ISSN NO: 9726-001X
Volume 11 Issue 02 April 2023

JOURNAL OF CURRENT SCIENCE

DEEP NETWORK OPTIMIZATION UTILIZING ADAPTIVE RATES

Mrs.K Vijaya Lakshmi , Mrs.G Madhavi , Mrs.G S Gowthami Kumari
Assistant Professor[1,2,3]
Department of CSE,
Viswam Engineering College (VISM) Madanapalle-517325 Chittoor District, Andhra Pradesh, India

Abstract: - The developing complexity of deep learning architectures results in weeks or maybe months of schooling time. This sluggish education is due to "vanishing gradients," in which the gradients once again used by propagation are rather massive for weights connecting deep layers (layers near the output layer) and quite small for shallow layers (near the input layer); this results in slow learning within the shallow layers. In addition, it has been confirmed that low curvature saddle factors may proliferate during particularly non-convex disorders, including deep neural networks, which dramatically slows down learning [1]. On this paper, through the presentation of an optimization method for education of deep neural networks, we strive to overcome the two above problems by using study prices that may be specific to each layer in the network and adaptive to the curvature of the feature, developing the knowledge of load at low curvature elements. This allows us to hurry up to learn in the network's shallow layers and short break out excessive-errors of low curvature saddle components. We look at our approach to large image magnificence datasets that include MNIST, CIFAR10 and Image Net, and show that our method will further boost accuracy to reduce the required time for schooling over giant algorithms.

I. INTRODUCTION

Over the past few years, deep neural networks have been exceptionally effective, achieving state-of-the-art results on a wide variety of tasks, such as picture classification [2], face recognition [3], feeling analysis [4], voice recognition [5], etc. In these articles, one can note a general trend: outcomes appear to get stronger as the volume of training data grows, coupled with a rise in the sophistication of the design of the deep network. Even with high-performance hardware, increasingly complicated deep networks can take weeks or months to train, however. Therefore, for training deep networks, there is a need for more powerful approaches. By performing a sequence of non-linear transformations, deep neural networks learn high-level features. Let the training data set A consist of n data points a1, a2, an x Mr and corresponding labels B = {bi} n i=1. Let us assume the activation role of a 3-layer network with f. Let X1 and X2 denote the weights that we are attempting to learn on - line, i.e., X1 denotes the weights between the first and second layer nodes, and X2 denotes the weights between the second layer and third layer nodes. The learning problem can be formulated as the following optimization problem for this particular example:

$$\underset{X_1, X_2}{\text{minimize}} \quad \left\| f(f(A \cdot X_1) \cdot X_2) - B \right\|_2^2 \qquad (1)$$

Any non-linear mapping may be the activation function f, which historically is a sigmoid or tan function. Recently, rectified linear (Relook) units (f (z) =max {0, z}) have become common since, for certain issues, they appear to be simple to train and deliver superior results [6]. Using iterative approaches (such as back-propagation) in the hope of converging to a good local minimum, the non-convex objective (1) is usually reduced. Most iterative schemes produce additive changes to the parameter set x (weight matrices, in our case) of the shape.

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)} \qquad (2)$$

Where Δx (k) is a suitably selected modification. Note that from traditional optimization literature, we use somewhat different notation here in that we integrate the phase size or learning rate t (k) into x (k). This is done to assist us in the subsequent parts to conveniently explain other optimization algorithms. Therefore, in the parameters, Δx (k) denotes the update and consists of a quest path and a phase size or learning rate t (k), which controls how wide a step in that direction is to be taken. The most popular updating rules are gradient descent versions, where the negative gradient g (k): gives the search direction:

$$\Delta x^{(k)} = -t^{(k)} g^{(k)} = -t^{(k)} \nabla f(x^{(k)}) \qquad (3)$$

As the scale of training data for these deep networks is typically on the range of millions or billions of data points, it is not possible to reliably measure the gradient. Instead, utilizing a single data point or a limited batch of data points, the gradient is always calculated. This is the foundation for stochastic gradient descent (SGD) [7], the most frequently used form in deep net preparation. SGD has to pick an initial learning rate manually and then create an update law for the learning rate that lowers it over time (for example, exponential decay with time). However, SGD's output is very responsive to this upgrade option, resulting in adaptive methods that change the learning rate automatically as the machine learns [8], [9]. Additional issues are added as these descent approaches are used for training deep networks. The gradients that are propagated back to the original layers become very minimal as the amount of layers in a network grows. This slows down the pace of learning in the initial layers significantly and slows down the integration of the whole network [10].
Recently, it has also been seen that the presence of local minima of high error compared to the global minimum is exponentially small in the number of dimensions for high-dimensional non-convex topics, such as deep networks. Instead, there is an exponentially significant amount of low-curvature high-error saddle points [1], [11], [12] in these issues. Gradient descent approaches, by observing the paths of negative curvature, usually travel away from saddle points. However, the moves taken become very limited due to the low curvature of small negative eigenvalues, thereby slowing down learning considerably. We suggest a process in this paper that alleviates the above-mentioned concerns. Below is the key contribution of our process, summarized

- In the network, the learning rates are unique to each substrate. In shallow layers, this requires higher learning speeds to compensate for the limited scale of gradients.
- At low curvature points, the learning rates for each layer begin to rise. This helps the method to escape easily from high-error, low-curvature saddle points, which exist in deep network abundance.
- It is applicable to most current methods of stochastic gradient optimization that use a worldwide learning scale.
- It needs relatively little additional computation over traditional methods of stochastic gradient and does not involve any additional storage of previous gradients, as in AdaGrad [9].
- We discuss some common gradient approaches in Section II, which have been effective for deep networks. We define our optimization algorithm in Section III. Finally, we equate our solution to traditional optimization algorithms on datasets such as MNIST, CIFAR10 and Image Net in Section IV.

II. RELATED WORK

Stochastic Gradient Descent (SGD) remains one of the most frequently employed large-scale machine learning approaches, mainly because of its simplicity of execution. In SGD, the parameter updates are described by equations (2) and (3), and as iterates reach a local optimum, the learning rate is diminished over time. A standard update of the learning rate is given b

$$t^{(k)} = t^{(0)}/(1+\gamma k)^p \qquad (4)$$

If hyper parameters selected by the consumer are the initial learning rate t (0), γ and p. Many improvements have been proposed to the simple gradient descent algorithm. Newton's strategy, which uses the Hessian of the objective function f(x) to calculate the phase scale, is a popular approach in the convex optimization literature:

$$\Delta x_{nt}^{(k)} = -\nabla^2 f(x^{(k)})^{-1} g^{(k)} \qquad (5)$$

Unfortunately, computing the Hessian becomes very computationally costly as the sum of parameters grows, even to moderate scale. Therefore, several changes have been suggested that either aim to optimize the usage of first-order knowledge or seek to estimate the objective function of the Hessian. We concentrate on modifications to first-order approaches in this article. For parameters for which the gradient continuously points in the same direction, the classical momentum method [13] is a strategy that raises the learning rate, thus reducing the learning rate for parameters for which the gradient varies rapidly. The update equation then retains track of previous parameter changes for an exponential decay:

$$\Delta x^{(k)} = \mu \Delta x^{(k-1)} - t g^{(k)} \qquad (6)$$

Where μ ?? [0, 1] is alluded to as the momentum coefficient and the global learning rate is t > 0. In some cases, Nesterov's Accelerated Gradient (NAG) [14], a first order process, has a better convergence rate than gradient descent. This approach forecasts the gradient for the next iteration and, depending on the expected gradient, adjusts the learning rate for the current iteration. Thus, if the gradient is higher for the next stage, the learning rate for the current iteration will be improved and it will slow down if it is smaller. Recently, [15] shows that with the change equation, this approach can be thought of as a momentum method as follows:

ISSN NO: 9726-001X
Volume 08 Issue 01 March 2020

$$\Delta x^{(k)} = \mu \Delta x^{(k-1)} - t \nabla f(x^{(k-1)} + \mu \Delta x^{(k-1)}) \quad (7)$$

This approach will achieve high levels of efficiency when used on deep networks [15] by a carefully constructed random initialization and using a special kind of slowly rising schedule for μ. Recent analysis has demonstrated that using a learning rate unique to each parameter may be a far more efficient method, instead of using a standard learning rate for all parameters. A tool which has become common is AdaGrad [9], which uses the following upgrade rule:

$$\Delta x^{(k)} = -\frac{t}{\sqrt{\sum_{i=1}^{k} (g^{(i)})^2}} g^{(k)} \quad (8)$$

The denominator of all the gradients of the previous iterations is the l2 norm. To give a parameter-specific learning rate, this escalates the global learning rate t, which is shared by all parameters. One downside of AdaGrad is that, across all previous iterations, it accumulates gradients, the amount of which tends to rise during preparation. This shrinks the learning rate on each parameter (along with weight decay) until each is infinitesimally tiny, reducing the amount of effective training iterations. AdaDelta [8] is a method which builds on AdaGrad and attempts to address some of the drawbacks described above. Using an exponentially decaying average of the squared gradients, AdaDelta accumulates the gradients in previous time measures. This keeps the denominator from being infinitesimally tiny and means that, even after a significant number of iterations, the parameters continue to be modified. It also substitutes an exponentially decaying sum of the squares of the parameter changes x over the previous iterations for the global learning rate t. When used to train deep networks, this approach has been shown to do reasonably well, and is far less susceptible to hyper-parameter selection. However, in terms of accuracy [8], it does not do as well as other approaches such as SGD and AdaGrad.

III. OUR APPROACH

"Shallow network layers appear to have much narrower gradients than deep layers owing to the "vanishing gradients" effect, often varying in order of magnitude from one layer to the next [10]. Methods either retain a global learning rate that is replicated across all parameters or use an adaptive learning rate unique to each parameter in most prior work in optimization for deep networks. The following observation is exploited by our method: parameters in the same layer have gradients of identical magnitudes and may thus effectively share a shared learning rate. It is possible to use layer-specific learning speeds to accelerate layers with lower gradients. Another value of this strategy is that our system stays computationally effective by preventing the calculation of huge numbers of parameter-specific learning speeds. Finally, as described in Section I, we also want our technique to take big steps at low curvature points in order to prevent slowing down learning at high-error low curvature saddle points. For any regular optimization process, let t (k) is the learning rate at the k-the iteration. This would be given by equation 4 in the case of SGD, while for AdaGrad it would only be the global learning rate t, as in equation 8. We suggest that t (k) be changed as follows:

$$t_l^{(k)} = t^{(k)}(1 + \log(1 + 1/(\|g_l^{(k)}\|_2))) \quad (9)$$

Here, t (k) l denotes the new learning rate for the parameters at the k-the iteration in the l-the layer and g (k) l denotes the vector of the parameter gradients at the k-the iteration in the l-the layer. Thus, we see that to calculate the learning rate for that line, we use only the gradients in the same layer. It is also necessary to remember that, from previous versions; we do not use any gradients and therefore save on storage. We see from equation 9 that the equation essentially reduces to using the usual learning rate t (k) when the gradients in a layer are very high. However, we are more inclined to be at a low curvature point where the gradients are very small. Thus, the equation raises the learning rate to ensure that the network's initial layers learn quicker and that we easily avoid high-error low-curvature saddle points. On top of SGD, we can use this layer-specific learning pace. The change in that case, using equation 3, will be:

$$\Delta x_l^{(k)} = -t_l^{(k)} g_l^{(k)} \quad (10)$$

$$= -t^{(k)}(1 + \log(1 + 1/(\|g_l^{(k)}\|_2)))g_l^{(k)} \quad (11)$$

Where (k) l denotes the change at the k-the iteration in the l-the layer parameters. Similarly, to use our updated learning speeds, we should change AdaGrad's upgrade equation (8).

$$\Delta x_l^{(k)} = -\frac{t_l^{(k)}}{\sqrt{\sum_{i=1}^{k} (g_l^{(i)})^2}} g_l^{(k)} \quad (12)$$

ISSN NO: 9726-001X
Volume 08 Issue 01 March 2020

Notice that we use a separate learning rate for each layer, which is shared by all weights in that layer, unlike AdaGrad, which uses a distinct learning rate for each parameter. In addition, AdaGrad modifies the learning rate based on the entire background of gradients observed for that weight, whereas we change the learning rate of a layer based only on gradients observed in the current iteration for all weights in a given layer. Thus, our method prohibits both the accumulation of gradient knowledge from previous iterations and the calculation of learning rates for each parameter; relative to AdaGrad, it is therefore less computational and memory intensive. On large-scale datasets such as Image Net (when extended over SGD), where AdaGrad struggles to converge to a successful solution, the suggested layer unique learning rates still perform well. Any current optimization methodology that uses a global learning rate has a layer-specific learning rate, and easily escapes saddle points, both without losing computing or memory usage, can be used for the proposed process. Using our adaptive learning rates on top of established optimization strategies almost always boosts efficiency on regular datasets, as we illustrate in Section IV. For any current optimization strategy that uses a global learning rate, the suggested approach may be used. This allows reaching a layer-specific learning pace and, with relatively little computing overhead, helps to prevent saddle points sooner. Using our adaptive learning rates on top of established optimization strategies almost always boosts efficiency on regular datasets, as we illustrate in Section IV.

IV. EXPERIMENTAL RESULTS

A. Dataset

On three regular datasets, we present picture classification results: MNIST, CIFAR10 and Image Net (ILSVRC 2012 dataset, part of the Image Net challenge). MNIST provides 60,000 digital handwritten images for preparation and 10,000 digital handwritten images for research. In each class, CIFAR10 consists of 10 groups of 6,000 pictures. 1.2 million Colour photographs from 1000 separate groups are used in Image Net. B. Experimental Information to enforce our process, we use Cafe [16]. Stochastic Gradient Descent (SGD), Nester's Accelerated Gradient (NAG) and AdaGrad are optimization methods given by Cafe. We apply our adaptive layer-specific learning rate approach on top of both of these optimization methods for a fair contrast between state-of-the-art methods. The efficacy of our algorithm on convolutionary neural networks on 3 datasets is seen in our experiments. On CIFAR10, as provided in Cafe, we use the same global learning rate. Although our approach often rises the layer-specific learning rate on the basis of the global learning rate (with regard to other optimization methods), we begin with a slightly lower learning rate of 0.006 to render the Image Net experiment less aggressive for learning. With the learning rate used in [2] for studies performed on Image Net, SGD was initialized. 1) MNIST: For our trials on MNIST, we use the same architecture as Lent. On top of stochastic gradient descent, Nester's accelerated gradient approach and Adored on the MNIST dataset; we present the effects of the usage of our suggested layer-specific learning speeds. Since all approaches agree on this dataset very easily, we just present the precision and loss for the first 2,000 iterations. Table I am presenting

Iteration	SGD	Over-SGD	Nesterov	Over-NAG	AdaGrad	Over-AdaGrad
200	7.90 ± 0.44	7.25 ± 0.46	6.66 ± 0.47	5.57 ± 0.5	4.12 ± 0.32	3.40 ± 0.3
600	3.29 ± 0.22	3.05 ± 0.21	3.01 ± 0.19	2.84 ± 0.17	2.21 ± 0.14	1.95 ± 0.16
1000	1.89 ± 0.08	1.80 ± 0.13	1.92 ± 0.07	1.83 ± 0.18	1.66 ± 0.11	1.57 ± 0.14
1400	1.60 ± 0.11	1.49 ± 0.09	1.74 ± 0.12	1.52 ± 0.11	1.01 ± 0.08	1.01 ± 0.09
1800	1.52 ± 0.09	1.41 ± 0.12	1.56 ± 0.09	1.37 ± 0.09	1.41 ± 0.06	1.34 ± 0.08

the

TABLE me: The mean error rate on MNIST as seen in the table after multiple iterations for stochastic gradient descent, the accelerated gradient of Nester and Adored with their layer-specific adaptive models. Each procedure has been executed 10 times, and its mean and standard deviation have been recorded.

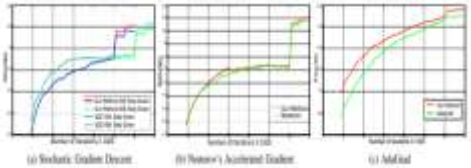


Fig. Fig. 1: CIFAR data set: accuracy-showing plots (Figures 1a-1c) contrasting SGD, NAG and AdaGrad, each with our layer-wise adaptive learning speeds. We display results for the SGD plot both when we move down the learning rate at 50,000 iterations and at 60,000 iterations.

When each procedure was run 10 times, the mean precision and standard deviation. We notice that our layer-specific learning rate proposed is consistently higher than the accelerated gradient, stochastic gradient descent and AdaGrad of Nesterov. Including stochastic gradient descent, Nesterov's accelerated gradient and AdaGrad, the suggested approach also achieves the highest precision of 99.2 percent in all the experiments.

2) CIFAR10:

We use a convolutionary neural network on CIFAR10 with 2 layers of 32 characteristic maps from 5 to 5 convolution kernels, each accompanied by 3 to 3 total pooling layers. After that, we have another convolution sheet, accompanied by a 3 ?? 3 max pooling layers, with 64 functions maps from a 5 ?? 5 convolution kernels. Finally, with 10 secret nodes and a soft-max logistic regression layer, we have a totally linked layer. A ReLu non-linearity is introduced after each convolution sheet. This architecture is the same as that defined by Cafe. The learning performance for the first 60,000 iterations was 0.001 and at 60,000 and 65,000 iterations it was reduced by a factor of 10. On this dataset, we find again that our method's final error and failure is consistently lower than SGD, NAG and AdaGrad (Table II). Our adaptive strategy achieves a lower precision after phase down than both SGD and NAG. Notice that we can get an increase of 0.32 percent over the mean accuracy for SGD just using our optimization approach (without modifying the network architecture). We achieve an accuracy of 82.08 percent, which is higher than SGD after 70,000 iterations, even though we decrease the learning rate at 50,000 iterations (taking 60000 iterations in total), greatly decreasing the needed training period Fig. 1. Because our approach converges even quicker when used with SGD, the phase down on the learning rate may be done much sooner, theoretically further minimizing training time. While Adagrad does not do very well with default parameters on CIFAR10, a 1.3 percent increase over the average final accuracy is observed, with a substantial speed-up in training period again.

3) Image Net:

To equate our approach with other optimization algorithms, we use an implementation of Alex Net [2] in Cafe, deep convolutional neural network architecture. AlexNet consists of 5 layers of convolution, accompanied by 3 layers that are entirely linked. See the paper [2] for more information about the architecture. Because Alex Net is a deep neural network with considerable difficulty, it is necessary to extend our approach to this design of the network. The outcomes by using our approach over SGD are seen in Fig 2. We remember that after 100,000 and 200,000 iterations, our system obtains substantially greater precision and lower loss. In comparison, on the validation set after 295,000 iterations, which SGD completes only after 345,000 iterations, we are still able to meet the maximum precision of 57.5 percent, resulting in a 15 percent reduction in training period. This is a substantial reduction, considering that such a large model requires more than a week to train properly. Across all iterations, our loss is still consistently smaller than SGD. We execute a step-down by a factor of 10 for every 100,000 iterations in the current model. Until performing a phase down, we adjust the amount of training iterations at a given learning pace in order to evaluate how our strategy works as we reduce the number of training iterations. After 350,000 iterations of SGD and our operation, Table III demonstrates the final accuracy. But as we reduce the amount of iterations after which we conduct the phase down in the learning pace, the final accuracy decreases marginally, it is apparent that our approach achieves better accuracy than SGD. Notice that we report accuracy to the top-1 class. Our findings are marginally lower than those stated in [2], because we use the Cafe implementation of the Alex Net framework and do not use any data augmentation techniques.

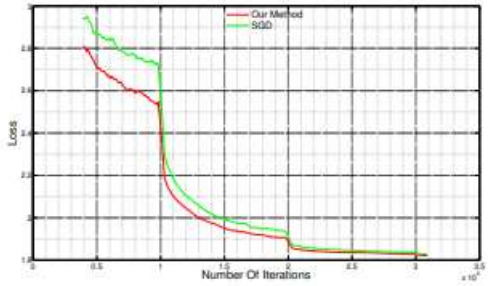
IV. CONCLUSIONS

A general technique for training deep neural networks utilizing layer-specific adaptive learning rates is proposed in this article.

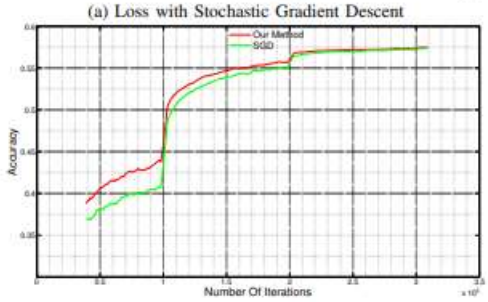
Iteration	SGD	Ours-SGD	Nesterov	Ours-NAG	AdaGrad	Ours-AdaGrad
5000	68.8 ± 0.49	70.10 ± 0.89	69.36 ± 0.31	70.10 ± 0.59	54.90 ± 0.26	57.53 ± 0.67
10000	74.05 ± 0.51	74.48 ± 0.59	73.17 ± 0.25	74.00 ± 0.29	58.26 ± 0.58	60.95 ± 0.59
25000	77.40 ± 0.32	77.43 ± 0.15	76.17 ± 0.61	77.20 ± 0.59	63.02 ± 0.95	64.90 ± 0.57
60000	78.76 ± 0.67	78.74 ± 0.38	78.35 ± 0.35	78.18 ± 0.65	66.86 ± 0.93	68.03 ± 0.23
70000	81.78 ± 0.14	82.10 ± 0.32	81.75 ± 0.25	81.92 ± 0.26	67.04 ± 0.91	68.30 ± 0.39

ISSN NO: 9726-001X
Volume 08 Issue 01 March 2020

TABLE II: Mean accuracy on CIFAR10 as seen in the table after multiple iterations for SGD, NAG and AdaGrad with layer-specific adaptive models. There is a report of the mean and standard deviation over 5 runs.



(a) Loss with Stochastic Gradient Descent



(b) Accuracy with Stochastic Gradient Descent

Fig. Fig. 2: Data collection on Image Net: plot relating stochastic gradient descent to our layer-wise adaptive learning speeds. Throughout all iterations, we can see a clear increase in precision and loss over the standard SGD process.

Iterations	SGD	Our Method
70,000	55.72%	55.84%
80,000	56.25%	56.57%
90,000	56.96%	57.13%

TABLE III: Contrast of stochastic gradient descent and our step-down approach at various iterations on Image Net, which can be used with a global learning rate on top of any optimization method.

To calculate an adaptive learning rate for each layer, the system uses gradients from each layer. When the parameters are in a low curvature saddle point area, it aims to speed up convergence. Layer-specific learning rates often enable the system to avoid slow learning, typically induced by very small gradient values, in the initial layers of the deep network.

REFERENCES

[1] R. Pascanu, Y. N. Dauphin, S. Ganguli, and Y. Bengio, "On the saddle point problem for non-convex optimization," arXiv preprint arXiv:1405.4604, 2014.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, 2012, pp. 1097–1105.

[3] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on. IEEE, 2014, pp. 1701–1708.

[4] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP). Citeseer, 2013, pp. 1631–1642.

[5] 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," Signal Processing Magazine, IEEE, vol. 29, no. 6, pp. 82–97, 2012.

[6] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier networks," in Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP Volume, vol. 15, 2011, pp. 315–323.

[7] H. Robbins, S. Monro et al., "A stochastic approximation method," The Annals of Mathematical Statistics, vol. 22, no. 3, pp. 400–407, 1951.

[8] M. D. Zeiler, "Adadelta: An adaptive learning rate method," arXiv preprint arXiv:1212.5701, 2012.

[9] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," The Journal of Machine Learning Research, vol. 12, pp. 2121–2159, 2011.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] A. J. Bray and D. S. Dean, "Statistics of critical points of gaussian fields on large-dimensional spaces," Physical review letters, vol. 98, no. 15, p. 150201, 2007.

[12] Y. V. Fyodorov and I. Williams, "Replica symmetry breaking condition exposed by random matrix calculation of landscape complexity," Journal of Statistical Physics, vol. 129, no. 5-6, pp. 1081–1116, 2007. [13] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," USSR Computational Mathematics and Mathematical Physics, vol. 4, no. 5, pp. 1–17, 1964.

[14] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," in Soviet Mathematics Doklady, vol. 27, no. 2, 1983, pp. 372–376.

ISSN NO: 9726-001X
Volume 08 Issue 01 March 2020

[15] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in Proceedings of the 30th International Conference on Machine Learning (ICML-13), 2013, pp. 1139–1147.
[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," arXiv preprint arXiv:1408 .5093, 2014.