsuperscript{1,1†}

\*Hubei University of Technology, HongShan, WuHan, 430068, HuBei, China.

*Corresponding author(s). E-mail(s): chen@hbut.edu.cn;
Contributing authors: 102010965n@hbut.edu.cn;
102010973n@hbut.edu.cn;
†These authors contributed equally to this work.

Abstract

Data processing is one of the essential methods to optimize the performance of neural networks. In this paper, we give up the traditional data processing method and propose a method to optimize the deep neural network by processing the mini data set based on the loss. Using this method, each Iteration training can obtain a relatively optimal result, and the optimization effects of each time are integrated to optimize the results of each epoch finally. At the same time, in order to verify the effectiveness of this data processing method, experiments are carried out on MNIST, HAGRID, and CIFAR-10 datasets to compare the effects of using this method and not using this method under different hyperparameters, and finally, the effectiveness of this data processing method is verified. On this basis, the advantages and disadvantages of this method are summarized. Finally, the future research direction has been prospected.

Keywords: mini data process, loss, iteration, deep neural network, optimize
2 Article Title

1 Introduction

As a kind of artificial neural network, a deep neural network can independently construct (train) basic rules according to sample data in the Learning process. It has achieved excellent performance in more and more application fields, especially in the field of target detection[1], image recognition[2], software engineering[3], and so on[4]. Neural networks are trained by supervised learning, that is, by sample data and predefined results of sample data. In simple terms, deep learning is the imitation of neurons in the form of a layer to get the data characteristics and store features in an artificial neural network, and the difference is that the network is a black box[5], just like the human brain. Deep neural network models are generally composed of multiple layers, each containing a large number of neurons[6], and the weight of connections between these neurons is calculated through a large number of training and testing data. Deep learning models are mainly divided into a convolutional neural network (CNN) and recurrent neural network (RNN). CNN involves convolutional computation and is usually used to process data with grid-like topology like images[7]. On the other hand, the RNN model processes and makes decisions based on the information calculated from the existing data. Hence, it is more suitable for processing changing temporal information, such as speech, natural language, and other data containing sequence information[8]. We generally use software testing methods[9] to ensure the quality of deep neural network models and measure the quality of neural networks through loss function[10].

Generally, to improve the performance of the deep neural network, we can start from four aspects. The first is to improve performance through data, such as acquiring more data, which is the most direct and efficient way. In addition, we can scale the existing data: transform, feature selection, noise processing, reframe, and so on. The second approach is to improve the structure of the model. Such as increasing the number of layers of neural networks, improving performance through nested models that combine multiple "good enough" models to achieve excellent predictive power, or replacing simple neuron units with complex LSTM neurons, for example, in the field of natural language processing, using LSTM models to exploit the advantages of grammar analysis[11]. The third way is by adjusting the parameters of performance improvement, and this is a method of optimization of the neural network is inevitable, such as the initialization of an improved model, to ensure that the early gradient has a large number of sparse[12], or take advantage of the principle of linear algebra[13] to initialize the learning rate, the size of batch size, regularization coefficient, dropout coefficient, and the number of layers of the neural network. The final method is to choose a more robust learning algorithm, such as a way of updating the logarithm gradient or dividing the previous gradient L2 norm to update all parameters[14], Or using a Nonlinear time-delay system[15], or even choosing a second-order algorithm with high computational cost[16].

However, the neural network will be in the process of learning because of various reasons: noise data[17], data initialization[18], and other problems, the final training results and the predicted results are far from the results, and the
loss is too significant. Therefore, we need to optimize the model constantly. For example, combining automatic regression integration\cite{19}, stabilizing fuzzy relations\cite{20}... One common way to optimize is to use optimizers, a common form of optimizing the optimizer. During most of the same training time, the process of training and learning, especially small mill are often unable to get a large amount of data and also does not have a powerful machine, so we need to under the condition of the same data set, further pursue the convergence rate, namely under the same time, relatively higher precision and lower loss.

Based on this requirement, this paper proposes a widely applicable and micro-level improvement method: Limit Comparison Training in Literature (LCT), which is rooted in the first method of the above four categories of performance improvement methods. On the basis of different optimizers, the attention of optimization is focused on the results of each round inside each batch and each time. The quality of each result is measured through the data obtained by the loss function, and the bad effects are traced to the training process that brought the result. The main contributions are as follows:

1. This paper proposes a framework for optimizing the training performance of deep neural networks.
2. In this framework, the LCT method is proposed to select the relatively optimal mini data set in each epoch.
3. The loss is taken as the evaluation standard, and the cross-entropy loss function is determined to calculate the corresponding value.
4. The effectiveness of the LCT method for optimizing deep neural networks is verified for three different datasets.

The remainder of this paper is described as follows: Section II discusses related work. The third part describes the background, specific framework, and principle of the proposed method in detail. The fourth part describes the experimental data, related Settings, experimental process, variable analysis, and experimental results. Finally, we discuss and summarize our method in Section V and give some prospects.

2 Related Work

This work mainly involves three aspects:(1) Why put the optimization objective into the local mini dataset and do data processing during training; (2) The role of gradient optimizer in neural network training; (3) Application of cross-entropy loss function in the neural network.

2.1 Putting the optimization objective into the local mini dataset

In the training process of deep neural networks, the importance of the input dataset is beyond doubt. A high-quality dataset can speed up our convergence speed and reduce our loss. Therefore, to obtain better training results under selected datasets, various optimized processing methods\cite{21} for data have emerged, such as whitening, de-mean, PCA dimensionality reduction,
normalization, noise reduction, and so on. These methods have been verified by experiments and achieved lower loss results in the end. However, researchers do in the field of macro data processing in full flow but ignore the micro-data processing. Due to the extensive use of various open-source neural network libraries, such as TensorFlow, Keras, and PyTorch, we can build neural networks like building blocks. Still, it is also further and further away from this microscopic data processing method.

Ideally, the results of each neural network training are compared to the last time should be better. Because each round within each epoch iteration mini data set is different, some of the datasets have too much noise data, and so on reasons, leading to the final result curve fluctuation twists and turns, improving the data for this situation to achieve the final optimization results, That is, by finding mini datasets with less relative noise, the results we get are more superior.

At the same time, in the training of the neural network, the result is owned by the current in each round of training results calculated together, under the condition of this influence, this way of internal microdata processing after thousands of tens of thousands of times, several rounds of training, have soothing effects on the final result.

2.2 Deep Learning Optimization

1. Mini-batch Gradient Descent(mini-batch GD)

With the development of The Times and the promotion and popularity of the idea of Mini-batch, Stochastic Gradient Descent has been increasingly referred to as Stochastic Gradient Descent under the condition of Mini-batch in recent years. It is also called mini-batch Gradient Descent[22]. Generally, under different frameworks of current deep learning, the random Gradient Descent method is called the mini-batch Gradient Descent method by default. SGD below in this paper also refers to Mini-batch Gradient Descent without exceptional circumstances. Mini-batch Gradient Descent means that the training dataset is first shuffled and shuffled before training, and the training dataset is segmented with sizes such as 32,64,128. At each iteration, a part of the segmented dataset is selected for the Gradient Descent of the model. In other words, for m total training samples, X samples are used for iteration, and 1 \leq x \leq m, the update rule is:

$$\theta_i = \theta_i - \alpha \sum_{j=t}^{t+s-1} (y_j - h_\theta(x_j)) x_{ji}$$

Mini-batch Gradient Descent combines the advantages of Batch Gradient Descent and Stochastic Gradient Descent to accelerate the neural network’s convergence speed and improve accuracy simultaneously. In the current development process of deep learning, Hinton et al.[23], Deng et al.[24], and Graves et al.[25], all verified the effectiveness of Mini-batch Gradient Descent in their papers.

2. Adaptive Moment Estimation(Adam)
Diederik P. Kingma et al. proposed the Adaptive Moment Estimation algorithm\cite{26} in 2013, which is a combination of the Moment algorithm and RMSProp algorithm\cite{27}. Adam not only stores the exponentially decaying average value VT of the past square gradient as the RMSProp algorithm does, but also keeps the exponentially decaying average value MT of the past gradient, similar to the momentum algorithm:

\[
\begin{align*}
    m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\
    v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2
\end{align*}
\]

Mt and VT are the estimates of the first moment (mean) and the second moment (non-central variance) of the gradient, respectively. The default value is close to 1 and the deviation is close to 0. The deviation is improved by calculating the first and second-moment estimates of the bias correction:

\[
\begin{align*}
    \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\
    \hat{v}_t &= \frac{v_t}{1 - \beta_2^t}
\end{align*}
\]

These are then used to update the parameters, resulting in Adam’s update rule:

\[
\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t
\]

In 2017, Richard Socher et al.\cite{28} found that although the Adam algorithm has shown faster convergence speed and higher final accuracy in countless neural network training processes, researchers found that after testing on the CIFAR-10 dataset, the convergence speed of Adam is faster than that of SGD (Mini-batch GD), but the final convergence result is not as good as SGD, which also leads to the fact that at present, the frequency of neural network models with high requirements for absolute accuracy is higher than that of Adam compared with SGD.

2.3 Loss Function

The difference between the predicted value of a machine learning model and the actual value for a single sample is called the loss. The smaller the loss, the better the model. If the predicted value equals the actual value, there is no loss. The function used to calculate the loss is called the loss function. A loss function is used to measure how good the model is at each prediction. These functions are essentially the same function that computes the difference between the predicted value and the actual value and then is wrapped in a library to form a function with a specific name.

In machine learning, we want the predicted value to be infinitely close to the actual value, so we need to minimize the difference, and in this process, we need to introduce a loss function. In specific projects, some loss functions decrease the gradient of the difference calculated by the gradient fast,
6 Article Title

while some decrease slowly, so choosing the appropriate loss function is also very critical. Currently, the commonly used loss functions in the training of classification neural networks include 0-1 loss, Logistics loss, Hinge loss[29], exponential loss[30], and cross-entropy loss[31]. In this paper, the loss function we use is the cross-entropy loss function. The cross-entropy loss function can be divided into binary cross-entropy loss function based on Sigmoid and multi-classification cross-entropy loss function based on Softmax.

Cross-entropy loss functions are divided into binary cross-entropy loss functions based on sigmoid and multi-class cross-entropy loss functions based on Softmax. In the case of binary classification, there are only two cases that the model needs to predict. For each class, the probability of our prediction is $\rho$ and $1 - \rho$. In this case, our function expression is:

$$L = \frac{1}{N} \sum_{i} L_i = -\frac{1}{N} \sum_{i} [y^i \log(\rho^i) + (1 - y^i) \log(1 - \rho^i)]$$

In this expression, $y^i$ is the label, and $\rho^i$ is the predicted value.

The multiclass classification case is actually an extension of binary classification, and the function expression is:

$$L = \frac{1}{N} \sum_{i} L_i = -\frac{1}{N} \sum_{i} \sum_{c} y_{ic} \log(\rho_{ic})$$

In this expression, $M$ denotes the number of categories, $y_{ic}$ denotes the label of category $C$, and $\rho_{ic}$ is the predicted value of category $C$.

3 Methodology

This section describes the research problem, principle, and implementation framework of the LCT method in detail.

3.1 Problem model

Take the ANN neural network for solving multi-classification problems as an example:
From the input of the data set to the training of the hidden layer and the final result output constitutes the simplest ANN neural network. There are some problems that we always need to solve: the final prediction accuracy of the neural network is not high enough, and the loss to measure the quality of the neural network is too high.

As we can see from the model graph, we can’t optimize the middle part of the training very well, but in the blue wire frame data processing, input, and so on function and the function of the black box actually a portion of the cross, we can see this part of the analysis, in the inside of the mini input process makes the corresponding data processing, Ignoring the incomprehensible calculation process, starting from the final results, and finally achieving the purpose of optimizing the whole model.

3.2 Principle frame

As mentioned above, the core idea of the LCT method is: We will focus our attention on dataset optimization on the training data of each iteration, actively focus our attention on this “micro-field,” and then add a data processing layer to it. The main work of this data processing layer is to train the data of each iteration during the training process. The MINI dataset with relatively minor noise is selected, and finally, the selected data is used to retrain and learn.

To reach this end, we solve the following two problems: 1. How to conduct data processing and select the relatively optimal MINI data set; 2. How to train and learn after the MINI data is processed.

1. How to conduct data processing and select the relatively optimal MINI data set:

The above model diagram shows that the LCT method’s main functional modules are divided into two parts: the random shuffling grouping module and the mini dataset processing module, respectively.

In the random shuffling grouping module, we add random seeds in the processing part of the macro data set and then shuffle our data set. According to the stochastic gradient descent optimizer, the data set is re-shuffled after every epoch, and the whole training process is successfully differentiated into the structure of epoch and iteration.
8 Article Title

Where epoch refers to when a complete data set passes a training of the neural network and returns once, this process is called an epoch. (That is, all training examples go through one forward propagation and one back propagation through the neural network.) Informally, an Epoch is a process of training all training examples through the model once. The primary purpose of setting epochs is to divide the whole training process of the model into several segments so that we can better observe and adjust the model training.

When the number of samples (i.e., all training samples) for an Epoch may be too large (for a computer), we need to split it into smaller chunks, i.e., batches, for training. A batch is composed of several pieces of data, which is the basic unit of network optimization. Each round of network parameter optimization needs to use a batch. Samples in a batch are processed in parallel. Compared with a single sample, a batch of data can better simulate the distribution of the data set. The larger the batch is, the better the simulation of the input data distribution, which is reflected in the network training, which can make the direction of network training “more correct.”

\[
\text{num of batches} = \frac{\text{training dataset size}}{\text{batch size}}
\]  

Training once with a Batch of data is an iteration, the process of updating the parameters of the model, called “training once.”

The MINI data set processing module is the core module of our LCT method. In this module, we first input the mini data set into the network and select the cross-entropy loss function to calculate the loss corresponding to each iteration. Record the obtained loss with weight parameters and other hyperparameters. Secondly, the loss value obtained from MINI dataset A is compared with the loss value obtained from Mini dataset B. When the loss value obtained from Mini dataset A is greater than the loss obtained from Mini dataset B, it means that Mini dataset B is better than Mini dataset A. Let’s replace the contents of Mini dataset A with the contents of Mini dataset B; Finally, this operation is repeated until the end of the epoch of this round, when all iteration mini datasets of this round have been processed and transformed. The formula is as follows:

\[
\text{minibatch} = \{\text{pre}_{\text{loss}} < \text{loss} \? \text{pre}_{\text{minibatch}} : \text{minibatch}\}
\]

2. How to train and learn after the MINI data is processed:

Returning to the data input step of the neural network, the processed mini data is re-input into the network, and the previous record nodes are visited at the same time, and all the weight parameters and hyperparameters such as W, B, learning rate, Grads, etc. are restored to the pre-training state. After the recovery is completed, the current network is re-trained.

The optimization processing through the Limit Comparison Training method can accelerate the convergence speed of the neural network and
improve the accuracy to a certain extent in theory. For the fixed training model, the dataset with more noise samples will get better results using this method.

4 Experiments

In this section, three datasets are used for experimental comparison. The effectiveness and applicability of the LCT method are verified by comparing the results of convergence speed, loss, and accuracy between neural network models using the LCT method and those without the LCT method in different data sets, different batch sizes, and different learning rates.

4.1 Dataset

The open source datasets used in this paper are MNIST Dataset, Hand Gesture Recognition Image dataset (HaGRID) and CIFAR-10 dataset.

4.1.1 MNIST

The MNIST dataset is a dataset of images of handwritten digits sponsored by the National Institute of Standards and Technology (NIST). The images were taken from 250 different people, 50 percent of whom were high school students and 50 percent of whom were Census Bureau workers. In 1998, Yan LeCun et al. published the first LeNet-5 network and implemented handwriting recognition using the MNIST dataset, with 60,000 images and labels in the training set and 10,000 in the test set. Since 1998, this dataset has been widely used in machine learning and deep learning to test the performance of algorithms.

MNIST
4.1.2 Hand Gesture Recognition Image Dataset (HaGRID)

Hand Gesture Recognition Image Dataset (HaGRID) dataset size is 716GB, contains 552,992 FullHD (1920 × 1080) RGB images, divided into 18 categories of gestures. The annotation consists of the bounding box of the hand with the gesture label and the marker of the leading hand. The data is split into 92% training set and 8% test set, where 509,323 images are used for training and 43,669 images for testing.

![HaGRID](image)

4.1.3 CIFAR-10

CIFAR-10 is a small dataset for identifying pervasive objects curated by Hinton’s students Alex Krizhevsky and Ilya Sutskever. There are ten categories of RGB color tiles: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The size of the images is 32×32, and there are 50,000 training images and 10,000 test images in the dataset. A sample image of CIFAR-10 is shown in Figure. The dataset consists of 60,000 32×32 color images with ten categories—6000 images per category. There are 50,000 training images and 10,000 test images. Compared with the above two datasets, CIFAR-10 contains natural objects in the real world, which are not only noisy but also have different proportions and features, which brings great difficulties to the recognition.
4.2 Parameter Setting

The parameters and hyperparameters in this experimental process are distributed in each process of neural network training.

1. Initialize the weight parameters W and b: use the random distribution function to assign a set of random sample values that follow a uniform distribution of 0 to 1 to the W of each layer, and set the b of each layer to an array of 0 of the given shape.

2. Initialization of the Adam optimizer’s order one hyperparameter V and order two hyperparameter S: Both are initialized to 0 matrices of the order corresponding to the weight parameters.

3. Correction parameters k1 and k2 for Adam optimizer and offset u: \( k1 = 0.9, k2 = 0.999, u = e^{-8} \)

4. Setting of mini-batch, batch size: Generally, the value of batch size is an exponent of 2. In this experiment, we adjust its value to 32, 64, and the mini-batch data set is shuffled with a random index and then grouped according to the size of the batch size.

5. Learning rate: The learning rate is generally set below 0.1, and the initial learning rate of this implementation is adjusted to \( \{10^{-3}, 7 \times 10^{-4}\} \).

In all the experimental comparisons, the hyperparameters are either based on the experimental requirements or the original paper.
4.3 Performance Comparison

All experiments in this paper are based on the Tensorflow2.x framework, and the language is Python. Based on the optimizers SGD and Adam, the loss function uses the multi-class cross-entropy loss function.

After finishing the preliminary parameter Settings, we under the two different datasets by changing the batch size and the size of the vector and carrying on the contrast experiment many times. At the same time, losses will be recorded in the experiment process through the image in the form of a display, including blue, which doesn’t use the LCT method, and red, which uses the LCT method. The contrast images are always SGD first and ADAM second.

First, the experiments on the MNIST dataset, epoch = 10:
R=0.001, batch size = 32:

![Graph 1: R=0.001, batch size = 32](image1)

**Fig. 1:** R=0.001, batch size = 32

R=0.001, batch size = 64:

![Graph 2: R=0.001, batch size = 64](image2)

**Fig. 2:** R=0.001, batch size = 64
R=0.0007, batch size = 32:

![Graphs](image1)

**Fig. 3:** R=0.0007, batch size = 32

R=0.0007, batch size = 64:

![Graphs](image2)

**Fig. 4:** R=0.0007, batch size = 32

Second, the experiments on the HAGRID dataset:
R=0.001, batch size = 32:

![Graphs](image3)

**Fig. 5:** R=0.001, batch size = 32

R=0.001, batch size = 64:
14 Article Title

Fig. 6: \(R=0.001\), batch size = 64

R=0.0007, batch size = 32:

Fig. 7: \(R=0.0007\), batch size = 32

R=0.0007, batch size = 64:

Fig. 8: \(R=0.0007\), batch size = 32

Finally, the experiments on the CIFAR-10 dataset, epoch=20:
R=0.001, batch size = 32:
Fig. 9: $R=0.001$, batch size = 32

R=0.001, batch size = 64:

Fig. 10: $R=0.001$, batch size = 64

R=0.0007, batch size = 32:

Fig. 11: $R=0.0007$, batch size = 32

R=0.0007, batch size = 64:
Through the above group comparison experiments under different data sets, we can see that the results obtained by using LCT method are significantly better than those obtained without using this method. Of course, our time consumption is inevitably relatively higher due to the addition of mini data processing operations internally. Taking the last dataset CIFAR-10 as an example, we find that the time taken without the proposed method is 4h44.8m, while the time taken with the LCT method is 5h15.4m.

4.4 Experimental Results

After the completion of the experiment, we counted the loss results of all comparative tests and calculated the prediction accuracy of the model, which was visually displayed in the form of a chart. The precision is calculated as follows:

\[
\text{accuracy} = \frac{\sum_{i=0}^{n} \{y^i = \rho^i \ ? \ 1 : 0\}}{n} \quad (3)
\]

4.4.1 Performance of MNIST

Firstly, the MNIST dataset was used for the experiment, and the test was carried out under the learning rate of 0.0007 and 0.001 respectively. At the same time, the two batch sizes of 32,64 training were corresponded, and the epoch was fixed as 10 times. After the training, the final results were compared through the graphical structure.

experiment 1: Learning rate = 0.001, batch size = 32:

<table>
<thead>
<tr>
<th>MNIST</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.1181</td>
<td>0.0489</td>
</tr>
<tr>
<td>accuracy</td>
<td>74.01%</td>
<td>87.92%</td>
</tr>
</tbody>
</table>

Table 1: the result1 of MNIST
loss optimization effect:
SGD: 0.098——82.98%  Adam: 0.0354——72.39%
Accuracy optimization effect:
SGD: 14.73%  Adam: 2.56%

experiment 2: Learning rate = 0.001, batch size = 64:

<table>
<thead>
<tr>
<th>MNIST</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.0801</td>
<td>0.0224</td>
</tr>
<tr>
<td>accuracy</td>
<td>68.49%</td>
<td>90.26%</td>
</tr>
</tbody>
</table>

loss optimization effect:
SGD: 0.043——53.68%  Adam: 0.0167——74.55%
Accuracy optimization effect:
SGD: 6.59%  Adam: 2.28%

experiment 3: Learning rate = 0.0007, batch size = 32:

<table>
<thead>
<tr>
<th>MNIST</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.1365</td>
<td>0.0452</td>
</tr>
<tr>
<td>accuracy</td>
<td>73.28%</td>
<td>87.70%</td>
</tr>
</tbody>
</table>

loss optimization effect:
SGD: 0.0994——72.82%  Adam: 0.0373——82.52%
Accuracy optimization effect:
SGD: 11.01%  Adam: 8.2%

experiment 4: Learning rate = 0.0007, batch size = 64:

<table>
<thead>
<tr>
<th>MNIST</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.0421</td>
<td>0.0221</td>
</tr>
<tr>
<td>accuracy</td>
<td>41.56%</td>
<td>88.05%</td>
</tr>
</tbody>
</table>

loss optimization effect:
SGD: 6.37%  Adam: 2.9%
**Table 4**: the result of MNIST

<table>
<thead>
<tr>
<th>MNIST</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.1013</td>
<td>0.0251</td>
</tr>
<tr>
<td>accuracy</td>
<td>68.51%</td>
<td>90.1%</td>
</tr>
</tbody>
</table>

**4.4.2 Performance of HAGRID**

Second, the experiment was carried out using gesture data set (HGR), and the test was carried out at a learning rate of 0.0007 and 0.001. At the same time, it corresponded to two different batch sizes of 32, 64, and the epoch was fixed at 20,000 times.

experiment 1: Learning rate = 0.001, batch size = 32:

**Table 5**: the result of HAGRID

<table>
<thead>
<tr>
<th>HAGRID</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.1237</td>
<td>0.1486</td>
</tr>
<tr>
<td>accuracy</td>
<td>88.67%</td>
<td>93%</td>
</tr>
</tbody>
</table>

loss optimization effect:
SGD: 0.0464 —— 37.51%  Adam: 0.1171 —— 78.8%

Accuracy optimization effect:
SGD: 4.66%  Adam: 1.67%

experiment 2: Learning rate = 0.001, batch size = 64:

**Table 6**: the result of HAGRID

<table>
<thead>
<tr>
<th>HAGRID</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.1014</td>
<td>0.1001</td>
</tr>
<tr>
<td>accuracy</td>
<td>88%</td>
<td>93.67%</td>
</tr>
</tbody>
</table>

loss optimization effect:
SGD: 0.0401 — 39.55%  Adam: 0.036 — 35.96%
Accuracy optimization effect:
SGD: 0.33%  Adam: 2%

experiment 3: Learning rate = 0.0007, batch size = 32:

<table>
<thead>
<tr>
<th>HAGRID</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.1526</td>
<td>0.1501</td>
</tr>
<tr>
<td>accuracy</td>
<td>86.67%</td>
<td>94%</td>
</tr>
</tbody>
</table>

loss optimization effect:
SGD: 0.0792 — 51.9%  Adam: 0.1192 — 79.41%
Accuracy optimization effect:
SGD: 5%  Adam: 2.67%

experiment 4: Learning rate = 0.0007, batch size = 64:

<table>
<thead>
<tr>
<th>HAGRID</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.1179</td>
<td>0.1021</td>
</tr>
<tr>
<td>accuracy</td>
<td>79.67%</td>
<td>92.65%</td>
</tr>
</tbody>
</table>

loss optimization effect:
SGD: 0.0264 — 22.39%  Adam: 0.037 — 36.24%
Accuracy optimization effect:
SGD: 1.66%  Adam: 1.68%

4.4.3 Result of CIFAR-10

Finally, CIFAR-10 was used for experiments, and the learning rates of 0.001 and 0.0007 were tested. At the same time, it corresponded to two different batch sizes of 32, 64, and the epoch was fixed at 20 times. After the training, the final results were compared through the graphical structure.

experiment 1: Learning rate = 0.001, batch size = 32:
loss optimization effect:
### Table 9: the result of CIFAR-10

<table>
<thead>
<tr>
<th>CIFAR-10</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.0556</td>
<td>0.0473</td>
</tr>
<tr>
<td>accuracy</td>
<td>35.84%</td>
<td>43.08%</td>
</tr>
</tbody>
</table>

SGD: 0.0464 —— 83.45%  Adam: 0.045 —— 95.13%

Accuracy optimization effect:
SGD: 46%  Adam: 28.36%

experiment 2: Learning rate = 0.001, batch size = 64:

### Table 10: the result2 of CIFAR-10

<table>
<thead>
<tr>
<th>CIFAR-10</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.0279</td>
<td>0.0235</td>
</tr>
<tr>
<td>accuracy</td>
<td>24.66%</td>
<td>43.74%</td>
</tr>
</tbody>
</table>

Loss optimization effect:
SGD: 0.0141 —— 50.53%  Adam: 0.0182 —— 77.45%

Accuracy optimization effect:
SGD: 35.84%  Adam: 21.53%

experiment 3: Learning rate = 0.0007, batch size = 32:

### Table 11: the result3 of CIFAR-10

<table>
<thead>
<tr>
<th>CIFAR-10</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.0543</td>
<td>0.0446</td>
</tr>
<tr>
<td>accuracy</td>
<td>36.75%</td>
<td>45.56%</td>
</tr>
</tbody>
</table>

Loss optimization effect:
SGD: 0.0454 —— 83.61%  Adam: 0.0414 —— 92.83%

Accuracy optimization effect:
SGD: 21.74%  Adam: 25.9%

experiment 4: Learning rate = 0.0007, batch size = 64:

<table>
<thead>
<tr>
<th>CIFAR-10</th>
<th>Without LCT</th>
<th>With LCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGD</td>
<td>Adam</td>
</tr>
<tr>
<td>loss</td>
<td>0.0287</td>
<td>0.0217</td>
</tr>
<tr>
<td>accuracy</td>
<td>34.4%</td>
<td>47.52%</td>
</tr>
</tbody>
</table>

loss optimization effect:
SGD: 0.012 —— 41.81%  Adam: 0.02 —— 92.17%

Accuracy optimization effect:
SGD: 22.3%  Adam: 14.76%

Based on the data shown in the above images and tables, we can clearly find that the final result is always better than that obtained by not using the LCT method in any data set. Both loss and accuracy have a specific optimization effect.

### 5 Discussion and Conclusion

In this paper, we propose a Limit Comparison Training method based on iteration in batch training to solve the loss in the process of neural network model training. By comparing the loss of this iteration and the previous iteration, the input feature of which time is selected to be optimized. And through experiments, we verify the effectiveness of using the LCT method on neural network optimization effect under three different data sets: MNIST dataset, HAGRID dataset, and CIFAR-10 dataset. Under different learning rates, batch sizes, and other conditions, the average optimization effect of loss in MNIST is SGD: 62.76%; Adam: 79.38%, Accuracy: 9.68%; Adam: 3.99%. The average optimization effect of loss in HAGRID is 37.84% for SGD; Adam: 57.6%, average accuracy improvement of SGD: 2.91%; Adam: 2.01%. The average optimization effect of loss in CIFAR-10 is 64.85% for SGD; Adam: 89.4%, Accuracy: 31.47%; Adam: 22.64%. In the end, it can be easily seen that the LCT method can achieve the optimization effect of reducing loss and improving accuracy, especially on CIFAR-10. It is proved that our LCT method has applicability to different datasets, and at the same time, the more noise data, the better the effect.

Of course, due to the limitations of Iteration itself and the choice of data processing methods, our experiments consume relatively more time. In future
work, we will actively map out an optimal mini dataset based on literation and combine it with Hybridised loss function to obtain more accurate loss and achieve better results.

6 Declarations

- Conflict of interest: The authors declare that they have no conflict of interest.
- Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

References


24 Article Title


