Optimizer Selection for Convolutional Neural Networks in Handwritten digit recognition

Shuangyu Zhao, Wenting Liu, Yuhan Zhu, Tiancheng Liu, Yan Lyu

Abstraction

This project focused on optimizing Convolutional Neural Networks (CNNs) for handwritten digit recognition using different optimization techniques, including Stochastic Gradient Descent (SGD), Momentum, RMSprop, and Adam. The goal is to achieve high accuracy while reducing the computational complexity and memory requirements of CNNs. The project uses the MNIST dataset and compares the performance of the different optimization techniques using evaluation methods such as accuracy, recall, precision, and F1 score. The results indicate that RMSprop is the best-performing optimizer, achieving the highest evaluation scores and the lowest convergence iteration epochs. The study has practical implications for developing efficient CNN-based systems in various applications.

1. Introduction

The objective of both machine learning and neural network models is prediction, and a crucial aspect of achieving better predictions is identifying a dependable optimizer.

The MNIST dataset, consisting of 70,000 handwritten digit images, with 60,000 images in the training set and 10,000 images in the testing set, has been used extensively for evaluating and comparing various machine learning algorithms, particularly for image classification tasks.

Convolutional Neural Networks (CNNs) have shown remarkable success in image recognition[1]. However, the high computational cost and memory requirements[2] of CNNs have become a major challenge. Therefore, the optimization of CNNs is crucial to reduce the computational complexity and memory footprint while maintaining their accuracy.

In this project, we aim to study different techniques for optimizing CNNs, including Stochastic Gradient Descent (SGD), Momentum, RMSprop, and Adam. Our goal is to find the best combination of these techniques that can achieve high accuracy on handwritten digits detection while reducing the size and complexity of CNNs. The results of this study can have practical implications for the development of efficient CNN-based systems in various applications[3].

2. Optimizer description

The optimizer in CNN is used to calculate the matrix for convolution, the weights and bias in every layer. The structure of CNN is shown in figure 1. For the first layer of the CNN, convolution and ReLU activation function is applied on input X. The math equations are shown below, the output of the first layer is $A_1 \in R^{n\times 128}$.

$$Z_1 = X * f, \text{ where } X \in R^{n\times 784},\ f \in R^{657\times 1}$$
$$(A_1)_{ij} = ReLU(Z_1)_{ij} = max(0, Z_{ij})$$

For the second layer, linear transformation is applied first, and then sigmoid activation function is applied. The output of the second layer is $A_2 \in R^{n\times 64}$.

$$(Z_2)_{i.} = (A_1)_{i.} \cdot W_2 + b_2, \text{ where } W_2 \in R^{128\times 64},\ b_2 \in R^{1\times 64}$$
$$(A_2)_{ij} = sig(Z_2)_{ij} = \frac{1}{1+e^{-(Z_2)_{ij}}}$$

For the output layer, linear transformation is also applied to reduce the column number to 10, and then softmax activation function is applied.

$$(Z_3)_{i.} = (A_2)_{i.} \cdot W_3 + b_3, \text{ where } W_2 \in R^{64\times 10},\ b_2 \in R^{1\times 10}$$

$$(Y)_{ij} = soft(Z_3)_{ij} = \frac{e^{(Z_3)_{ij}}}{\sum_{j=1}^{10} e^{(Z_3)_{ij}}}$$

Therefore, the optimization problem of our project is:

$$\min_{f, W_2, b_2, W_3, b_3} \sum_{i}^{nrow} \| Y_{i.} - \{sig[ReLU(X * f)_{i.} \cdot W_2 + b_2] \cdot W_3 + b_3\} \|_2^2$$

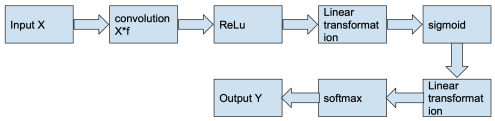

Figure 1. CNN structure we build

As for the 4 optimizers we focus on, Momentum, RMSprop and Adam are developed from SGD. Therefore, Let's start with SGD

2.1. SGD

The basic theory of SGD is randomly choosing subsets from the original dataset, calculating the parameters in every subset by gradient descent method, and finally calculating the average values for every parameter. The average parameters are the final parameters of the CNN model.

In this project, batch size = 200 is chosen. Multiple groups of 200 rows are randomly selected from X, and applied gradient descent method. The background math equations are shown below. **k** in the following equations means the parameters of the k_{th} subset.

$$f_k := f_k - t \frac{\partial \sum_{i}^{nrow} \|Y_{i.} - \{sig[ReLU(X*f)_{i.} \cdot W_2 + b_2] \cdot W_3 + b_3\}\|_2^2}{\partial f}$$

$$W_{2k} := W_{2k} - t \frac{\partial \sum_{i}^{nrow} \|Y_{i.} - \{sig[ReLU(X*f)_{i.} \cdot W_2 + b_2] \cdot W_3 + b_3\}\|_2^2}{\partial W_2}$$

$$b_{2k} := b_{2k} - t \frac{\partial \sum_{i}^{nrow} \|Y_{i.} - \{sig[ReLU(X*f)_{i.} \cdot W_2 + b_2] \cdot W_3 + b_3\}\|_2^2}{\partial b_2}$$

$$W_{3k} := W_{3k} - t \frac{\partial \sum_{i}^{nrow} \|Y_{i.} - \{sig[ReLU(X*f)_{i.} \cdot W_2 + b_2] \cdot W_3 + b_3\}\|_2^2}{\partial W_3}$$

$$b_{3k} := b_{3k} - t \frac{\partial \sum_{i}^{nrow} \|Y_{i.} - \{sig[ReLU(X*f)_{i.} \cdot W_2 + b_2] \cdot W_3 + b_3\}\|_2^2}{\partial b_3}$$

$$f = avg(f_k)$$

$$W_2 = avg(W_{2k})$$

$$b_2 = avg(b_{2k})$$

$$W_3 = avg(W_{3k})$$

The python package(keras) is directly used to apply SGD optimizer. Figure 2 visualizes the model estimation. The accuracy of this model is 0.9363. And the model converges when epoch = 5.

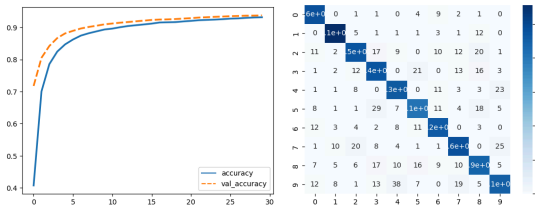

Figure 2. Model estimation of CNN model with SGD optimizer
(a. Confusion matrix; b. Accuracy changes across epochs)

Table 1. Precision-recall-f1 score of every class in CNN with SGD optimizer

class	precision	recall	F1 score	support
0	0.9478	0.9816	0.9644	980
1	0.9720	0.9789	0.9754	1135
2	0.9425	0.9205	0.9314	1032
3	0.9146	0.9327	0.9235	1010
4	0.9237	0.9491	0.9362	982
5	0.9298	0.9058	0.9177	892
6	0.9443	0.9552	0.9497	958
7	0.9374	0.9319	0.9345	1028
8	0.9193	0.9127	0.9160	974

2.2. Momentum

In addition to SGD, we also worked on the Momentum optimizer. In physics, momentum is a term describing the intended quality of motion that an object has in a specific direction. This direction of momentum does not always have the same direction as the movement direction. People carried over that idea to optimization technique, where they take a fraction of the update vector from the previous iteration, and add it to the current update. This can help the optimizer overcome things like saddle points or local minimum.

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

As for the update rule of Momentum, we have:
γ(Gamma): momentum parameter
η(Eta): learning rate
θ(Theta): current value for model parameter
v_t : update vector at time step "t"

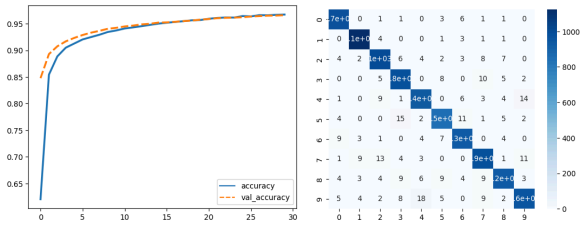

Figure 3. Model estimation of CNN model with Momentum optimizer (a. Accuracy changes across epochs; b. Confusion matrix)

Table 2. Precision-recall-f1 score of every class in CNN with Momentum optimizer

class	precision	recall	F1 score	suppor
0	0.9719	0.9867	0.9792	980
1	0.9817	0.9912	0.9864	1135
2	0.9623	0.9651	0.9637	1032
3	0.9570	0.9703	0.9636	1010
4	0.9623	0.9613	0.9618	982
5	0.9605	0.9552	0.9578	892
6	0.9657	0.9708	0.9682	958
7	0.9591	0.9591	0.9591	1028
8	0.9685	0.9476	0.9580	974
9	0.9676	0.9475	0.9574	1009

We are able to achieve an average accuracy of 0.9655 with the momentum optimizer.

2.3 RMSprop(Root Mean Square Propagation)

RMSprop optimizer can be thought of as an improved version of gradient descent algorithm, which adapts the learning rate to improve the training efficiency and convergence speed of the model. It adapts the learning rate adaptively based on the gradient history of each parameter. The advantage of PMSprop is avoiding the problems of learning rate being too large or too small. That's why RMSprop is more effective than traditional gradient descent algorithms.

The core idea of RMSprop is to divide the exponential moving average of the squared gradients of each parameter by the gradient itself. This exponential moving average can be regarded as the average value of the gradient square for the parameter.

Specifically, for each parameter $\boldsymbol{w}$, RMSprop maintains a variable $\boldsymbol{v}$ to store the exponential moving average of the squared gradients of $\boldsymbol{w}$. At each time step $\boldsymbol{t}$, the algorithm updates $\boldsymbol{v}$ using the following formula:

$$v_t = \beta v_{t-1} + (1 - \beta) \times [\frac{\partial L}{\partial w_t}]^2$$

After apply RMSprop optimizer, the result of visualization show below

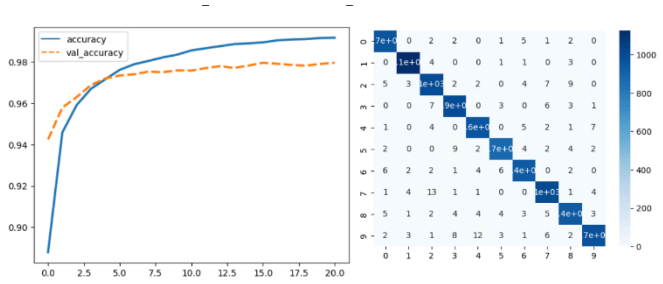

Figure 4. Model estimation of CNN model with RMSprop optimizer(a. Accuracy changes across epochs; b. Confusion matrix)

Table 3. Precision-recall-f1 score of every class in CNN with RMSprop optimizer

class	precision	recall	F1 score	support
0	0.9808	0.9898	0.9853	980
1	0.9912	0.9921	0.9916	1135
2	0.9691	0.9738	0.9715	1032

3	0.9791	0.9762	0.9777	1010
4	0.9796	0.9796	0.9796	982
5	0.9657	0.9787	0.9722	892
6	0.9791	0.9802	0.9797	958
7	0.9786	0.9786	0.9786	1028
8	0.9690	0.9641	0.9665	974
9	0.9848	0.9653	0.9750	1009

2.4. Adam

Adaptive Moment Estimation optimizer is an optimization algorithm used for updating the weights of neural networks during training. It is a variant of SGD that uses adaptive learning rates for different parameters. The advantage of Adam is that it combines the benefits of other optimization methods. Such as, RMSprop and Gradient descent with momentum algorithms. Adam is effective in high-dimensional data, such as images. It is faster and more reliable.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)[\frac{\partial L}{\partial w_t}]v_t$$
$$m_t = \beta_2 v_{t-1} + (1 - \beta_2)[\frac{\partial L}{\partial w_t}]^2$$

There are some disadvantages and limitations for Adam. It is sensitivity to hyperparameters. It tends to adapt too quickly to the noise of small datasets, which can lead to poor performance.

We used the Adam algorithm from python package keras. Image 5 visualizes the model performance. The accuracy of this model is 0.9765.

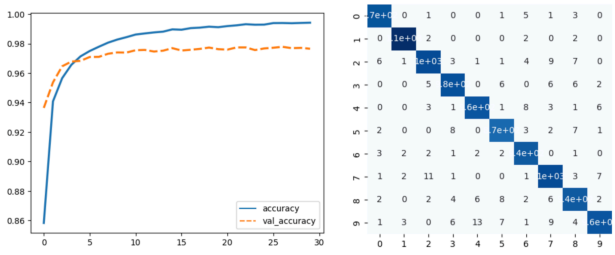

Figure 5. Model estimation of CNN model with Adam optimizer(a. Accuracy changes across epochs; b. Confusion matrix)

Now, in terms of class performance, the below table shows the precision, recall, and F1 score.

Table 4. Precision-recall-f1 score of every class in CNN with Adam

class	precision	recall	F1 score	support
0	0.9848	0.9888	0.9868	980
1	0.9930	0.9947	0.9938	1135
2	0.9747	0.9690	0.9718	1032
3	0.9762	0.9752	0.9757	1010
4	0.9776	0.9766	0.9771	982
5	0.9709	0.9742	0.9726	892
6	0.9732	0.9864	0.9798	958
7	0.9653	0.9747	0.9700	1028
8	0.9652	0.9671	0.9662	974
9	0.9817	0.9564	0.9689	1009

3. Conclusion and Discussion

3.1 Conclusion

For final results comparison, we applied 4 different evaluation methods, which are accuracy, recall, precision and F1 score. We also compared the convergence iteration epochs of each optimizer.

As we can see in the below table, the best performance optimizer is RMSprop which has highest evaluation scores among all kinds of methods as well as lowest convergence iteration epochs.

Table 5. Evaluation of optimizers

Optimizer	Accuracy	Recall	Precision	F1	Convergence Iteration Epochs
SGD	0.9373	0.9376	**0.9376**	0.9375	5
Momentum	0.9656	0.9655	**0.9657**	0.9655	5
RMSprop	***0.9780***	***0.9778***	***0.9777***	***0.9778***	***2.5***
ADAM	0.9763	0.9763	**0.9763**	0.9765	3

3.2 Discussion

Although we have achieved relatively high evaluation scores, one of the real world applications of this project is about handwritten recognition which is used for bank risk management. For example, the handwritten is our signature, which can be used for debit card cash withdrawal or check payment. It is ok that our true signature has not been recognized a couple of times, however, it is not ok for the fraud signature to have been mis-recognized. Apparently, in this case, the 0.9777 Precision score (false positive rate) that we cared most about is not enough, we would like to achieve 0.9999 even more in order to manage the bank risk.

After some literature review, we found a paper called “An effective digit recognition model using enhanced convolutional neural network based chaotic grey wolf optimization”. It gives us a new improvement method named Orthogonal Learning Chaotic Grey Wolf Optimization (CNN-OLCGWO) that can be our further research direction.

Reference

[1] Valueva, M.V.; Nagornov, N.N.; Lyakhov, P.A.; Valuev, G.V.; Chervyakov, N.I. (2020). "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation". Mathematics and Computers in Simulation. Elsevier BV. 177: 232–243. doi:10.1016/j.matcom.2020.04.031.

[2] Durjoy Sen Maitra; Ujjwal Bhattacharya; S.K. Parui, "CNN based common approach to handwritten character recognition of multiple scripts", in Document Analysis and Recognition (ICDAR), 2015 13th International Conference on, vol., no., pp.1021–1025, 23–26 Aug. 2015

[3] K. Muhammad, J. Ahmad, Z. Lv, P. Bellavista, P. Yang and S. W. Baik, "Efficient Deep CNN-Based Fire Detection and Localization in Video Surveillance Applications," in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 49, no. 7, pp. 1419-1434, July 2019, doi: 10.1109/TSMC.2018.2830099.

[4] Vadim Smolyakov. (2018) “Neural Network Optimization Algorithms” https://towardsdatascience.com/neural-network-optimization-algorithms-1a44c282f61d

[5] Sanket Doshi. (2019) “Various Optimization Algorithms For Training Neural Network” https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

[6] https://en.wikipedia.org/wiki/MNIST_database

[7] https://en.wikipedia.org/wiki/Stochastic_gradient_descent

[8] https://github.com/vsmolyakov/experiments_with_python/blob/master/chp03/tensorflow_optimizers.ipynb

[9] https://en.wikipedia.org/wiki/Stochastic_gradient_descent#Momentum

[10] Kurbiel, Thomas, and Shahrzad Khaleghian. "Training of deep neural networks based on distance measures using RMSProp." arXiv preprint arXiv:1708.01911 (2017).

[11] https://en.wikipedia.org/wiki/Stochastic_gradient_descent#Adam

[12] S M Shamim. (2018). Handwritten Digit Recognition Using Machine Learning Algorithms. Global Journal of Computer Science and Technology, 18(D1), 17–23. Retrieved from https://computerresearch.org/index.php/computer/article/view/1685

[13] Y. Li, "Research and Application of Deep Learning in Image Recognition," 2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA), Shenyang, China, 2022, pp. 994-999, doi: 10.1109/ICPECA53709.2022.9718847.