# A Novel way of Training a Neural Network with Reinforcement learning and without Back Propagation

James Lindsay
*Department of Electrical and Computer Engineering*
*Royal Military College of Canada*
Kingston, Ontario, Canada
james.lindsay2@forces.gc.ca

Sidney Gigivi
*School of Computing*
*Queen's University*
Kingston, Ontario, Canada
sidney.givigi@queensu.ca

***Abstract*—This paper introduces, explores and shows a new way to optimize the weights of a neural network. This new technique does not use back propagation or relies on gradient descent. The core idea is to combine both Game Theory and Reinforcement Learning to train a Neural network. We structure a game of learning automata agents, specifically Continuous Action Learning Automata (CALA) agents, that iteratively converges to a global minimum. Each CALA agent is associated with a weight in a neural network, and when the game converges to a global minimum, we can say that the neural network has been trained. By using a game of CALA agents we can deal with input that has been corrupted by noise, do not have to worry about the vanishing gradient problem or worry about over fitting and is structured better for parallel computing.**

***Index Terms*—Neural Networks, Learning Automata, CALA, Non-Gradient descent training methods, Deep Learning, Machine Learning**

## I. Introduction

Since 2006, deep learning has exploded and now is used in most industries [1]. Deep learning uses Artificial Neural Networks (ANNs) that have more than one hidden layer [2]. As more and different applications use ANNs, the chances of the training data being corrupted by noise increases. Therefore, for the success of deep learning to continue, one must solve the training issues associated with ANNs.

In Santharam et al. [3], it is proposed that one could use a game of CALA agents to derive the weights of an ANN, instead of the traditional way of updating the weights with gradient descent. However, to the authors' knowledge, no one has ever actually done this for a real ANN. Since no one has actually used a game of CALA agents to derive the weights of an ANN, there is no literature on the best ways to implement and optimize the CALA game for an ANN. The intent of this paper is to show that it is possible to train an ANN with a CALA game and to explore the advantages and disadvantages of using a CALA based techniques compared to standardized gradient descent. The approach we will take is to build ANNs, for all the Hebbian perceptron gates (AND gate, NOT gate, OR gate, XOR gate). Then we will train those ANNs and derive weights found using CALA games. If we can show that we can approximate all the hebbian perceptrons, then it is possible to build an ANN, and train it with a CALA game that can approximate any possible circuit.

Since a game of CALA agents does not use gradient descent or back-propagation, many the the problems associated with back-propagation and gradient descent go away. As well, since this is a new way of training ANNs, the paper explores both the positive and negative aspects to this new approach.

It should be noted that the intent of this paper is just to show that training an ANN with a game of CALA agents is possible. Subsequent research is required to optimize this technique. Once optimized, it can be used to train deep ANNs.

The paper is organized as follows. In Section II, we discuss the background of the main topics that this paper touches on, and discuss the training issues that are present in gradient descent training methods. Section III, explores how the experiments will be conducted, to prove that the new method is possible. Section IV shows the results of experiments. Section V compares the results against the established method of gradient descent. Finally, section VI talks about future work and concludes the paper.

## II. Background

### *A. Problems with Back-propagation and Gradient descent*

Gradient descent is a popular optimization algorithm and is the most common way to optimize the weights of an ANN [4]. It is an iterative optimization algorithm that is used to find the minimum of a cost function.

Gradient descent has been widely successful, especially when used to training ANNs. However, gradient descent does have some draw backs that makes it unsuitable for some problems. The biggest problems from which gradient descent suffers are the vanishing gradient problem, the exploding gradient problem, and over fitting .

*1) Vanishing Gradient Problem:* The vanishing gradient problem is found in the training of deep ANNs with back propagation. When training a deep neural network, if the derivatives of the original error are less than one, then the

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

activations of layers further down the network can get exponentially small and thus making training difficult. The training is more difficult since the deep ANN will keep taking smaller and smaller steps and therefore, it will take a long time for gradient descent to learn anything.

There are partial solutions that do not completely solve this problem but improves performance significantly. For example, one can use specifically designed activation functions such as ReLU [5]. Or one can make careful choices in how you initialize the weights.

*2) Exploding Gradient Problem:* Similar to the vanishing gradient problem, when using back propagation on a very deep ANN, if the initial derivative is greater that one, then the activation of the very deep layers will get exponentially large. This means that the steps the ANN takes will be too large and will make it extremely difficult for the ANN to converge on an optimal solution. Like the vanishing gradient problem, the exploding gradient problem can be mitigated by carefully selecting how the initialization of the weights is done or by using a specialized activation function. It should be noted that these methods will significantly reduce the problem but not solve it.

*3) Over Fitting:* Over fitting is a fundamental problem with machine learning models, and ANNs are very prone to it [6]. Over fitting occurs when the function approximation, that the ANN provides, is closely fit to a limited set of data points and thus the function cannot be generalized to a wider input set. As the number of parameters of an ANN increases, the probability that over fitting will occur also increases. Since deep ANNs are usually very large, with many parameters, they are very susceptible to over fitting. There are mechanisms to deal with this problem, such as, early-stopping [6], expanding the input data set with noise variables [7] and dropout [8]. However there are draw backs to these mechanisms as the dropout mechanism takes two to three times longer to train a standard model [8] and designing the stopping criteria for early stopping is a challenge [6].

## B. Continuous Action Learning Automata (CALA)

Learning Automata is a form of Temporal Difference (TD) learning, and TD learning is a Reinforcement Learning (RL) methodology. The primary way Learning Automata (LA) differ from other TD and RL methodologies is that the search for the optimal action is conducted over the probability distribution of the action set, rather than the action set [9]. In fact, the action at any given time is randomly chosen, while the environment gives feedback to change the probability distribution to reflect reality. Due to the probabilistic nature of LA, they deal with noisy input data very well.

LA work as follows: at each time step, the automata choose an action at random based on probability distributions, the response from the environment is observed, and the probability distribution is updated based on that response. Then the procedure is repeated.

There are numerous sub methodologies of LAa. This paper will only focus on Continuous Action Learning Automata (CALA).

A CALA controller works in the continuous space of actions. The set of actions of a CALA is the real line. The probability distribution is the normal distribution with a mean value and a standard deviation value that gets updated whenever reinforcement is received from the environment. Like all learning automata, a CALA is very good at dealing with noisy inputs.

## C. CALA Games

A CALA agent can be very effective at finding the optimization of one parameter. However, in an ANN, each weight is one parameter and the vast majority of ANNs have more than one weight. Luckily we can group CALA agents in groups, these groups of CALA agents are called a game of CALA agents. In a game of CALA agents, each individual CALA agent is trying to solve its own objective; while at the same time the agent is also contributing to a global objective. Due to the nature of game theory, we can have relatively simple individual agent objective functions, but at the same time have a complex global objective function [10].

In Lindsay [11], the authors used a CALA game to optimize the weights of two PID controllers that controlled the steering for a robot. To the authors knowledge this was the first publication that showed how one could use a game of CALA agents on a real world problem and this paper follows similar methodologies in constructing the CALA agents and structuring the CALA game. The paper showed how a game of CALA agents can ingest input that that is ever changing, dynamic and noisy but still optimize the weights better than classical methods to derive optimal PID values.

A game of CALA agents are especially suited for common payoff games. If the learning step-size, $\lambda$, is sufficiently small then a CALA team will converge to a modal point. That means a game of CALA agents can be efficiently used in solving optimization problems of regression functions in a noisy environment. In our case the weights of an ANN even when the training samples are noisy [9]. As well, due to the probabilistic nature of the CALA agents, this technique should be resistance the problem of over fitting [7].

In a game of CALA agents, the agents are completely decentralized, so there is no requirement to exchange information or even know about the players or if they are part of a game. This decentralized, no information sharing features approach lends itself to run on parallel computing infrastructure. As well, no matter the size of the game, the game only needs to calculate the global objective function once, at each time step. Therefore, a game of CALA agents trying to find the weights of an ANN should never run into the vanishing gradient problem and there is no theoretical limit on how large the game can be, thus there is no theoretical limit on how large, or deep, the ANN could be.

### *D. Other approaches to solving the problems posed by Back-propagation and Gradient descent*

The problems posed in section II-A are well known and many algorithms can be used to mitigate them. Some of the more popular algorithms are: Heuristically Enhanced Gradient Approximation (HEGA) [12]; weight perturbation [13] and Alopex [14]. These algorithms differ from conventional back-propagation and gradient descent in many ways; however, they still utilize some aspects of back-propagation or gradient descent. For example, Both HEGA and weight perturbation, do not use gradient descent but does use back propagation. Whereas, using a game of CALA agents to train an ANN does not require either back-propagation or gradient descent.

## III. Proof of Concept

Since Santharan et al. [3] only theorized that it could be possible to optimize the weights of an ANN with a game of CALA agents, but gave no indication how one could actually do it, the intent of this paper is to show that the concept is possible. The authors make no claim about the optimality of their approach. For our proof of concept we built eight standard feed forward ANNs. Four of those ANNs perform back propagation using a standardize method of gradient descent. These are known to be accurate function approximations for the four hebbian perceptron circuits (NOT Gate, OR Gate, AND Gate, and XOR Gate). Then we built four ANNs that approximate the four hebbian perceptron circuits, but instead of using back propagation to tune the weights, we used a game of CALA agents that minimized the global cost function.

In both cases (gradient descent and CALA games) we are using an ANN to approximate logistic regression. We will use a sigmoid activation function, which returns a number between 0 and 1, in order the generate a prediction $\hat{y}$, (1). This prediction is then evaluated within a loss function, $\mathcal{L}(\hat{y}, y)$, where $y$ is the expected output. Since this is an optimization problem we need a loss function that is convex, to ensure that we find the global optimum [15]. Therefore the loss function, that is used on a single training example, is given by (2) and the cost function, that is used for the entire training set, is given by (3). In (1), $w$ is the vector of weights of the ANN, $b$ is the bias of the ANN and m is the number of examples in the training set.

$$\hat{y} = \sigma(wx^T + b) \tag{1}$$

$$\mathcal{L}(\hat{y}, y) = -(ylog\hat{y} - (1-y)log(1-\hat{y})) \tag{2}$$

Therefore, the goal of both gradient descent and CALA game based algorithm is to find $w$ and $b$ values that minimize $J(w, b)$.

$$\begin{aligned} J(w,b) &= \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}, y) \\ &= -\frac{1}{m}\sum_{i=1}^{m}[y_i log\hat{y}_i + (1-y_i)log(1-\hat{y}_i)] \end{aligned} \tag{3}$$

For ANNs that are approximating the NOT, AND and OR gates, we use a two layer ANN. The NOT gate has two nodes in the input layer (a bias plus one input node) and one node in the output layer, for a total of two weights. The AND and OR gate have three nodes in the input layer (a bias node plus two input nodes) and one node in the output layer, for a total of three weights. Fig. 1 is the computational graph for a two layer, three node ANN, that we built. The XOR gate requires a three layer ANN, that has three nodes in the input layer (a bias plus two input nodes), four nodes in the hidden layer, and one node in the output layer for a total of thirteen weights.

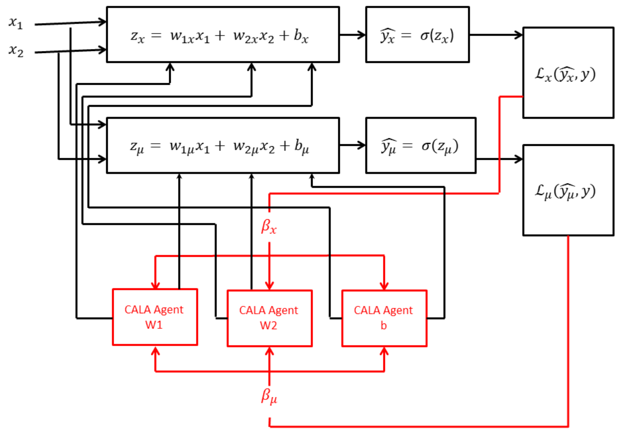
$x_1$
$x_2$
$z_x = w_{1x}x_1 + w_{2x}x_2 + b_x$
$\widehat{y_x} = \sigma(z_x)$
$\mathcal{L}_x(\widehat{y_x}, y)$
$z_\mu = w_{1\mu}x_1 + w_{2\mu}x_2 + b_\mu$
$\widehat{y_\mu} = \sigma(z_\mu)$
$\mathcal{L}_\mu(\widehat{y_\mu}, y)$
$\beta_x$
CALA Agent W1
CALA Agent W2
CALA Agent b
$\beta_\mu$

Fig. 1. Computational graph for a 2 layer ANN, tuned by a CALA game

As illustrated in Fig. 1, in the CALA approach, each weight of the ANN will be associated with a CALA agent. At the start of each training iteration (an iteration is defined as the four possible input/output pairings for the hebbian gates), the CALA agents will select a random value based off each agents' mean and standard deviation. Also, a separate ANN is generated that uses the mean of each CALA agent as the weight. These two ANNs then use the cost function (3) to generate two output signals used as feedback to generate the next CALA game that will be used in the next iteration.

The CALA game takes the two output signals, which are interpreted as reinforcement signals $\beta_{\mu(k)}$ and $\beta_{x(k)}$ respectively. Each CALA agent then updates the mean, $\mu(k)$, and standard deviation, $\sigma(k)$, with the following learning algorithms [9]:

$$\mu(k+1) = \mu(k) + \lambda\frac{\beta_{x(k)} - \beta_{\mu(k)}}{\phi(\sigma(k))}\frac{x(k) - \mu(k)}{\phi(\sigma(k))} \tag{4}$$

$$\begin{aligned} \sigma(k+1) = \sigma(k) + \lambda\frac{\beta_{x(k)} - \beta_{\mu(k)}}{\phi(\sigma(k))}[(\frac{x(k) - \mu(k)}{\phi(\sigma(k))})^2 - 1] \\ -\lambda K(\sigma(k) - \sigma_L) \end{aligned} \tag{5}$$

where $\lambda$ is the learning parameter ($\lambda = 0.05$), $K$ is a constant ($K = 0.5$), $\sigma_L = 0.0005$ is the lower bound on $\sigma$, and $\phi$ is a function that returns the standard deviation $\sigma(k)$ but bounds the standard deviation to $\sigma_L$ [9]. This process then repeats itself at each time step until the cost function converges to a value that is under a certain threshold.

Each CALA agent starts with a mean, $\mu(0)$, of 0.0 and a standard deviation, $\sigma(0)$, of 0.25.

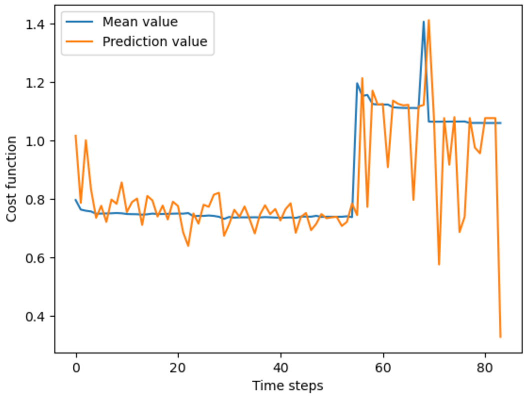

Fig. 2. NOT Gate CALA results

## IV. Results

We ran four experiments, where we trained ANNs to approximate the four hebbian perceptron gates. Figs. 2 to 5 show the results of the NOT, AND, OR and XOR gates. In each ANN we assigned a CALA agent to a weight and then updated the CALA agents with (4) and (5). Every time step consists of the four possible states a gate could be in ([0,0],[0,1],[1,0],[1,1]). We calculate the loss function, (2), for each state and at the end of the time step calculate the cost function, (3). We set a threshold of 0.35 on the cost function and used that as our stopping criterion. At every time step in a game of CALA agents, two signals are recorded, the cost function of the prediction, $\beta_x$ and the cost function of the mean, $\beta_\mu$. As well, Figs. 2 to 4 show how the algorithm works. The mean stays somewhat constant only slowly moving in the direction of convergence, while the prediction is exploring all the possibilities and some times getting a very high or low score.

As can be seen in Fig. 3, and in an extreme case, Fig. 5, the cost function of the mean, $\beta_\mu$, can spike. When we examine (4), this happens when the standard deviation becomes very low. Spiking of the mean signal is problematic, since without the spike we would have a faster rate of convergence. However, the more interesting thing about the spikes of the mean is how fast the mean returns to its original position. This can be seen in Fig. 5, although the mean is constantly spiking, it quickly returns to a value between 0.3 and 0.4. This is a good indication of the robustness of the algorithm we are using.

Fig. 6 is the graph of a NOT gate focused on one of these spikes. The graph includes the cost functions of both the mean, $\beta_\mu$, and prediction, $\beta_x$, as well as the standard deviation of the CALA agents that are associated with the two weights. Between time step 121 and 122, the mean value spikes. It can be seen that at time step 121 the standard deviation is at it lower bound. The standard deviation is at its lower bound because at time step 120, the difference between $x(k) - \mu(k)$ is a large negative number, in this case $-2.248$ and, when substituted in (5) it drives the standard deviation to $\sigma_L$.

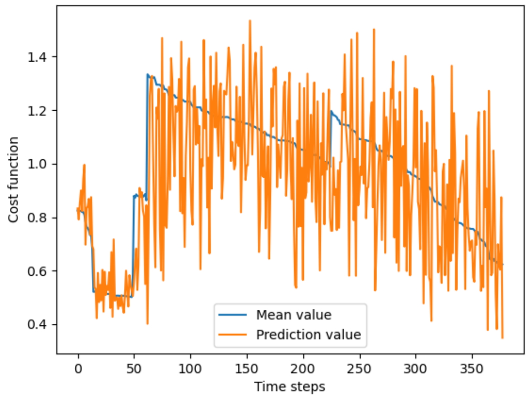

Fig. 3. AND Gate CALA results

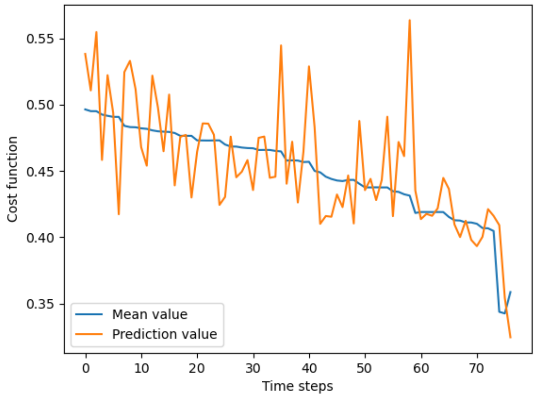

Fig. 4. OR Gate CALA results

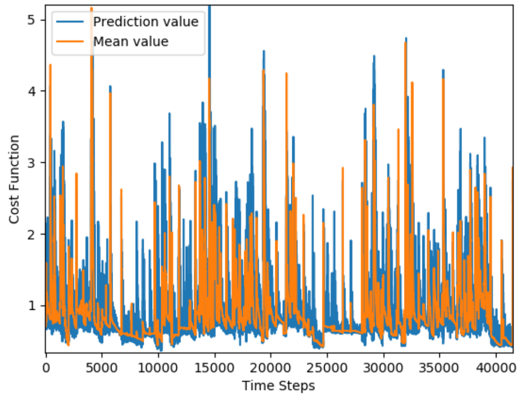

Fig. 5. XOR Gate CALA results

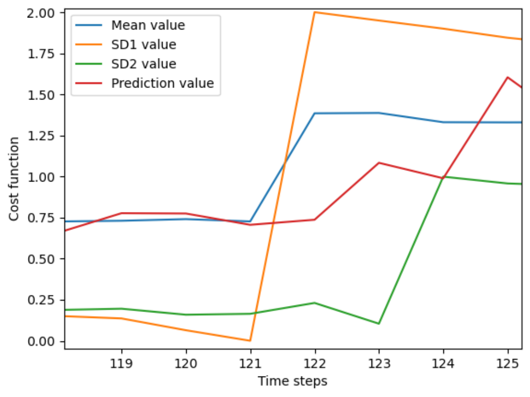

Fig. 6. NOT Gate spike example

TABLE I
NOT GATE

| Method | Time step | Standard Deviation |
|---|---|---|
| $\lambda = 0.05$ | 79 | 37.63 |
| $\lambda = 0.01$ | 367 | 177 |
| $-1.0 < (x(k) - \mu(k)) < 1.0$ | 72.6 | 30.97 |

There are two ways that we tried to solve this "spike" problem. The first way is to lower the learning rate ($\lambda$) to 0.01. The second way is to bound the $x(k) - \mu(k)$ term to -1.0 to 1.0. We ran the method identified in section III, and the two methods mentioned above, a total of 10 times. Table I shows the averages and the standard deviations. Although lowering the learning rate, decreased the amount of spikes, it did not eliminate the problem while also adding a significant amount of time steps to convergence. As well, bounding the variables that were problematic did not give us a significantly better result and did not eliminate the spiking problem.

## V. Comparison against Gradient descent

As discussed in section III, we built a series of CALA games that generated the weights of the ANNs that approximated the hebbian perceptron gates. We trained those same ANNs with standardized gradient descent. Table II contains the results of both methods and compares them.

TABLE II
COMPARING CALA METHOD WITH GRADIENT DESCENT

| Gate Type | Average Time Step for CALA | Standard deviation (for CALA) | Time step for Gradient descent | Standard Deviation (for Gradient descent) |
|---|---|---|---|---|
| NOT | 79 | 37.63 | 23 | 0 |
| AND | 330 | 141.2 | 349 | 0 |
| OR | 59 | 26.2 | 212 | 0 |
| XOR | 270,201 | 228,189 | 6,998 | 233.3 |

TABLE III
GRADIENT DESCENT WITH RANDOM PROBABILITY

| Gate Type | Time step for Gradient descent (random start) | Standard Deviation (for Gradient descent) |
|---|---|---|
| NOT | 23.4 | 0.7 |
| AND | 356.8 | 10.14 |
| OR | 204.9 | 15.97 |

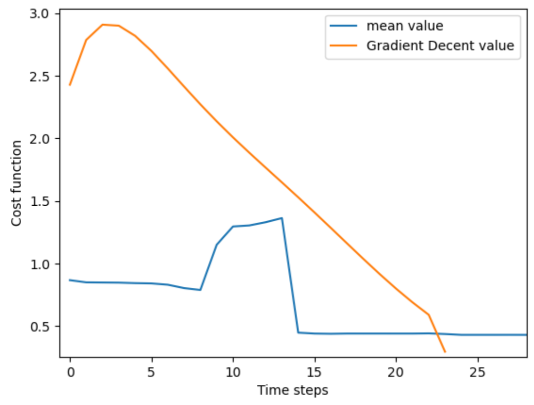

Fig. 7. NOT Gate CALA mean and gradient descent results

As can be seen in table II, there is no standard deviation for the gradient descent method for the NOT, OR, and AND gates. These gates behaved in a deterministic manner, because there was no random variables within the code. Since a core feature of a CALA agent is that the agent selects actions based on random variables, it would be interesting to add a bit of randomness to the gradient descent gates. Table III contains results, where the weights are initialized with a random variable between $0$ and $0.5$

As can be seen in table II, the XOR takes significantly more time steps before it reaches convergence. This is primarily due to the fact that thirteen weights make up this ANN, while only three weights make up the ANN that approximates the OR gate. Due to the exponential increase in complexity with the addition of new perceptron, additional work should explore bounding the CALA game where each layer of ANN has its own independent constraint.

It can also be seen that the ANNs that have a smaller number of weights on average converge faster than gradient descent algorithms. However, the standard deviation is far higher. In order to visualize this, figures 7 to 10 are the cost function for the four hebbian perceptron gates, with both the gradient descent and the mean value from the CALA game. It should be noted that since we chose to display experiments where both the cost functions converges in roughly the same amount of time steps, in most figures the CALA game had a higher cost function score. As well, by inspection, we can also conclude that the "spikes" in the CALA mean are an issue for all the gates. More research needs to be done to smooth these spikes.

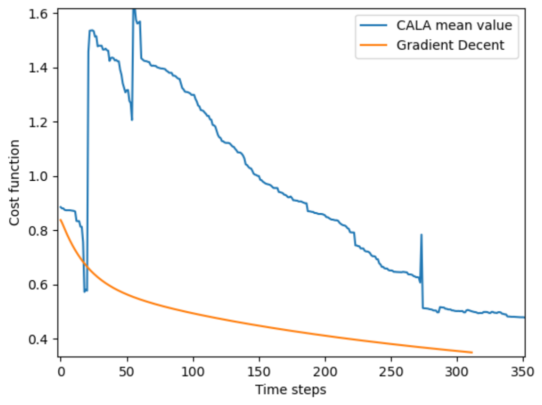

Fig. 8. AND Gate CALA mean and gradient descent results

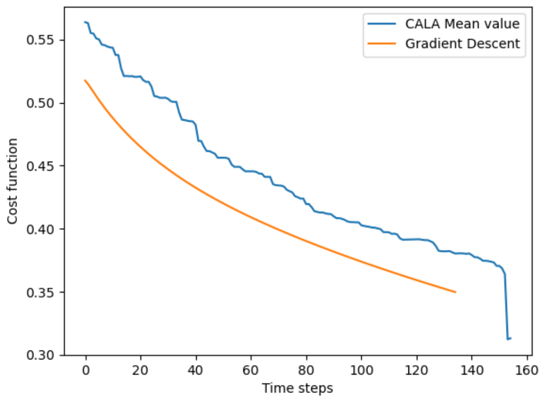

Fig. 9. OR Gate CALA mean and gradient descent results

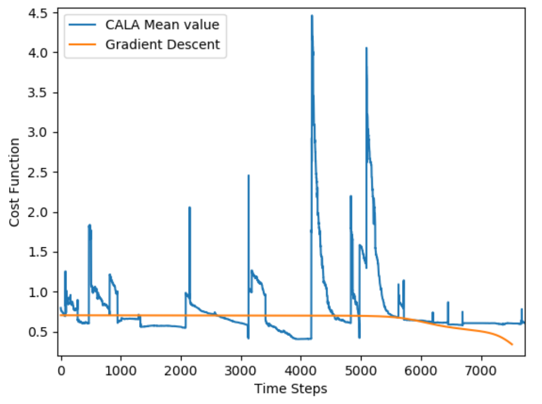

Fig. 10. XOR Gate CALA mean and gradient descent results

## VI. Conclusion and Future Work

The paper has shown that it is possible to train a ANNs with a game of CALA agents. As the field of deep learning keeps expanding, ANNs are going to grow at an exponential rate. The major benefits of using this technique is that it solves many of the problems that deep ANNs will encounter in the future, mainly, the vanishing/exploding gradient problem, over fitting and input data that is corrupted by noise.

Another advantage of using a game of CALA agent methodology is that each agent is decentralized and relatively simplistic. Therefore, computing time can be significantly cut down by using parallel computing infrastructure. More research is required to figure out the optimal way to make use of parallel computing infrastructure.

Future work should include optimizing the algorithms presented in this paper. This optimization should include ensuring that the mean of the CALA agents does not spike, and finding a way to run each layer of the ANN as its own game. After this is done, a game of CALA agents should be run on a deep ANN.

## References

[1] Y. Li, "Deep reinforcement learning: An overview," *CoRR*, vol. abs/1701.07274, 2017. [Online]. Available: http://arxiv.org/abs/1701.07274

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[3] G. Santharam, P. Sastry, and M. Thathachar, "Continuous action set learning automata for stochastic optimization," *Journal of the Franklin Institute*, vol. 331, no. 5, pp. 607 – 628, 1994.

[4] S. S. Du, X. Zhai, B. Poczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=S1eK3i09YQ

[5] A. F. Agarap, "Deep learning using rectified linear units (relu)," *CoRR*, vol. abs/1803.08375, 2018. [Online]. Available: http://arxiv.org/abs/1803.08375

[6] L. Prechelt, *Early Stopping But When?*, 01 2012, pp. 53–67.

[7] B. Ghojogh and M. Crowley, "The theory behind overfitting, cross validation, regularization, bagging, and boosting: Tutorial," 2019.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 19291958, Jan. 2014.

[9] M. A. L. Thathachar and P. S. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[10] R. B. Myerson, *Game Theory: Analysis of Conflict*, 1st ed. Harvard University Press, 1997.

[11] J. Lindsay and S. Givigi, "Solving home robotics challenges with game theory and machine learning," *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1294–1299, 2018.

[12] D. Panagiotopoulos, C. Orovas, and D. Syndoukas, "A heuristically enhanced gradient approximation (hega) algorithm for training neural networks," *Neurocomputing*, vol. 73, pp. 1303–1323, 03 2010.

[13] M. Jabri and B. Flower, "Weight perturbation: An optimal architecture and learning technique for analog vlsi feedforward and recurrent multilayer networks," *Neural Computation*, vol. 3, no. 4, pp. 546–565, 1991.

[14] K. Unnikrishnan and K. Venugopal, "Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks," *Neural Computation*, vol. 6, pp. 469–490, 05 1994.

[15] A. A. Ahmadi, A. Olshevsky, P. A. Parrilo, and J. N. Tsitsiklis, "Np-hardness of deciding convexity of quartic polynomials and related problems," *Mathematical Programming*, vol. 137, pp. 453–476, 2013.