ADAPID: AN ADAPTIVE PID OPTIMIZER FOR TRAINING DEEP NEURAL NETWORKS

Boxi Weng[1,2]*, Jian Sun*[1,2]*, Alireza Sadeghi*[3]*, and Gang Wang*[1,2]

[1]School of Automation, Beijing Institute of Technology, Beijing 100081, China
[2]Beijing Institute of Technology Chongqing Innovation Center, Chongqing 401120, China
[3]Dept. of Electrical and Computer Engineering, University of Minnesota, Mpls, MN 55455
E-mail: {wengboxi, sunjian, gangwang}@bit.edu.cn, sadeghi@umn.edu

ABSTRACT

Deep neural networks (DNNs) have well-documented merits in learning nonlinear functions in high-dimensional spaces. Stochastic gradient descent (SGD)-type optimization algorithms are the 'workhorse' for training DNNs. Nonetheless, such algorithms often suffer from slow convergence, sizable fluctuations, and abundant local solutions, to name a few. In this context, the present paper draws ideas from adaptive control of dynamical systems, and develops an *adaptive proportional-integral-derivative* (AdaPID) solver for fast, stable, and effective training of DNNs. AdaPID relies on second-order moment estimates of gradients to adaptively adjust the PID coefficients. Numerical tests corroborate the merits of AdaPID on several tasks such as image generation using generative adversarial networks (GANs) and image classification using convolutional neural networks (CNNs) as well as long-short term memories (LSTMs).

Index Terms— Deep neural network, PID control, adaptive control, stochastic optimization, adaptive learning rate

1. INTRODUCTION

Nowadays, deep learning (DL) is arguably one of the most widely used artificial intelligence technologies. Compared to traditional machine learning methods, DL has the ability to automatically extract features from complex data, tailored for down-stream tasks (e.g., classification, clustering, anomaly detection, and reinforcement learning). Representative applications of DL include natural language processing [1], computer vision [2], and unmanned system [3, 4]. In practice, the structure of DNNs along with the optimization method are the main factors influencing the performance [5]. However, finding the 'best' set of weight parameters for a given DNN model is challenging [6, 7].

The work was supported in part by the National Key RD Program of China under Grant 2021YFB1714800, in part by the National Natural Science Foundation of China under Grants 62173034, 61925303, 62088101, U20B2073, 61720106011, in part by the CAAI-Huawei MindSpore Open Fund, and in part by the Chongqing Natural Science Foundation under Grant 2021ZX4100027.

To successfully train DNNs, a very large volume of data is often needed, which makes computing the full gradient of DNN loss with respect to the weight parameters practically inefficient or non-affordable. Fortunately, stochastic gradient descent (SGD) method enables effective minimization of the training loss while preventing costly full gradient computation at every iteration. The SGD algorithm no longer uses the total gradient from all samples in the parameter updating process; but instead, it relies on gradient estimates obtained based on only a few randomly sampled data. Such a simple, yet practically powerful idea can significantly reduce the computational complexity of the training process. However, the estimated gradients often result in oscillations in the convergence of SGD-type methods in training DNNs, which is also known as overshooting in the control field.

Proportional-integral-derivative (PID) control is the most widely used control method in practice. The main idea of PID control is judiciously taking advantage of the current, past, and future error information between the system state as well as the expected state and forming a negative feedback control signal. To alleviate the overshooting problem of stochastic gradient-based DNN optimization, the work of [8] proposed a PID approach which exploits the difference in gradients in the training process. And the work of [9], on the other hand, introduced a PID optimizer with distinct P and I terms of gradients. Stemmed from these PID optimizers, we develop a novel adaptive learning process by considering the past, current, and future information of the gradients. Our proposed adaptive PID (AdaPID) method essentially relies on gradients' second-order moment estimates and gradient differences to update the DNN weights.

2. RELATED WORKS

A multitude of SGD-type solvers have been proposed for fast optimization of DNNs, such as SGD with momentum (SGD-M) [10], Nesterov's momentum [11], and those with adaptive learning rates such as AdaGrad [12], RMSProp [13], AdaDelta [14], Adam [15], and variants of Adam [16, 17, 18, 19].

Although the principle behind SGD is simple, it suffers from slow convergence, large oscillations during the training process, and convergence to local optimum [20]. SGD-M suppresses such oscillations and accelerates convergence by adding an extra momentum term to the parameter update [10]. Nesterov's momentum is a variation of SGD-M that uses the gradient of the projection position rather than the actual position to calculate the momentum [11].

AdaGrad adaptively adjusts the learning rate by accumulating the gradient squares as a correction factor in the SGD update [12], while RMSProp adds an attenuation factor when accumulating the gradients, and further reduces the oscillation by using moving weighted averages [13]. AdaDelta, on the other hand, only accumulates gradients in a given window, and thus can avoid fast learning rate attenuation and gradient vanishing [14]. A very popular solver in the field of DL is Adam, which generates an adaptive learning rate using the first- and second-order moment estimates of gradients [15]. Adam essentially combines the advantages of AdaGrad and RMSProp.

Recently, relationships between PID control and optimization algorithms have been studied; see e.g., [21] and references therein. Different PID optimizers have also been proposed in e.g., [8] and [9], where derivative (D) and proportional (P) terms are leveraged to alleviate the overshooting issue. Adaptive learning rate combined with PID optimizer however, has not yet been investigated, which is the topic of this paper.

3. ADAPTIVE LEARNING RATE PID OPTIMIZER

Comparing network optimization and control processes, we highlight some similarities [22]. If the gradient in optimization is associated with the error in control, the gradient-based parameter update can be analogously viewed as the adjustment of a dynamical system [8].

A PID controller continuously computes an error value $e(t)$ as the difference between a reference signal $r(t)$ and a measured system output $y(t)$, and employs a correction term based on proportional, integral, and derivative of the error [23]. A PID controller can be described as follows

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{d}{dt} e(t) \tag{1}$$

where $u(t)$ is the correction, K_p, K_i, and K_d are the gain coefficients of the P, I, and D terms of $e(t)$, respectively.

In DNN training process, from iteration t to $t+1$, the parameter update rule based on SGD is given by

$$\theta_{t+1} - \theta_t = -r\partial L_t/\partial\theta_t \tag{2}$$

where θ is the model parameter, r is the learning rate, and L is the loss function. Comparing (2) with (1), we can see that SGD only uses the proportional term of the gradients. Based on the similarity revealed above, if the ordinary gradient descent is regarded as a PID controller, then SGD uses only the current gradient information.

Being the most widely used control technique and with outstanding performance, PID control has the core advantage of incorporating both the proportional, integral, and derivative information of the errors in the control process, which represent the current, past, and future (predicted) information respectively. Therefore, by analogy, the use of proportional, integral, and derivative values of gradients in the optimization process may yield excellent results. There is an example of the parameter update rule using PID optimizer summarized in equation (3) below, which has been shown effective in certain cases [8]

$$\begin{cases} V_{t+1} = \alpha V_t - r\partial L_t/\partial\theta_t \\ D_{t+1} = \alpha D_t + (1-\alpha)(\partial L_t/\partial\theta_t - \partial L_{t-1}/\partial\theta_{t-1}) \\ \theta_{t+1} = \theta_t + V_{t+1} + K_d D_{t+1} \end{cases} \tag{3}$$

where D records the accumulation of gradient differences, $0 \leq \alpha < 1$ is a weighting factor, and K_d is the influence factor of D items.

In the design of optimization algorithms based on stochastic gradients, the two most pronounced directions for performance improvement are performing gradient correction and adaptive learning rate, as have been pursued in existing works reviewed in Section 2. Motivated by this observation, we put forth an adaptive learning rate that uses gradient second-moment estimates and gradient differences, which are to be processed by exponentially weighted moving average and bias correction steps. Concretely, the exponentially weighted moving average is given by

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2 \tag{4}$$

$$d_t = \beta d_{t-1} + (1-\beta)(g_t - g_{t-1})^2 \tag{5}$$

where v_t, d_t are biased second moment estimate of gradients and their difference, β is the adjustable weight parameter, and g_t is the gradient of time t. Through exponential weighted averaging, the weights of historical gradients or gradient difference decay exponentially in parameter updating. This effectively avoids the problem of 'learning fatigue' in the subsequent iterations, and also effectively alleviates the noise impact due to mini-batch samples used in stochastic optimization. This kind of algorithmic design is also shared by e.g., RMSProp, AdaDelta, and Adam. The bias correction step is carried out as follows

$$\hat{v}_t = \frac{v_t}{1-\beta^t}, \quad \hat{d}_t = \frac{d_t}{1-\beta^t} \tag{6}$$

where $\hat{v}_t, \hat{d}_t$ are bias-corrected second moment estimate of gradients and their difference. Through this bias correction, the resultant algorithm obtains more accurate moment estimates in early stages of training, and therefore accelerates the convergence rate of the algorithm.

Algorithm 1 *AdaPID*. Default settings are $\alpha = 0.9, \beta = 0.99$. Constant $\sigma = 10^{-7}$.

Require: Learning rate r; momentum factor α; exponential decay rates for moment estimates β; gain coefficient of cumulative gradient K_i; gain coefficient of cumulative gradient difference K_d; objective function $f(\theta)$; initial parameter vector θ_0.
  $v_0 \leftarrow 0, \quad \mathrm{d}_0 \leftarrow 0$
  $I_0 \leftarrow 0, \quad \mathrm{D}_0 \leftarrow 0$
  $t \leftarrow 0$
While θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \bigtriangledown_\theta f_t(\theta_{t-1})$
  $I_t \leftarrow \alpha I_{t-1} + g_t$
  $D_t \leftarrow \alpha D_{t-1} + (1-\alpha)(g_t - g_{t-1})$
  $v_t \leftarrow \beta v_{t-1} + (1-\beta) g_t^2$
  $d_t \leftarrow \beta d_{t-1} + (1-\beta)(g_t - g_{t-1})^2$
  $\hat{v}_t \leftarrow v_t / (1-\beta^t)$
  $\hat{d}_t \leftarrow d_t / (1-\beta^t)$
  $\theta_t \leftarrow \theta_{t-1} - r[K_i I_t / (\sqrt{\hat{v}_t} + \sigma) + K_d D_t / (\sqrt{\hat{d}_t} + \sigma)]$
end while
return θ_t

The proposed AdaPID method updates network parameter θ from iteration t to $t+1$ using both P, I, and D terms of gradients with an adaptive learning rate as follows

$$\begin{cases} I_t = \alpha I_{t-1} + g_t \\ D_t = \alpha D_{t-1} + (1-\alpha)(g_t - g_{t-1}) \\ v_t = \beta v_{t-1} + (1-\beta) g_t^2 \\ d_t = \beta d_{t-1} + (1-\beta)(g_t - g_{t-1})^2 \\ \hat{v}_t = \dfrac{v_t}{1-\beta^t}, \quad \hat{d}_t = \dfrac{d_t}{1-\beta^t} \\ \theta_t = \theta_{t-1} - r\left[\dfrac{K_i I_t}{\sqrt{\hat{v}_t} + \sigma} + \dfrac{K_d D_t}{\sqrt{\hat{d}_t} + \sigma}\right] \end{cases} \tag{7}$$

where $\sigma > 0$ is a given small constant. Our adaptive PID optimizer for training DNNs is tabulated in Alg. 1.

4. EXPERIMENT

We compared the proposed AdaPID optimizer with competing alternatives including Adam, PID, and SGD-M by training CNNs, GANs, and RNNs on four commonly used datasets. Specifically, we first trained a deep convolutional (DC) GAN and an LSTM model on MNIST dataset. We then trained a CNN on CIFAR10 dataset to show that our AdaPID optimizer achieves faster training speed and higher accuracy. We also used LSTM on Flights dataset to verify that the AdaPID optimizer is equally valid for time-series data. Based on the PetImage dataset, we carried out a series of experiments. The results indicate that our AdaPID optimizer also performs well on a more complex modern network. Hyperparameters of optimizers and initial learning rates vary across tests. For each type of test, we have manually tuned the hyperparameters and initial learning rates of the four optimizers through a large number of experiments.

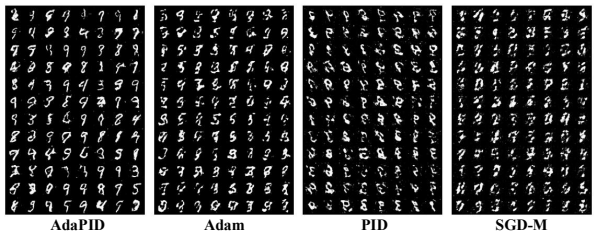

Fig. 1. Experimental results of GAN model on MNIST dataset.

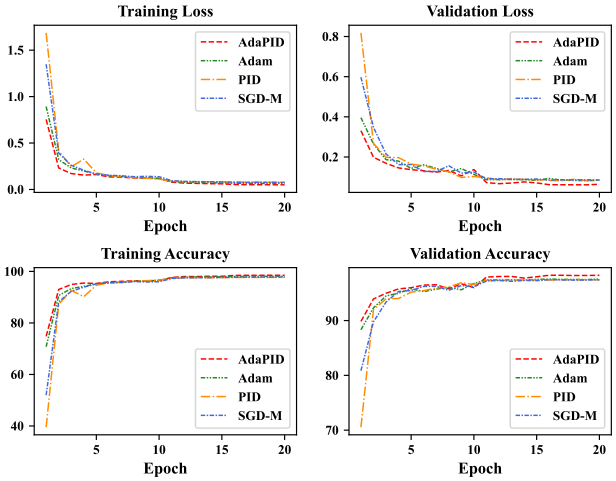

Fig. 2. Experimental results of LSTM model on MNIST dataset.

4.1. Results on MNIST dataset

MNIST [24] is an image dataset of handwritten digits from 0 to 9. The dataset contains $70,000$ samples, each of which is a gray image of 28×28 pixels. In this experiment, we first used DCGANs to test our proposed AdaPID optimizer. For AdaPID, we set $K_i = 5.0$, $K_d = 15.0$; and, for PID, $K_i = 3.0$, $K_d = 100.0$. The learning rates of optimizers were 0.00032, 0.0003, 0.001, and 0.003, respectively. Four optimizers can finally generate similar realistic images, so we compared their results at epoch 2 shown in Fig. 1. It can be seen that the AdaPID optimizer enables GAN to generate realistic images faster than alternative methods.

Next, we called for an LSTM with 2 hidden layers and a linear classifier to produce a prediction of handwritten numbers. For AdaPID, we set $K_i = 0.5$ and $K_d = 5.0$; and, for PID, $K_i = 0.8$ and $K_d = 5.0$. The initial learning rates were 0.016, 0.018, 0.15, and 0.3, respectively. The learning rate was reduced by half at 50% and 75% of training epochs. The

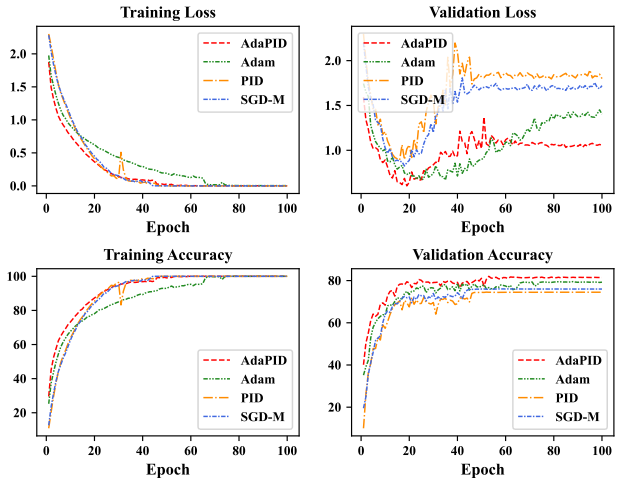

Fig. 3. Experimental results of CNN model on CIFAR10 dataset.

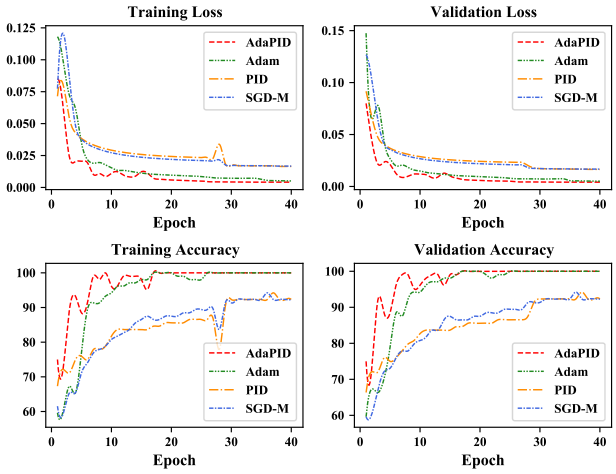

Fig. 4. Experimental results of LSTM model on Flights dataset.

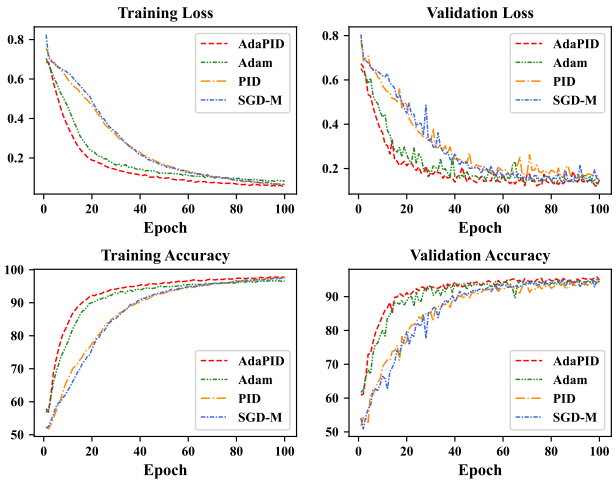

Fig. 5. Experimental results of CNNs model on PetImage dataset.

training batch size is 128 for 20 epochs. The results are shown in Fig. 2, demonstrating that our proposed AdaPID optimizer can reach a smaller error at a faster convergence rate.

4.2. Result on CIFAR10 dataset

The CIFAR10 [25] dataset contains $60,000$ images from 10 categories. Each sample in this dataset is an RGB color image, the shape of which is 32×32. The CNN used consists of five convolutional layers (with ReLU activation function and max pooling), followed by a fully connected layer. We set $K_i = 5.0$ and $K_d = 1.0$ for AdaPID, and $K_i = 1.2$ and $K_d = 5.0$ for PID. The batch size is 256. The initial learning rate for the optimizers were 0.0003, 0.0025, 0.02, and 0.03, respectively. And the learning rate was reduced by half when the training loss is no longer reduced. The results are shown in Fig. 3. Clearly, our proposed AdaPID optimizer achieves faster convergence and higher precision.

4.3. Result on Flights Dataset

Flights dataset [26] is a flight information dataset provided by Seaborn, which counted the number of boarding passengers per month during 1949–1960. We selected 12 samples as a sequence and the 13th one as the label for the sequence. In this manner, sequences were constructed from each sample. We employed an LSTM that has 2 hidden layers and a linear classifier to predict the number of future passengers. For AdaPID, we set $K_i = 0.5$, $K_d = 9.0$; and, for PID, $K_i = 5.0$, $K_d = 100.0$. The initial learning rates were 0.0009, 0.0012, 0.005, and 0.015, respectively. Depicted in Fig 4, AdaPID achieves competitive performance on the LSTM model.

4.4. Result on PetImage dataset

The PetImage dataset [27] is a contest dataset that Kaggle exposes. It contains $25,000$ training images and $12,500$ testing images. Each image sample is an RGB color image, but the size of the image is uncertain. We examined our AdaPID optimizer on the PetImage dataset using ResNet50 network [28]. We set $K_i = 10.0$ and $K_d = 1.0$ for AdaPID, and $K_i = 12.0$ and $K_d = 20.0$ for PID. The initial learning rates were set as 5×10^{-5}, 1.6×10^{-4}, 7×10^{-5}, and 10^{-3}, respectively. The results are shown in Fig. 5, among all AdaPID exhibiting the best performance. It indicates that our proposed AdaPID optimizer can also effectively train modern DNN models.

5. CONCLUSIONS

This paper presented a new method termed AdaPID for stochastic gradient-based optimization of deep neural networks. The proposed AdaPID optimizer permeates the idea of adaptive PID control into stochastic optimization, by means of adaptively leveraging the past, present, and predicted future information of the gradient throughout the training process. Numerical experiments have been provided to demonstrate its merits in training DNNs for varying tasks.

6. REFERENCES

[1] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1915–1929, 2012.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, 2012.

[3] J. Chen, J. Sun, and G. Wang, "From unmanned systems to autonomous intelligent systems," *Eng.*, 2021, DOI: https://doi.org/10.1016/j.eng.2021.10.007.

[4] L. Zhang, G. Wang, and G. B. Giannakis, "Real-time power system state estimation and forecasting via deep unrolled neural networks," *IEEE Trans. Signal Process.*, vol. 67, no. 15, pp. 4069–4077, 2019.

[5] S. S. Liew, M. Khalil-Hani, and R. Bakhteri, "An optimized second order stochastic learning algorithm for neural network training," *Neurocomput.*, vol. 186, pp. 74–89, 2016.

[6] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *Int. Conf. Mach. Learn.*, 2015, pp. 1613–1622.

[7] G. Wang, G. B. Giannakis, and J. Chen, "Learning ReLU networks on linearly separable data: Algorithm, optimality, and generalization," *IEEE Trans. Signal Process.*, vol. 67, no. 9, pp. 2357–2370, 2019.

[8] W. An, H. Wang, Q. Sun, J. Xu, Q. Dai, and L. Zhang, "A PID controller approach for stochastic optimization of deep networks," in *IEEE Conf. on Comput. Vision and Pattern Recogn.*, 2018, pp. 8522–8531.

[9] L. Shi, Y. Zhang, W. Wang, J. Cheng, and H. Lu, "Rethinking the PID optimizer for stochastic optimization of deep networks," in *IEEE Int. Conf. on Multimedia and Expo*, 2020, pp. 1–6.

[10] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Netw.*, vol. 12, no. 1, pp. 145–151, 1999.

[11] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$," in *Sov. Math. Dokl*, vol. 27.

[12] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. 7, 2011.

[13] T. Tieleman, G. Hinton *et al.*, "Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.

[14] M. D. Zeiler, "AdaDelta: An adaptive learning rate method," *arXiv:1212.5701*, 2012.

[15] D. P. Kingma and J. Ba, "ADAM: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

[16] T. Dozat, "Incorporating nesterov momentum into Adam," 2016.

[17] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and beyond," *arXiv:1904.09237*, 2019.

[18] N. S. Keskar and R. Socher, "Improving generalization performance by switching from Adam to SGD," *arXiv:1712.07628*, 2017.

[19] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *arXiv:1908.03265*, 2019.

[20] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," *arXiv:1406.2572*, 2014.

[21] L. Lessard, B. Recht, and A. Packard, "Analysis and design of optimization algorithms via integral quadratic constraints," *SIAM J. Opt.*, vol. 26, no. 1, pp. 57–95, 2016.

[22] W. Wu, X. Jing, W. Du, and G. Chen, "Learning dynamics of gradient descent optimization in deep neural networks," *Sci. China Inf. Sci.*, vol. 64, no. 5, pp. 1–15, 2021.

[23] T. Hagglund and K. J. Astrom, "PID controllers: Theory, design, and tuning," *ISA-The Instrumentation, Systems, and Automation Society*, 1995.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[25] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[26] https://github.com/mwaskom/seaborn-data.

[27] https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conf. on Comput. Vision and Pattern Recogn.*, 2016, pp. 770–778.