2018 24th International Conference on Pattern Recognition (ICPR)
Beijing, China, August 20-24, 2018

Dynamic Learning Rate for Neural Networks: A Fixed-Time Stability Approach

Rodrigo Aldana, Leobardo Campos-Macías, Julio Zamora,
David Gomez-Gutierrez and Adan Cruz
Intel labs
Zapopan, Jalisco, Mexico
Telephone:+52 33 1645 0000

***Abstract*—Neural Networks (NN) have become important tools that have demonstrated their value solving complex problems regarding pattern recognition, natural language processing, automatic speech recognition, among others. Recently, the number of applications that require running the training process at the front-end in an online manner have increased dramatically. Unfortunately, in state-of-the-art (SoA) methods, this training process is an unbounded function of the initial conditions. Thus, there is no insight on the number of epochs required, making the online training a difficult problem. Speeding up the training process plays a key role in machine learning. In this work, an algorithm for dynamic learning rate is proposed based on recent results from fixed-time stability of continuous-time nonlinear systems, which ensures a convergence time bound to the equilibrium point independently of the initial conditions. We show experimentally that our discrete-time implementation presents promising results, proving that the number of epochs required for the training remains bounded, independently of the initial weights. This constitutes an important feature toward learning systems with real-time constraints. The efficiency of the method proposed is illustrated under different scenarios, including the public database MNIST, which shows that out algorithm outperforms SoA methods in terms of the number of epoch required for the training.**

I. Introduction

In recent years, deep learning (DL) has become an outstanding tool to resolve high-complexity pattern recognition problems, playing a key role for representing and extracting complex data in visual understanding problems [1] [2], and at the same time, the need of multimedia data have increased considerably [3], because such large databases are required to train new updated NN models. The first approaches of machine learning (ML) were based in *Back Propagation (BP)* reducing the error using gradient descent to solve a series of difficult problems, e.g. speech recognition, natural language processing, pattern recognition, etc. [4] [5].

Thus, the development of methods capable of solving the learning problem of the NN, defined in the next sections, has increased with the advance of DL methodologies; for this, several learning approaches have been proposed [6]. The main idea behind an optimization algorithm to solve this problem is to find a local minimum in a short period of time whose cost is sufficiently low for the application [7], instead of achieving the global minimum.

Either to find a local minima, or to continue to find a global minimum, the learning rate plays a critical role to decrease the convergence time on deep networks and adapting it has shown promising results, even though it's still an open problem [8]. Reported works have demonstrated that the main difficulty of the learning problem is the proliferation of saddle points, not local minima [9]. Such saddle points are usually surrounded by high error plateaus, deriving in a slow down of the learning and giving the impression of the existence of a local minimum. Taking this into account, gradient normalization has been explored, showing its effectiveness [10] [11] [12], in [13] authors propose a first order method over performing the Adam unfortunately it is not bounded, iterations depends of initialization.

Our method proposes a learning rate adaptation that demonstrates experimentally the capability of speeding up the training process when compared to the other methods. Through a learning problem of the multi-layered networks and a learning problem of the convolutional neural networks, the effectiveness of our proposed algorithm was verified. In addition, we compute the maximum number of iterations required to achieve the local minima.

II. Learning Algorithms

A. Problem formulation

In general, the training process is based on running an optimization algorithm to fit the training data to a NN model using the weights $w = (w_1, .., w_n)$ to the optimal of the NN. Most of the algorithms found in the scientific literature have the form:

$$w_{m+1} = w_m - H(w_m, w_{m-1}, \ldots, m),$$

where m is the iteration index, and $H(w_m, w_{m-1}, \ldots, m)$ is a vector that may be a function of the previous history of the weights and the current time step. This function is chosen so that the sequence of weights converges to a local minimum of the error function between the training data and the model. Currently, there are no methods that guarantee a maximum number of epochs required for the training process to finish.

B. State of the art

State of the art solutions focus on speeding up the learning process such as Adagrad, Adam or RMSprop.

978-1-5386-3788-3/18/$31.00 ©2018 IEEE

1) Adagrad: This is an algorithm for gradient-based optimization that adapts the learning rate $\hbar(m)$ to the weights w_{ij}, performing larger updates for infrequent and smaller updates for frequent weights[10]. For this reason, it is well-suited for dealing with sparse data. The approach reported in [14], the learning rate $\eta_{ij}(t)$ for each parameter $w_{i,j}$ is decayed by the square root of the sum of the gradients:

$$\eta_{ij}(t) = \frac{\eta(0)}{\sqrt{\sum_t g_{ij}(t)^2}}$$

2) RMSprop: This approach divides the learning rate $\hbar(m)$ by an exponentially decaying moving average of squared gradients[11]:

$$\theta_{t+1} = \theta_t - g(t)\frac{\eta}{\sqrt{E\left[g(t)^2\right] + \varepsilon}},$$

where ε is a hyper-parameter and,

$$E\left[g(t)^2\right] = \gamma E\left[g(t-1)^2\right] + (1-\gamma)(\nabla\theta)^2,$$

where γ is a hyper-parameter of the RMSprop algorithm.

3) Adam: Adaptive Moment Estimation is a mini-batch type method that computes adaptive learning rates $\hbar(m)$ for each weight. In addition to storing an exponentially decaying average of past squared gradients like RMSprop, Adam also keeps an exponentially decaying average of past gradients, similar to momentum [12]. Adam method uses the parameter update rule:

$$\theta_{t+1} = \theta_t - \hat{m}_t\frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon}$$

where ε is a hyper-parameter of the Adam method and

$$\hat{m}_t = \frac{m_t}{(1-\beta_1^t)}$$

$$\hat{v}_t = \frac{v_t}{(1-\beta_2^t)}$$

where β_1 and β_2 are hyper-parameters and m and v are the exponential moving average of $\nabla\theta$ and $(\nabla\theta)^2$, respectively.

4) Eve: This approach is an improvement over the Adam algorithm , where the most important difference is the feedback term that captures the relative change of the objective value[13], defined as:

$$\theta_{t+1} = \theta_t - \hat{m}_t\frac{\eta}{d_t\sqrt{\hat{v}_t} + \varepsilon}$$

here d is a nonlinear function depending of the objective values at time t and $t-1$, it means $d_t = \beta d_{t-1} + (1-\beta)r_t$ and r_t defined as

$$r_t = \begin{cases} \frac{f_{t-2}-f_{t-1}}{f_{t-1}} & f_{t-2} \geq f_{t-1} \\ \frac{f_{t-1}-f_{t-2}}{f_{t-2}} & f_{t-1} > f_{t-2} \end{cases}$$

where β is a decay rate in $[0, 1)$.

In general, all the described methods above implement a decay of the learning rate in order to obtain a better accuracy in the solution.

Inspecting the results of these algorithms, we summarize several disadvantages of the optimization algorithms to solve the learning problem of the NN.

- **Epochs required**: the uncertainty of not knowing how many epochs are needed for the training process to be completed in most of the methods, thus they are not suitable for problems with hard real time constraints.
- **Gradient accumulation**: the sum of the gradients accumulation over all previous iterations causing to shrink the learning rate until it is infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge.
- **Decay rates**: the use of a decay technique eventually becomes the learning rate very small being no longer able to acquire additional knowledge.

III. Lyapunov stability of nonlinear systems

A dynamical system can be expressed in a state space representation where the vector of states $x(t) = (x_1(t), \ldots, x_n(t))$ are the time dependent variables of interest, and the dynamics are written as a system of differential equations (often nonlinear):

$$\dot{x} = f(x)$$

where $\dot{x} = dx/dt$ and $f : \mathbb{R}^n \to \mathbb{R}^n$ is a vector field. A critical point x^* is one that satisfies $f(x^*) = 0$. The system is called to be stable in all $\mathbb{R}^n$ if, for any initial condition x_0, the system evolves and as $t \to \infty$ then $x \to x^*$ for some critical point x^*. Lyapunov stability analysis states that if there exists a continuous radially unbounded function [15], [16]

$$V : \mathbb{R}^n \to \mathbb{R}_+ \cup \{0\}$$

Such that $V(x^*) = 0$, (basically $V(x)$ is a function of the state which is always positive and only zero at the critical points), and that satisfies:

$$\dot{V} < 0$$

Then, the system is stable towards some x^*. Using the chain rule:

$$\dot{V} = \nabla V \cdot \dot{x} = \nabla V \cdot f$$

where,

$$\nabla V = \left(\frac{\partial V}{\partial x_1}, \ldots, \frac{\partial V}{\partial x_n}\right)$$

is the gradient and $\cdot$ is the dot product of vectors. Thinking of V as the energy of the system, $\dot{V} < 0$ means that if the energy always decreases, the system will reach ultimately a steady state (critical point) 2

Moreover, if:

$$\dot{V} < -(\alpha V^p + \beta V^q)^k$$

For $\alpha, \beta, p, q, k > 0$ such that $pk < 1$ and $qk > 1$, then x will reach some x^* in less that T_{max} where:

$$T_{max} = \frac{1}{\alpha^k(1-pk)} + \frac{1}{\beta^k(qk-1)}$$

This type of convergence is called a fixed time stability [15].

A. Lyapunov analysis for NN training (neuron weights convergence)

Suppose you have a cost function $V(w)$ which is the overall training error of a NN for a batch of training data. The training goal is to minimize $V(w)$ which depends on the neuron weights $w = (w_1, \ldots, w_n)$. A critical point w^* is one that satisfies:

$$\nabla V|_{w=w^*} = 0$$

In other words, one point in which the gradient $\nabla V = (\partial V/\partial w_1, \ldots, \partial V/\partial w_n)$ is zero. There could be many points satisfying this property, thus the existence of many local minima. An optimization algorithm is a rule which takes an initial guess w_0 and by executing the rule $w_{m+1} = g(w_m, w_{m-1}, \ldots)$ for some function g of the past steps, and reaches a critical point w^* as $m \to \infty$.

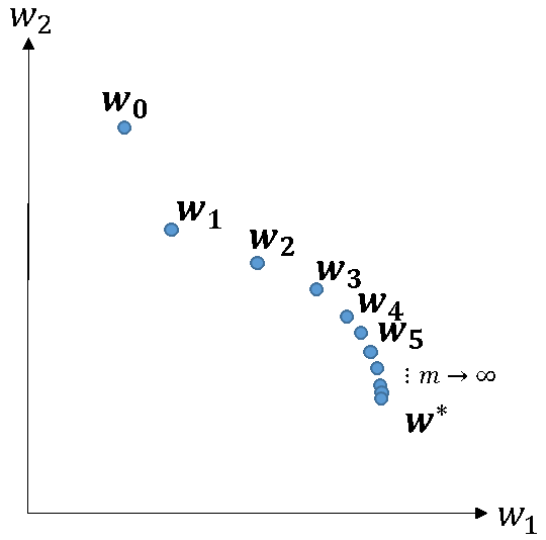


Fig. 1. Weights evolution through time

The idea is to approximate the evolution (steps) of the optimization algorithm by a continuous evolution (changing the discrete step variable m to a continuous variable t) in order to apply the Lyapunov stability analysis.

Then, the evolution of the weights (thus the optimization algorithm itself) will be represented as:

$$\dot{w} = f(w)$$

For the time dependent function $w(t)$, and then its discrete implementation will be recovered by doing $t_m = hm$ for a small increment h.

By taking our cost function as a Lyapunov function since it behaves like one at least near a critical point, the algorithm is designed by choosing f such that

$$\dot{V} = \nabla V \cdot f < 0.$$

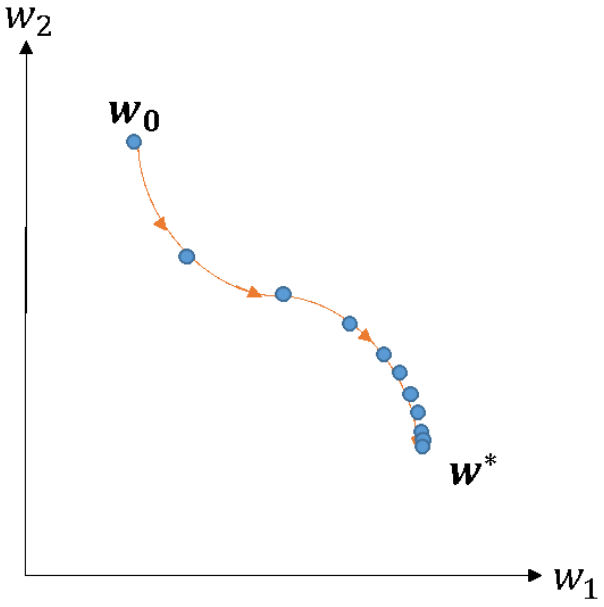


Fig. 2. Weights evolution throw the time, where w_0 is the initial state and w^* represents the local minimal.

If we choose $f = -\nabla V$, then $\dot{V} = -\|\nabla V\|^2 < 0$ so the system is stable to some point satisfying:

$$f(w^*) = \nabla V|_{(w=w^*)} = 0$$

Thus, stable points of the dynamical system are critical points of the cost function V (assuming that V does not have any maximum those critical points can be called local minima). In order to discretize the algorithm, the following approximation (Eulers derivative approximation) is taken

$$\dot{w} \cong \frac{(w_{m+1} - w_m)}{h}$$

where $w_m = w(t_m)$, then,

$$\dot{w} = f(w) = -\nabla V \to w_{m+1} = w_m - h\nabla V$$

which is the classical gradient descent algorithm, where h is the step size or often called learning rate. The novelty of our proposal is to design f in the following way:

$$f(w) = -\gamma \frac{(\alpha V^p + \beta V^q)^k}{\|\nabla V\|^2} \nabla V$$

For some $\gamma > 1$. Then is easy to show that:

$$\dot{V} = -\gamma(\alpha V^p + \beta V^q)^k < -(\alpha V^p + \beta V^q)^k$$

So the weights w will converge in a fixed time less than T_{max}. By discretizing, the final algorithm is expressed as:

$$w_{m+1} = w_m - h\gamma \frac{(\alpha V^p + \beta V^q)^k}{\|\nabla V\|^2} \nabla V$$

Defining:

$$\hbar(m) = h\gamma \frac{(\alpha V^p + \beta V^q)^k}{\|\nabla V\|^2} \quad (1)$$

The algorithm can be written as:

$$w_{m+1} = w_m - \hbar(m)\nabla V$$

which is a gradient descent algorithm with a variable learning rate given by $\hbar(m)$. Fixed time stability results allows to approximate the number of maximum iterations required for the training by:

$$M_{max} = \frac{T_{max}}{h} = \frac{1}{\alpha^k(1-pk)h} + \frac{1}{\beta^k(qk-1)h} \quad (2)$$

By carefully tuning the parameters $h, \gamma, \alpha, \beta, p, q, k$, a desired maximum number of iterations can be achieved.

IV. Experimental Results

The proposed learning rate algorithm was programmed in C++ language and implemented on a PC running Windows 10. The purpose of these experiments was to illustrate the behaviour of the proposed method in different functions, and to compare with SoA algorithms. Additionally, we present experimental results by solving a classification problem and compared with SoA algorithms, corroborating the property for establishing a limit in the iterations for the learning process.

A. Benchmark Results

In order to exemplify the performance of our algorithm, different functions were randomly selected, the three most outstanding results are presented in Fig. 3. In the first function $y = 4x^4 - 12x^2 - 5x + 16$ (Fig 3a) many different starting points were used from both sides of the global minimum and local minima. It is well known that gradient descendant methods are being caught to the local minima; in this case our algorithm was able to achieve the global minimum starting from both directions in less than 22 iteration as equation 2 estimates. The same methodology was used for the function $y = 3x^4 - 6x^2 + 3x + 7$ (Fig. 3b); even though the staring points were selected from both sides of local and global minimum, the algorithm *jumps* towards the global minimum, when the expectation was to reach the local minima, showing a clear advantage of this technique that can be presented in some of the functions. Finally, we experimented the behaviour of the algorithm in a non polynomial function, $y = 3 + \frac{10 \sin x}{x}$ (Fig. 3c), which led to similar results, finding shortcuts to reach global minimum. As a remark in this experiment, the global minimum was not always reached, despite this, a minimum was always obtained within the limit of iterations selected.

For this issue, we select the *Beale function* [17]

$$z = \left(\frac{3}{2} - x + xy\right)^2 + \left(\frac{9}{4} - x + xy^2\right)^2 + \left(\frac{21}{8} - x + xy^3\right)^2,$$

which have a single global minimum at (3,0.5) as benchmark scenario to validate against the SoA algorithm AdaGrad [10], RMSPro [11], and Adam [12]. Fig. 4 depicts the error decay with respect to the iteration number when optimizing this function. Only 52 iterations were required by our proposed algorithm over Adam which required 120 iterations. It should be noted that the maximum number of iterations estimated M_{max} was 150.

Finally, using the Beale function and starting from different coordinates (initial conditions), the global minimum was reached in less than the expected number of maximum iterations as shown in Fig. 5 and listed in table I. This results validate that our proposal helps to speed up the optimization process (see result on Fig 4), and also that the goal is reached in less than the estimated maximum number of iterations ensuring the application to real time problems.

$Init(x_0, y_0)$	Iterations
$(0.1, 0.1)$	9
$(1.0, 0.0)$	30
$(0.0, 3.0)$	78
$(0.0, -1.0)$	120
$(2.0, -1.0)$	122

TABLE I
NUMBER OF ITERATIONS REQUIRED TO REACH TO GLOBAL MINIMUM. STARTING FROM DIFFERENT INITIAL CONDITIONS IN BEALE FUNCTION.

B. Experimental Results

The benchmark scenario consists in solving the manually generated classification 2D problem shown in Fig. 6 using a multi-layer perceptron NN.

The NN topology used was defined by 2 inputs, representing the 2D scenario used for graphically represent the solution image (Fig. 7), 9 neurons in the hidden layer, and 5 output neurons to classify 5 different classes. The parameter values used in this experiment were: $\alpha = 0.3$,$\beta = 0.3$,$p = 0.2$,$q = 2$,$k = 0.7$,$\gamma = 1.001$,$h = 0.1$. These values were used to fulfill the conditions $pk < 1$, and $qk > 1$. Fig. 8 shows the error decay in relation with the iterations for this problem.

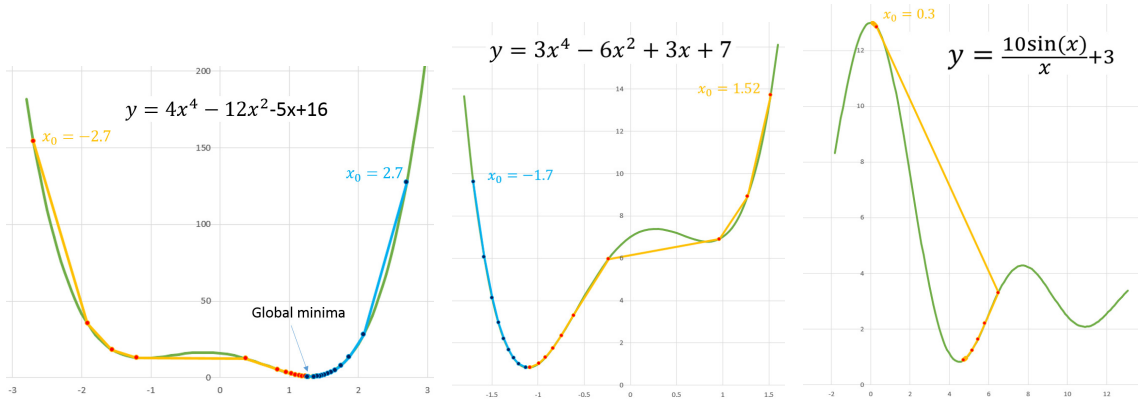
The methods used in this comparison started from the same initial weight values and they had the same network topology as well as the same training data.

The scenario shown in Fig. 6 represents a toy example. To illustrate the behavior of the dynamic learning described, we experimented with public database MNIST, and it was observed that the Dynamic learning technique outperforms traditional training methods.

As the results show in Fig. 9, this method is a good candidate for hardware implementation since it reduces training time, thus having a lower power consumption compared to other method. Furthermore, knowing the maximum number of iterations makes this method suitable for real-time applications.

V. Conclusion

The uncertainty of not knowing how many epochs are required for the training process of a NN model to be completed is one disadvantage of other methods, e.g. Adagrad, RMSprop, ADAM, EVE, thus they are not suitable for problems with hard real time constraints.These methods present certain



(a) (b) (c)

Fig. 3. A set of experiments for finding the global minimum: (a) the starting points were chosen at left $x_0 = -2.7$ and right $x_0 = 2.7$, (b) starting at $x_0 = -1.7$ and $x_0 = 1.52$ and (c) at $x_0 = -0.3$.

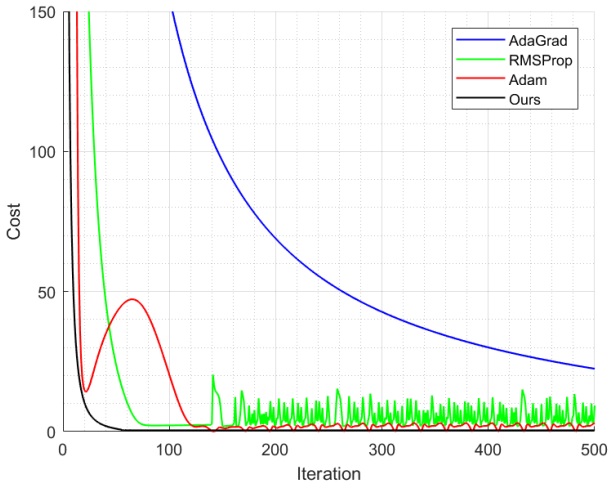


Fig. 4. Error decay using different methods for Beale function.

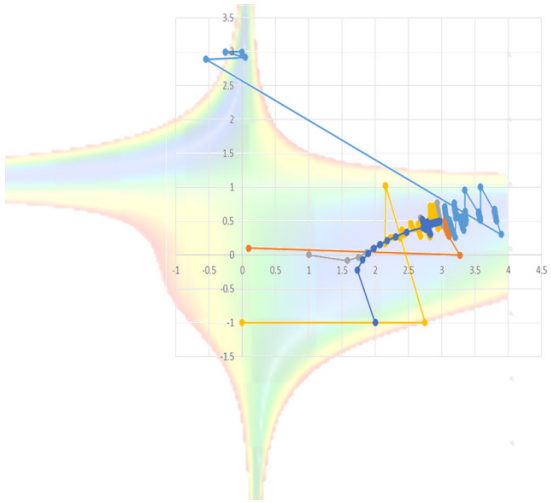


Fig. 5. Different starting points, multiple paths to converge to global minimum.

disadvantages, for instance Adagrad's main weakness is its accumulation of the squared gradients in the denominator, since every added term is positive, and the accumulated sum keeps growing during training. This turn causes the learning rate to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge. On the other hand, RMSprop and ADAM algorithms use a decay technique eventually the learning rate becomes very small being no longer able to acquire additional knowledge as well as Adagram. In order to tackle these problems, speed up the training process and having a possible mean to estimate the maximum numbers of iterations that ensures the convergence, the work presented here introduced the dynamic learning rate 1 technique. This algorithm was based on recent results from fixed-time stability of continuous-time nonlinear systems, which allows the training process to happen in maximum known number of epochs, given in 2. Additionally, this method speeds up the training process over other methods by correctly selecting the maximum number of iterations. This proposal could result suitable for real-time applications as well as for FPGA and ASIC implementation for NN integrated circuits.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105.

[2] S. Pouyanfar and S. C. Chen, "Semantic event detection using ensemble deep learning," in *2016 IEEE International Symposium on Multimedia (ISM)*, Dec 2016, pp. 203–208.

[3] X. Chen, C. Zhang, S. C. Chen, and S. Rubin, "A human-centered multiple instance learning framework for semantic video retrieval," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 39, no. 2, pp. 228–233, March 2009.

[4] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition:

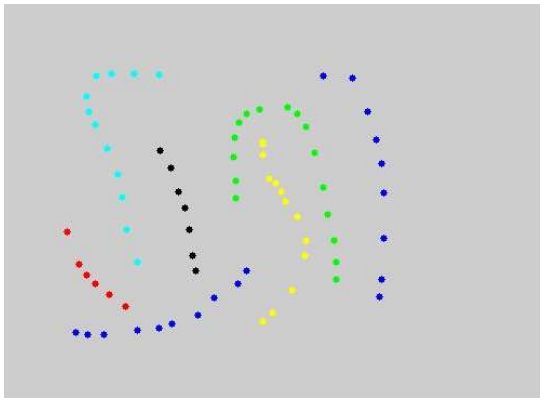


Fig. 6. Pattern used for comparing the approaches

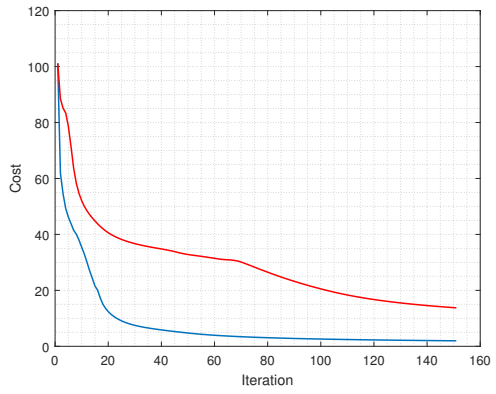


Fig. 7. Comparing the traditional training method vs Lyapunov's based dynamic learning rate (in blue).

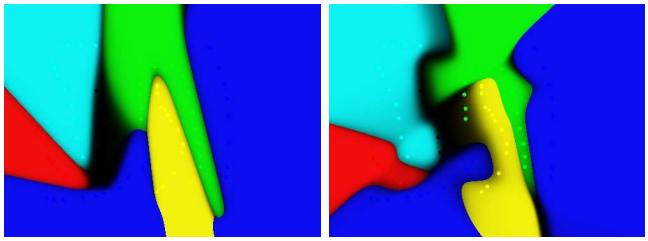


Fig. 8. The classification solution of the methods compared: Ours (Left), gradient descent (right).

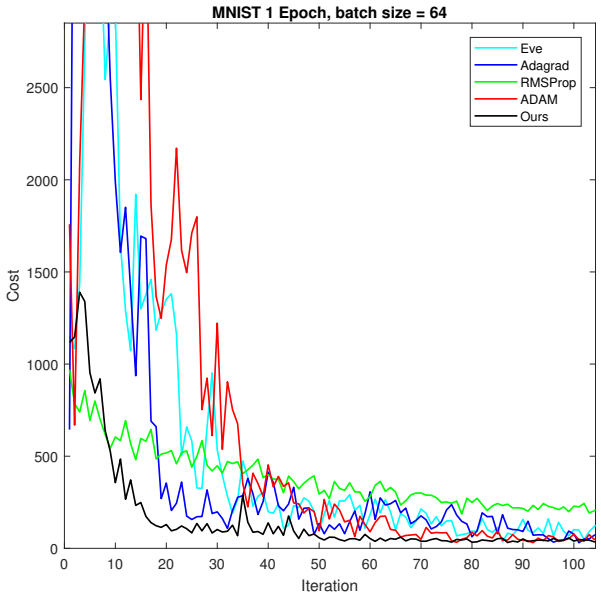


Fig. 9. Comparing the most popular techniques vs our dynamic learning rate approach for MNIST.

The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Nov 2012.

[5] Y. Kim, "Convolutional neural networks for sentence classification," in *EMNLP*, 2014.

[6] S. Pouyanfar and S. C. Chen, "Semantic concept detection using weighted discretization multiple correspondence analysis for disaster information management," in *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*, July 2016, pp. 556–564.

[7] A. Choromanska, M. Henaff, M. Mathieu, G. Ben Arous, and Y. LeCun, *The Loss Surfaces of Multilayer Networks*, 2015.

[8] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012.

[9] Y. Dauphin, R. Pascanu, Ç. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," *CoRR*, vol. abs/1406.2572, 2014.

[10] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011.

[11] N. S. G. Hinton and K. Swersky, "Rmsprop: Divide the gradient by a running average of its recent magnitude," University Lecture, 2012.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[13] J. Koushik and H. Hayashi, "Improving stochastic gradient descent with feedback," *ICLR 2017*, Nov 2017.

[14] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1223–1231.

[15] A. Polyakov, "Nonlinear feedback design for fixed-time stabilization of linear control systems," *IEEE Transactions on Automatic Control*, vol. 57, no. 8, pp. 2106–2110, Aug 2012.

[16] H. K. Khalil and J. Grizzle, *Nonlinear systems*. Prentice hall Upper Saddle River, 2002, vol. 3.

[17] P.-A. Simionescu and D. G. Beale, "New concepts in graphic visualization of objective functions," *ASME 2002 Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, October 2002.