International Journal of Hybrid Information Technology
Vol.14, No.1 (2021), pp.27-42
http://dx.doi.org/10.21742/ijhit.2021.14.1.02

Improving Learning Performance in Neural Networks

Falah Al-akashi

Faculty of Engineering, University of Kufa, Najaf, Iraq
falahh.alakaishi@uokufa.edu.iq

Abstract

In this paper, we propose an optimization framework for a robust deep learning algorithm using the influences of noisy recurring on artificial neural networks. Influences between nodes in the neural network remain very steady in the convergence towards a superior node even with several types of noises or rouges. Several characteristicss of noisy data sources have been used to optimize the observations in a group of neural networks during their learning process. While the standard network learns to emulate those around, it does not distinguish between professional and nonprofessional exemplars. A Collective system can accomplish and address such difficult tasks in both static and dynamic environments without using some external controls or central coordination. We will show how the algorithm approximates gradient descent of the expected solutions produced by the nodes in the space of pheromone trails. Positive feedback helps individual nodes to recognize and hone their skills, and covering their solution optimally and rapidly. Our experiment results showed how long-run disruption in the learning algorithm can successfully move towards the process that accomplishes favorable outcomes. Our results are comparable to and better than those proposed by other models considered significant, e.g., "large step Markov chain" and other local search heuristic algorithms.

Keywords: *Reinforcement learning, Convergence analysis, Deep learning, Features generation, Unsupervised learning, Robotics, Dynamic learning*

1. Introduction

A large number of optimization research studies has been contacted frequently using efficient deep learning models, e.g., reinforcement learning [1], self-supervised learning [2], unsupervised learning [3], pruning of decision trees [4], swarm intelligence [5], etc. Although self-training is not new, the learning process in a style-based environment is fairly used. For several years' researchers did not show up how those strategies of learning grouped themselves successfully and how nodes demonstrated their plans and preserved equilibrium. These indeed seem serious questions need machine learning techniques for persuading. Applying knowledge unintelligently makes the neural networks useless because applying neural networks in low dimensionality would fail to converge even small changes in weights could lead to significant changes in outputs. When the gradient becomes zero, optimization weight will be failed and data will be overfitting. Also, time complexity plays an important

Article history:
Received (October 6, 2020), Review Result (November 6, 2020), Accepted (December 20, 2020)

Print ISSN: 1738-9968, eISSN: 2652-2233 IJHIT

role in enhancing neural networks; for instance, when it is too high, the algorithm sometimes runs for days even on a small dataset. In our perspective, learning in a neural network could be optimized by increasing the pheromone level deposited for the transition from node to node in its space. This technique could influence and control parameters toward the optimal solution. Unlike previous methods proposed by more recent researchers [6][7], the goal is not to deal with the process of learning, but rather it is important to deal with the outcomes of learning to better increase the performance of optimization. The meta-knowledge could handle commonalities in the outcomes of learning algorithms and often could achieve high performance and superior results [8][9]. The learner algorithm could generalize the base-level learners and their tasks. Since learning is often adequate to optimize some objective functions, learning outcomes of intermediate nodes in the hidden layers in neural networks mostly increases the optimization performance. Machine learning focuses on a philosophy that learning patterns normally in information rather than meticulously crafting rules by hand. The data-driven approaches have been delivered by machine learning techniques in a wide range of application areas. Yet, there is a domain that has prominently been left untouched by machine learning, the planning tools that power machine learning itself. One of the foremost wide used tools in machine learning is mathematical optimization. Optimization in an algorithmic program works as an exceedingly model that tends to collect information and select the best parameters. The algorithm often stays static and it is reserved for human specialists who should toil through several rounds of theoretical analysis and empirical validation to plot a higher problem-solving. While the algorithm proposed by [10] focuses on a learning optimization for practicing models on a specific task, the algorithm proposed by [11] sets a lot of formidable objectives of learning for modeling a task-independent problem. Association has been developed a technique for learning optimization algorithms for high-dimensional random optimization issues, and just like the downside of practicing shallow neural networks. Learning to optimization framework learns association in optimization for using in reinforcement learning problems, and the final structure of association tends to be free continuous optimization algorithm. At each cycle, the algorithm takes a step Δx and uses it to update the present cycle x(i). In hand-engineered optimization algorithms, Δx is computed victimization of some fixed formula φ that depends on the optimal function of the current cycle and past cycles, and often, it is just a task of the present and past gradients. Different optimization algorithms are characterized by it is updated formula φ. Hence, by learning φ, it would learn association in optimization algorithms. Association in optimization algorithms often viewed as a mathematician call method such Markov Decision Process (MDP), that is, the states represent the current cycles. Hence, the matter of learning φ simply reduces to a search problem. Hinging upon the strategy projected to develop an extension of learning optimizations for high-dimensional random issues. Association in optimization algorithms had been used for learning shallow neural networks, it showed that it was outperformed the standard hand-engineered optimization algorithms, such as ADAM [12], AdaGrad [13], and RMSprop [14], as well as optimization algorithms for supervised learning techniques [10]. Changing the attributes of neural networks like weights and learning rates to reduce the losses and provide the most accurate results needs methods or algorithms. Some optimizers work to reduce the loss function between layers by changing the weights, such a method is called ‘Gradient Descent. While in the ‘Stochastic Gradient Descent’ the parameters are altered after computation of loss on each training example, whereas in the ‘Mini-Batch Gradient Descent method, the parameters are updated after every batch. Momentum, Nesterov Accelerated Gradient, Adagrad, AdaDelta, and Adam are other

methods used for optimizing the neural network algorithms. However, researchers are always realistic for using optimization while training the neural network [15].

The rest of this manuscript is organized as follows: In Section 2, we will illustrate some related works. In Section 3, we will briefly describe the metaphor of the learning algorithm. Section 4 will elaborate on the Gradient Descent and Stochastic Gradient Descent algorithms. In Section 5, we will describe our algorithm and formula of the learning process. In Section 6, we will outline the proposed experimental results, and Section 7 summarizes our outcomes based on the analysis of results.

2. Related work

The line of optimization on learning algorithms is fairly recent. The Learning task-independent optimization algorithmic programs used reinforcement learning to call the optimization algorithm [11], while supervised learning was used to exploit learning task-dependent optimization algorithms [10]. In the special case, objective algorithms that train the optimization algorithmic program are trained on area unit loss functions for learning different models. Although these techniques have been used from time to time in the literature [16][17], they were exploited by most researchers to visit disparate strategies with different functions. These techniques all share the target of learning some form of meta-knowledge regarding learning, however, the variety of meta-knowledge that tends to be proposed is different. The learning strategies could be divided into the subsequent two classes (i) what to learn and (ii) how to learn. Methods on a class "what to learn" tend to learn what parameter values of the base level learner level helpful across a family of connected tasks [17]. The meta-knowledge captures commonalities shared by tasks within the family that allows learning on a brand new task from the family to be accomplished more quickly. Most early strategies represent this category; which, this line of research has blossomed into a section that has later refer to as transfer learning and multi-task learning. However, methods on the second class "how to learn" tends to be base-level learner that achieves the simplest performance on a task [18]. The meta-knowledge captures correlations between totally different tasks and therefore the performance of various base-level learners on those tasks. One challenge below this strategy is to decide on a parameterization area of base-level learners that are each made enough to be capable of representing disparate base-level learners and compact enough to allow traceable search over this area. A non-constant parametric that stored examples of totally different base-level learners in a file were proposed by [19]; whereas representation on the base-level learner as an all-purpose algorithm was proposed by [20]. The previous has restricted illustration power, whereas the latter makes search and learning within the area of base-level learners balking. The online learning algorithm of any base-learner could map a sequence of training exemplars to a sequence of predictions and model it as an iterated neural network [21]. Within the scope of this formulation, meta-training is reduced to learning the iterated network, and the base level learner is encoded within the memory state of the iterated network. Hyperparameter optimization could be seen as another example of strategies during this class. The area of base-level learners to go looking over is parameterized by a predefined set of hyperparameters. Unlike the previous strategies, multiple trials with totally different hyperparameter settings on the same class are permissible, then, it is so generalization across classes isn't needed. Guided policy search for learning purposes and trained autonomous optimizers for different classes of convex and non-convex objective functions was proposed to demonstrate that the autonomous optimizers converged faster and better than hand-engineered optimizers [6]. A combination of the adaptive genetic and cuckoo search

algorithms was proposed by [22]. Algorithm intended to train a particular Artificial Neural Network (ANN) to obtain the finest weight load by using a hybrid ANN protocol. The overall accuracy was better than that of the genetic algorithms. Finally, a new method for learning optimization algorithms for high-dimensional stochastic problems was presented by [7], the researchers confirmed that the learned optimization algorithm is robust to change in the stochastic of gradients and the neural network architectures.

3. The metaphor of learning

Neural networks are modeled as a collection of learning nodes that are widely used for time series data prediction [23][24]. Feature extensive and state-of-the-art convergence exposed many facets of neural networks that helped technology capabilities and their applications [25]. Each node in a neural network is linked by a set of nodes represented by a simple feed-forward network to accept a binary vector of inputs and give a single output. The networks technically adjusted using a back-propagation algorithm to observe errors in each node based on the difference between planned output and observed output. This means the learning experience of each node in the networks is based upon observations of other nodes. However, there are three types of observable outputs in the defined network: (i) the new nodes or so-called "observers", (ii) the experienced nodes or so-called "instructors", and (iii) the nodes that give inconsistent results or outputs to the instructors, this kind of node is so-called "rogue". The observers born in a network like a network with randomized weights that needs a randomization model to deal with such case. The randomization models require a relationship between weights to achieve a proper output. Nodes do not have to achieve the exact similar weights as others but the network would be equivalent to any node that is fully learned.

It supposes that there is no centralization in the learning systems while nodes learn by the local structure of information in uncorrelated networks. While the observers' nodes are unable to differentiate between other observers, instructors, and rogues, they considered that any node could be observed to be an instructor. Learning of new or young nodes in a neural network means adjusting network weights through a back-propagation algorithm to measure the errors. Each node makes its error estimation in the output by monitoring what other nodes in the vicinity are doing. That means each node strives to observe its averaged output. The instructors in the network represent nodes that acquired the necessary knowledge to perform the required tasks. Sometimes, instructors do not affect interaction with the observers, but they simply go through their tasks within the vicinity of the learning nodes. In the context of such a learning scenario, they acquire some binary inputs to play as new nodes and give a binary output. Experimentally, binary functions are used XOR to connect all paths in the network. To implement the XOR function, a Sigmoid Neuron, as a node, must be used in the network [17]. Although the rogue nodes could conceptualize in any number of instantiations, the environmental externalities could make some nodes misbehave or, sometimes, be conflicted with others in the network. Rogue outputs were selected for their featured behavior rather than for their conceptual justification. The polymorphisms of rogue nodes have two variable characteristics: the regularity with those that appear in the network and the consistency of their output. The rogues appear either on fixed intervals or random intervals with a fixed frequency while the output is either random or by a specified rule. The number of rogues occurs at once and duration of their influences is varied in preliminary experiments, but in this case, they are assumed to be fixed and equal to the number of available instructors. Although the instructors presented in every stage, the rogues appeared in a small fraction of stages. However, nodes could not discriminate between good and bad observations, resulting

in hard to define the influential rogue of fully persistent forms. Influencing rogue often presents an instructor, and in such case, nodes would use knowledge rather than observation to filter other means from their learning.

In terms of measuring the ability to learn nodes, data must be collected in detail within a time frame and observation steps. Each node experienced its output and differentiated between the observed output and the instructors' output. However, Root-Mean-Squares error has been used to compute the difference between the instructors' average outputs and the observers' average outputs to measure the observer's convergence towards the instructors [27]. At that end, the observer's nodes simply discover the outputs of other nodes labeled observers, instructors, or rogues.

4. Stochastic gradient descent

Neural network algorithms usually work in the solution of the combinatorial problem at hand. The algorithm is usually used to find a tour of minimal length (i.e., an optimal solution of the combinatorial problem). However, we adopt in this work as working in the space of pheromone trails. In other words, we aim at finding an optimal set of pheromones, which can be defined in different ways. In this research, we focus on a specific form of optimization for pheromone values. We will call an optimal set of pheromones a configuration that optimizes the expected value of the solution produced by a node during its forward trip. We then study how this problem had been solved using gradient descent [10] in the continuous space of pheromone trails. In the case of a network, we aim at maximizing the expected value of the inverse length of a node's forward trip, given the current pheromone trails and the path selection rule. The gradient of the scaled "error" ε stochastically defined as:

$$\varepsilon = E\left[{}^{1}/_{L(b_n)}\,|T\right] = \sum_{b_n \epsilon x^n} Pr(b_n\,|T)\,\frac{1}{L(b_n)} \tag{1}$$

Note that the expectation is conditioned on T because the probability of a given tour happening depends on the current pheromone trail vector T, while the "local error" ${}^{1}/_{L(b_n)}$ did not use the weights τ (x, y). Then, for each pair of paths (x, y) ϵX^2, we have:

$$\frac{\partial \varepsilon}{\partial \tau(x,y)} = \sum_{b_n \epsilon x^n} \frac{\partial Pr(b_n|T)}{\partial \tau(x,y)}\,\frac{1}{L(b_n)} \tag{2}$$

The probability of a given path is equal to the product of the probabilities of all the initial events that compose it: if $b_n = (x_1, x_1, \ldots, x_n)\,\epsilon x^n$, then

$$Pr(b_n/T) = \prod_{t=1}^{n} Pr(x_1|\,T,\ b_{t-1}) \tag{3}$$

where b_t is equal to b_n truncated after step t: $b_t = (x_1, x_1, \ldots, x_t)\,\epsilon x^n$, and b_0 is the empty sequence. Therefore,

$$\frac{\partial Pr(b_n|T)}{\partial \tau(x,y)} = Pr(b_n|T)\ \sum_{t=1}^{n} \frac{\partial ln(Pr(x_t|T,b_{t-1}))}{\partial \tau(x,y)} \tag{4}$$

Here, we have supposed that $\Pr(x_t|T, b_{t-1}) > 0$, which is always true because $x_t \nexists b_{t-1}$ as b_t is an acyclic path and because the pheromone trails never fall to 0. Define the eligibility trace of (x, y) in the path b_n as:

$$T_{x,y}(b_n) = \frac{\partial\, ln(Pr(b_n|T))}{\partial \tau(x,y)} = \sum_{t=1}^{n} \frac{\partial ln(Pr(x_t|T,b_{t-1}))}{\partial \tau(x,y)} \tag{5}$$

Then,

$$\frac{\partial \varepsilon}{\partial \tau(x,y)} = \sum_{b_n \epsilon x^n} Pr(b_n|T) \frac{T_{x,y}(b_n)}{L(b_n)} = E\left[\frac{T_{x,y}(b_n)}{L(b_n)}|T\right] \quad (6)$$

We will see later how to calculate the traces $T_{x,y}$. We could show the form of the SGD algorithm. Scaling the gradient of ε represents updating T synchronously in the direction of the gradient of ε:

$$\Delta T = \propto \nabla T \varepsilon \quad (7)$$

That is, for each individual "weight" $\tau(x, y)$:

$$\Delta \tau(x, y) = \propto \frac{\partial \varepsilon}{\partial \tau(x,y)} \quad (8)$$

Where $\propto > 0$ is the step-size parameter or learning rate.

We can make a gradient ascent in the space of pheromone trails by following all possible paths b_n and computing, for each of them, the probability Pr(b_n | T) that a node follows that path during its forward trip (given the current pheromone trails), the length of the corresponding tour L(b_n), and the variable $T_{x,y}(b_n)$ for each (x, y) $\in$ x^2. Once all possible paths had been followed, the original problem could be solved simply by taking the best. Moreover, the size of x^n evolved exponentially with the number of paths, so this algorithm soon becomes unrealistically. Finally, the exact gradient descent performs poorly in many complex domains because it gets trapped in the first local optimum on its way.

In Stochastic Gradient Descent (SGD), a random estimation of the unbiased gradient is involved instead of the real gradient. In the case of our algorithm, the gradient of ε is the expected value of the random variable $T_{x,y}$/L given the current pheromones. Therefore, if we draw independently m paths $b_n^1, b_n^2, b_n^3, \ldots, b_n^m$ in x_n following the probability Pr(b_n|T) and average their contributions $T_{x,y}(b_n^i)$/ $L(b_n^i)$ to the gradient, the result becomes:

$$\frac{1}{m} \sum_{i=1}^{m} \frac{T_{x,y}(b_n^i)}{L(b_n^i)} \quad (9)$$

A random vector that assumed a value is similar to the gradient which is an unbiased estimation of the gradient. This is true regardless of the number of paths sampled, even if only one sample is used to estimate the gradient (i.e., m = 1). The results of the stochastic algorithm have a real complexity. Moreover, it may escape from some low-value local optima on its path. It often makes imperfect moves because the gradient estimation is incorrect, but such moves probably permit jumping out of an incorrect local optimum. Thus, SGD often performs better than the actual gradient descent in large and highly multimodal search spaces. The basis of comparison between the standard or actual gradient descent and SGD is the analogy between the actions of sending a node forward and sampling a tour b_n from Pr(b_n|T). During its forward trip, the action of a node is precisely to sample a solution following the probability distribution. Therefore, the forward component of a network could be used in an SGD algorithm as well, and it just has to change the weight of updated rules.

$$Pr(x_{t+1} = x|\, T, b_t) = \begin{cases} 0 & if \quad x \epsilon b_t \\ \frac{\tau(x_t,x)}{\sum_{y \epsilon x} \tau(x,y)} & otherwise \\ y \epsilon b_t \end{cases} \quad (10)$$

Where x∈ b_t means that the acyclic path traverse x.

5. Algorithm and formula

In our neural network algorithm, the node works as a basic computational agent that seeks the optimal solutions. The algorithm needs to find the shortest path on the weighted network and each node stochastically builds an iterative solution to compare and determine the new shortest paths in the network. At that end, the pheromone level on each edge has been updated. To select the edge, each node builds a solution to move on the edge. To determine the next path, each node shall assume the pheromone level and the length of each available path from the current position. That means, at each turn, the node moves from state x to state y and looks for an intermediate solution. Hence, node k computes a set of nodes in its range A_k(x) with feasible probabilities from the current state and then moves to one with high probability.

$$P_{xy}^{k} = \frac{(f_{xy}^{\propto})(a_{xy}^{\partial})}{\sum_{z \in x}(f_{xy}^{\propto})(d_{xy}^{\partial})} \quad (11)$$

Where $f_{xy}^{\propto}$ denotes the amount of pheromone deposited for the transition from state *x* to state *y*, $\propto$ denotes a parameter to control the influence of $f_{xy}^{\propto}$, d_{xy}^{∂} denotes the desirability of state transition *x y* (typically $1/d_{xy}$, d is the distance) and $\partial \geq 1$ denotes a parameter to control the influence of d_{xy}, f_{xy} denotes the trail level and attractiveness for other possible state transitions.

In terms of pheromone updates, when all nodes have completed their solution, pheromones are usually updated. Increasing or decreasing the level of pheromones is matched to good or bad solutions. Function below shows an example of the global pheromone updating rule:

$$f_{xy} \leftarrow (1\text{-}P)\, f_{xy} + \sum_k \Delta f_{xy}^{k} \quad (12)$$

Where f_{xy} is the amount of pheromone deposited for the transition from x y, *P* is the pheromone evaporation coefficient, and $f_{xy}^{\propto}$ is the amount of pheromone deposited by k^{th} node,

$$\Delta f_{xy}^{k} = \begin{cases} Q/L_k & if\ node\ k\ uses\ curve\ x\ y\ in\ its\ moving \\ 0 & otherwise \end{cases}$$

Where L_k is the length of moving *k* and *Q* is a constant.

$$P.\ T(x, y) + \sum_{k=1}^{m} A_k(x, y) \quad (13)$$

Where A_k(x, y) is the amount of pheromone added to the path *xy* by node *k*.
m is the number of nodes,
P is a parameter of the pheromone decay rate,
A_k is the length of tour completed by node k,
T(x, y) at the next iteration becomes:

$$A_k(x, y) = 1/L_k \quad (14)$$

P tends to remove the decaying learning rate problem. Instead of assembling all last squared gradients, P determines the window of last assembled gradients to part of fixed size 'w'. This exponentially moving average was used rather than the sum of all the gradients.

$$E[g^2](t) = \gamma.E[g^2](t-1) + (1-\gamma).g^2(t) \quad (15)$$

γ is a similar value as the momentum term (around 0.9).

The following iterative training algorithm is similar to classical self-training with some minor differences. Assuming labeled nodes $\{(A_1, B_1), (A_2, B_2),\ldots,(A_n, B_n)\}$ and unlabeled nodes $\{Ά_1, Ά_2, .., Ά_m\}$.

1- Learn an instructor model θ_* to minimize the cross-entropy loss on the labeled nodes.

$$\frac{1}{n}\sum_{i=1}^{n} \ell\,(B_i, f^{noisy}(A_i, \theta)) \qquad (16)$$

2- Learn an un-noised instructor model to generate labels for unlabeled nodes.

$$B_i = f(A_i, \theta_*), \forall_i = 1,.., m. \qquad (17)$$

3- Learn observer model $Ǿ_*$ to minimize the cross-entropy loss in labeled nodes and unlabeled nodes with noise added to the observer's model.

$$\frac{1}{n}\sum_{i=1}^{n} \ell\,(B_i, f^{noisy}(A_i, Ǿ)) + \frac{1}{m}\sum_{i=1}^{n} \ell\,(B_i, f^{noisy}(A_i, Ǿ)) \qquad (18)$$

4- Learn the instructor as an observer and go back to step 2.

However, the main parameters of the algorithm are the pheromone trails r(x, y) associated with every pair of paths (x, y) ∈ x^2. Assume *T* be the bi-dimensional vector collecting all the t (x, y)'s. The essential principle of the algorithm is to simulate artificial nodes that involve the pheromone trails to create a random tour. Once its tour was completed, every node creates a backward trip following the same path and refreshes the pheromones on its way back. Finally, some of the pheromone trails evaporate, that is, they decrease by a fixed factor *p*, 0 < $p \leq 1$, known as the evaporation rate. The behavior of every node is summarized as follows:

Forward procedure:

- Specify the starting path x1 sparsely.
- At every step ∈ $\{1, \ldots, n1 -)$, after following the path

  $b_t = (x_1, x_2, \ldots, x_t) \in x^t$, specify the following path at random after

$$\Pr(x_{t+1} = x \mid T, b_t) = \begin{cases} 0 & \text{if } x \in b_t \\ \dfrac{\tau(x_t, x)}{\sum_{y \in x,\, y \in b_t} \tau(x, y)} & \text{otherwise} \end{cases}$$

  where x∈ b_t means that the acyclic path b_t traverses x.

Backward procedure:

After compositing the path $b_n = (x_1, x_2, \ldots, x_n) \in x^n$, reinforce the pheromone trails $\tau(x_n, x_1)$ by the amount 1/L(b_n). There are other methods of implementing the algorithmic program. Within the original implementation of our network a group of *m* artificial nodes concurrently built m solutions as follows: Initially, all the nodes perform their forward trip while not change the pheromones, for the upcoming "generation" of *m* nodes. The pheromone evaporation stage occurs at every generation, before moving the node backward. The whole update at every generation of every pheromone τ (x, y), (x, y) ∈ x^2 is then:

$$\Delta\tau\,(x, y) = \left(\sum_{i=1}^{m} \frac{\delta_{x,y}\,(b_n^i)}{L(b_n)}\right) - \rho\tau(x, y) \qquad (19)$$

Where $b_n^i \in x^n$ is the path followed by the *i* node in its forward trip (i ∈ $\{1, 2, \ldots, m\}$), and $\delta_{x,y}\,(b_n)$=1 if y is the first successor of x in the tour linked to $b_n \in x^n$ and 0 otherwise.

When *m*=1, the other nodes sent one after one in a completely sequential way, waiting for the previous node to finish its backward trip before posting a new one. In such case, the pheromone updates were built by each node as, for all (x, y) ∈ x^2,

$$\Delta\tau\ (x,y)=\frac{\delta_{x,y}\ (b_n)}{L(b_n)}\ -\rho\tau(x,y) \qquad (20)$$

Where $b_n=(x_1,x_2,\dots,x_n)\ \in\ x^n$ is the path followed by the node in its forward trip.

This pheromone update rule was probably used completely not to the concurrent implementation of node network in which nodes act isolated of each other, while a pheromone evaporation stage was linked with each node. The function ideally replaced by the following rule, proposed for the first time in [28], in which, the enforcement of pheromone trails is multiplied by the evaporation rate ρ:

$$\Delta\tau\ (x,y)=\rho\frac{\delta_{x,y}\ (b_n)}{L(b_n)}\ -\rho\tau(x,y)\ =\rho(\frac{\delta_{x,y}\ (b_n)}{L(b_n)}-\ \tau(x,y) \qquad (21)$$

6. Experimental results

During our preliminary study of using the dataset in AntNet [29], the observer composition was varied. The network was chosen to be feed-forward with two inputs, one hidden layer of four neurons, and one output node. While two and three neurons in the hidden layer produce similar results, four neurons have been chosen to produce results with less training, as the inputs were chosen as a binary vector. In terms of learning starts from instructors without rouges, it is initially important to accept untrained nodes and to emulate them around, it would converge on the output of instructors. To that end, a group of six observers was trained with three instructors and no rogue influences. As shown in [Figure 1], the nodes converged on a proper XOR output, and the observers have little trouble converging in relatively few epochs, the depiction of nodes shows converging on a proper average output (6 nodes, 3 instructors, and 0 rouges) with fixed rouge occurrences and fixed rouge output. The final parameters consisted of the observer's training under varying conditions of rogue influences. The results were tested in groups that sized around three observers to two instructors to two rogues. In all scenarios with instructors, the portion of instructors was raised to three as with the rogues. The instructors were always presented and the rogues appeared intermittently and influence evolved their power like that in the instructors. The rogues in all cases appeared exclusively in one percent of the time duration out of five percent of the time that the observers were learning. This means the rogues presented about 5 percent of the observer time that learned. All simulations in the primary experiment were conducted with the instructor output according to the XOR function. In each epoch, all possible binary inputs were given to the nodes, that is, since the nodes were computed in a two inputs binary function, they have presented four binary vectors of input in sequence. The observers trace and take note of the other nodes' outputs at each input. The learning process tries to find certain types of rogue disturbances in the learning process that beneficial. In some cases, converging is difficult to achieve it and in the case that disturbance assists in shakeup the network weights. After some iteration, the rogue influences shall help the network to converge, and obviously, it seems that the undisturbed instructor-observer group would converge faster.

In all of the conducted training, the rogues had a significant influence on the observers' outputs while the influence was not sustained due to nodes behaved very sensitive to the rogues and they were very sensitive to leave the bad influence. The network was relieved of the rogue inputs that quickly recovered. In this scenario, as shown in [Figure 2], the rouges

had a fixed interval of appearance and a fixed output. Although nodes were still able to converge quickly, their outputs had more trouble converting initially than other cases when the rogues regularly occurred. In that scenario, the rogues had a fixed interval of appearance and a fixed output. While the observer output changed drastically during periods of rogue influences, it returned quickly to its limiting form. Also, it could be observed that the rogue influences were more disruptive when the rogues used random outputs. As shown in [Figure 3] and [Figure 4], the nodes experienced a random rouge occurrence but a fixed rouge output, the recovery from rouge influence while the average nodes' outputs experienced with 6 nodes, 3 instructors, and 3 rouges, as a consequence of fixed rouge occurrences and fixed rouge outputs. When the nodes experienced random rogue occurrences and in a fixed rogue output, it seems it is better to deal with fixed outputs than random outputs for the regularly appeared rogues. [Figure 3] shows the error rate when using 6 nodes, 3 instructors, and 3 rouges, the network causes random rouge occurrences and fixed rouge outputs; whereas the average error of 6 nodes were 3 instructors and 3 rouges that resulted in fixed rouge occurrences and random rouge output. Looking at the nodes' output in [Figure 5], it could be seen that the randomly occurring rouges had the opportunity to disrupt the learning process early, and this seems to be more significant than later disruptions. The learning network involved 8 nodes, 3 instructors, and 3 rouges, which caused random rouge occurrences and a fixed rouge output. In each experiment, we ran 100 trials and each trial was stopped after each node moved 1000 epochs.

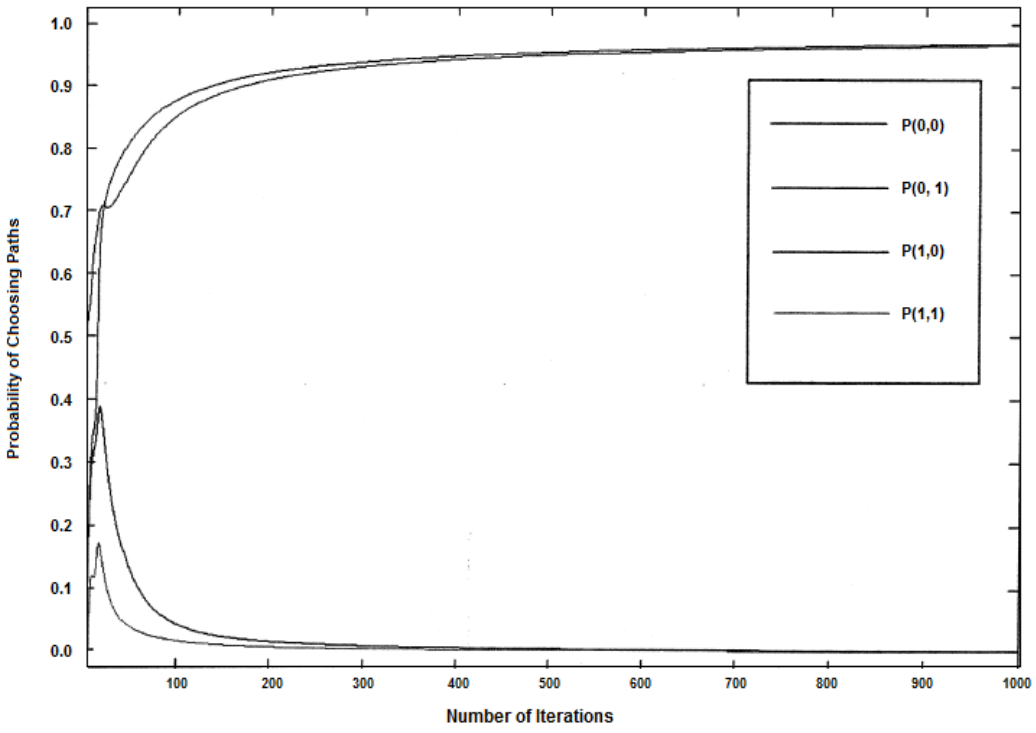

Figure 1. Observers converge to instructors

Evaporation p was set to 0 and the parameter $\propto$ was set to 2 which approximately a normal node behavior. At that end, the network was checked whether the pheromone trail was higher on the long path or not. We found for a given parameter setting the network showed

convergence behavior after 1000 epochs when observers moved toward instructors with some rouges. For a small number of observers (up to 32), the network converged relatively to the longer path. This is due to fluctuations in the path selection of the initial iterations in the algorithm which lead to a reinforcement of the long path. Yet, with an increasing number of nodes, the number of times observed such behaviors decreased drastically, and for a large number of nodes, e.g., 512, it was not observed to the long path in any of the 100 trials. The experiments also indicate that, as could be expected, the network efficiency performed poorly when only one instructor was used, and a number of observers might be large significantly to one that obtained convergence in a short path. [Table 1] shows how three types of nodes affected the results.

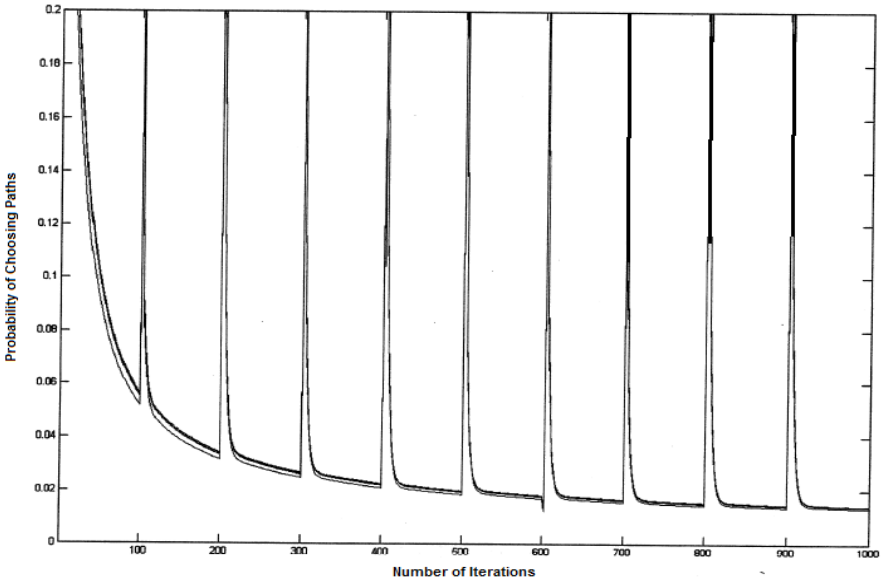

Figure 2. Observers recover from rogue influences

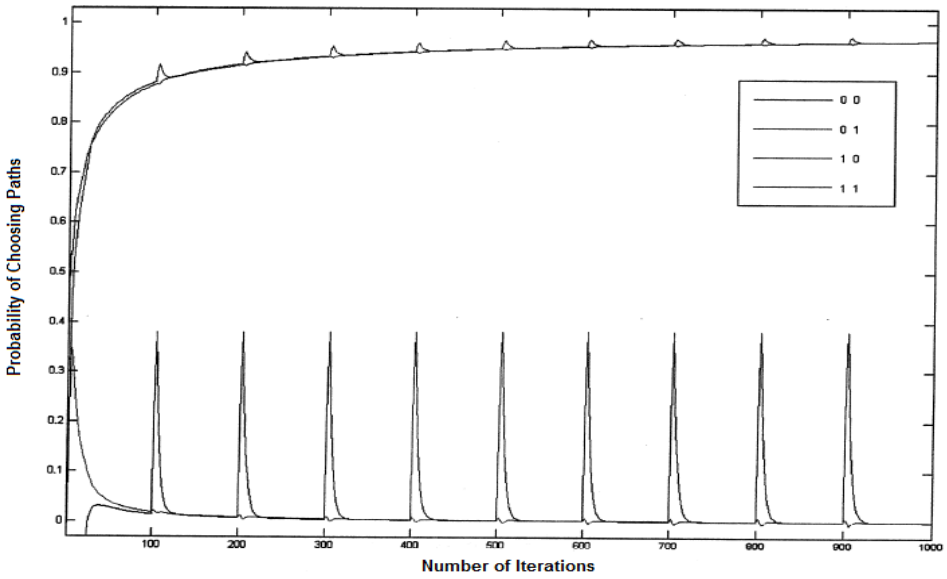

Figure 3. Nodes' errors with random rogue occurrences

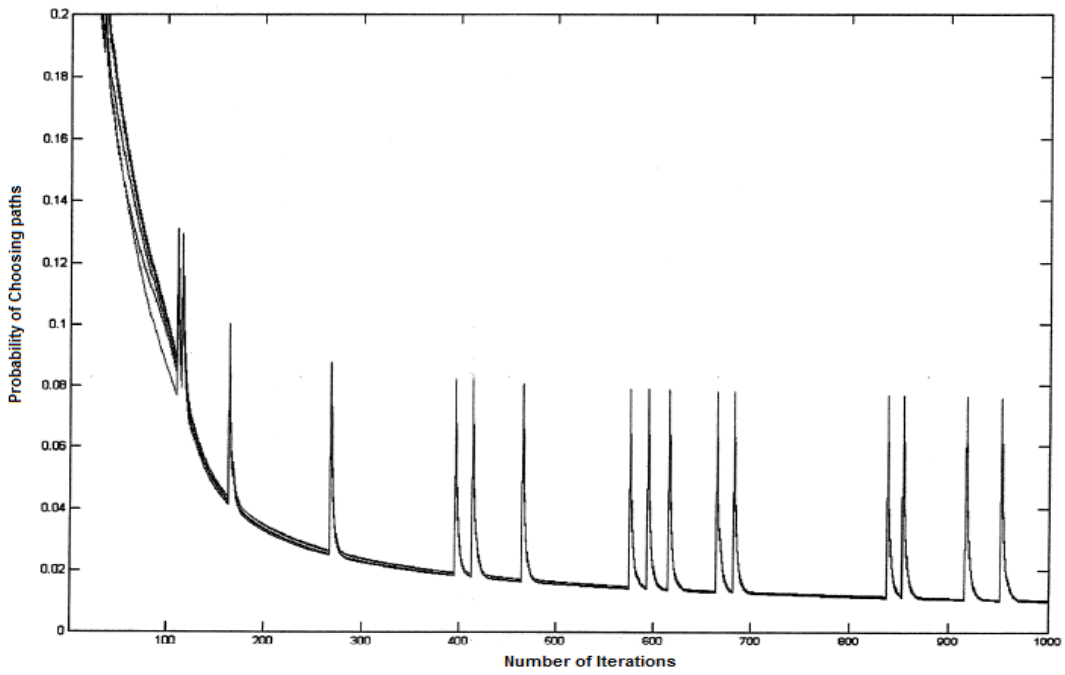

Figure 4. Nodes' errors with random rogue output

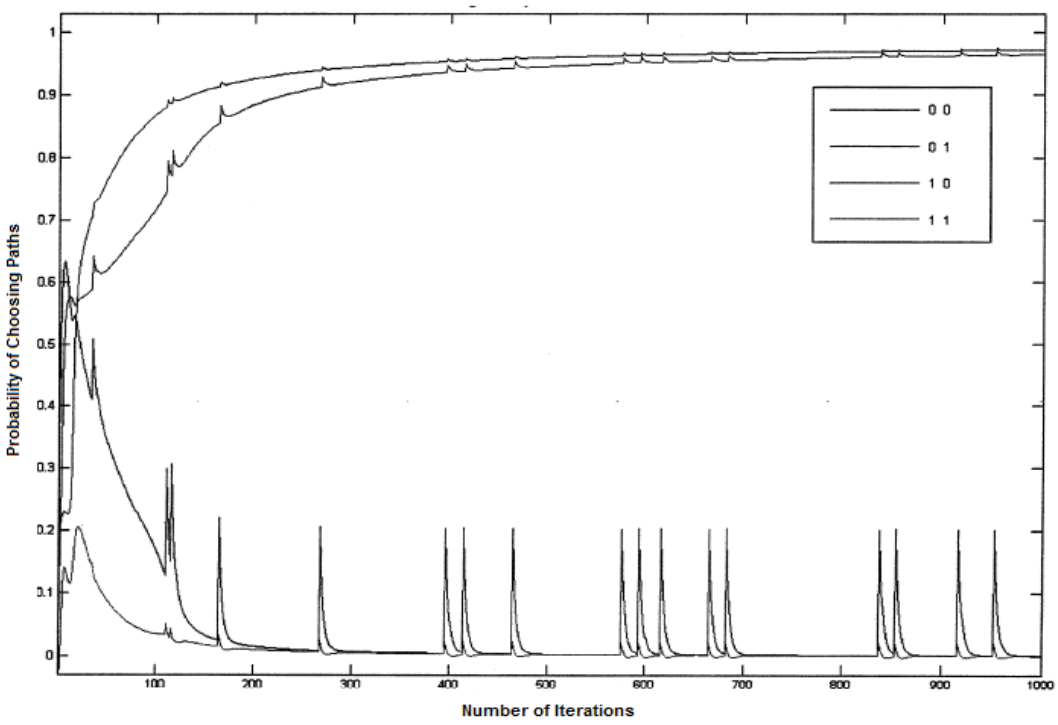

Figure 5. Nodes' outputs with random rouge occurrences and fixed outputs

As shown in [Table 2], in the short path artificial nodes try to build feasible solutions to the proposed problem by moving on a proper graph representation. The developed process was biased by the previous process memorized in a form of pheromone trails, and, in some cases,

by heuristic information for the proposed problem. In the second phase, the solutions constructed by the artificial nodes could be moved to the local optima by a proper local search routine. In the last phase, pheromone trails were updated by the nodes, pheromone evaporation, and/or other proper processes. When pheromone update was independent of the solution quality, network converged to the long path was high frequently. With only one instructor, the network converged to the long path in only 18 out of 100 trials, which is significantly less than in experiment 1, and with 8 instructors or more, it often converges to the short path. In the first case, we found that increasing $\propto$ harmed the convergence behavior while in the second case the results were rather independent of the particular value $\propto$. In general, we found that, in a fixed number of nodes, the algorithm tends to converge to the shortest path more often when $\propto$ was close to 1. This is intuitively clear because a large value $\propto$ tends to amplify the influence of initial random fluctuations. If by chance, the long path is initially selected by instructors, then the search of the whole network is quickly biased toward it. This happens to a lower extent when the value $\propto$ is close to 1. However, in an optimal network, both auto-catalysis and differential path length are worked to favor the emergence of short paths. While the results with network indicate that different path length alone could be enough to let the network converges to the optimal solution on small graphs. Also, relying on this effect as the main driving force of the algorithm comes at the charges of having to use a large network size, which results in long simulation times. Moreover, the efficiency of differentiating the path length was strongly reduced with reducing problem complexity. This is what is shown by the experiments reported as follows, where the column headings give the number m of nodes in the network. The first row gives results obtained when pheromone updates without considering path length, and the second row gives results when pheromone updates proportionally to the path length.

Table 1. Nodes' averages vs. best results

Iteration	Instructor, Observer, with rouge best result (length)	Instructor, Observer, with rouge best result (sec)	Instructor, Observer, with rouge average (length)	Instructor, Observer, with rouge average (sec)	Optimal result	Error %
200	12,520	11	12,527	207	12,520	0.07 %
400	35, 066	96	35,066	421	35,042	0.03%
600	21,244	105	21,317	615	21,644	0.08%
800	9,014	720	9,187	9,20	9,099	0.25%

Table 2. Nodes vs. proportional path length

M	1	2	4	8	16	32	64	128	256	512
Short path	66	49	31	27	23	29	9	6	3	0
Long path	33	25	17	9	4	0	0	0	0	0

Our results are comparable to those proposed by other models considered significant. For instance, the performance is as a "large step Markov chain" algorithm [30]. This algorithm is based on a simulated annealing mechanism that uses an improvement heuristic, and the heuristic is very similar to ours (the only difference is that it did not consider 3-paths) and a

mutation procedure called double-bridge. The double-bridge mutation has the smallest change (4-paths) that could not be reverted in one step by 3-paths. A comparison of our results with the results obtained for symmetric TSP [31] is different since it used a local search algorithm, "Lin-Kernigham", which is a heuristic based on a segment-tree data structure [32] that gives better results and faster than 3-paths procedure. It will be the subject work of the future to add such a procedure with meta-learning [33] to the proposed learning algorithm.

7. Conclusions

Optimizations in neural network algorithms tend to produce a mechanism to reduce the losses and provide the most accurate results possible. Improving the solutions constructed by nodes is the main challenge in neural network algorithms. It is a key to increasing the speed and efficiency of machine learning problems. We showed even with several different types of rogue influenced, the observers remain very steady in their convergence toward the instructors. After temporary stacks, they were able to get right back on track with the little overall effect. The rogue influences had little effect on all convergence rates. With a greater sustained rogue presence, the networks would jolt substantially, but without persistence, the instructors would pull the nodes into convergence. The noisy influences of rogues might be helpful in some contexts. It is possible when lacking a good training function, the disruptions might be helpful from time to time. Certainly, diversification and mutation have always contributed to success in the long run. It is also possible that with more difficult solutions, there will appear more chances to fall into a local solution, but in the case of non-optimal solutions, perturbations might allow networks to escape from the local solution. Some inherent advantages are: (i) computation avoids premature convergence, (ii) the positive feedback mechanism facilitates the rapid discovery of optimal solution, (iii) and the greedy heuristic helps to find acceptable solutions in the early stages of the search process. Disadvantages in such a learning system are: (i) slower convergence than other heuristics, (ii) no centralized processor to guide the system towards a perfect solution, and (iii) the process was performed poorly when used with a larger group, e.g., "75". However, the right optimization algorithm could reduce the amount of training exponentially.

Acknowledgments

This research does not receive a grant from any funding agency in the public, commercial, or not-for-profit sectors.

Disclosure statement

The author declares that there is no conflict of interest regarding the publication of this article.

References

[1] H., Ackley and L., Littman, "Generalization and scaling in reinforcement learning," In Touretzky, D. S. (Ed.), Advances in Neural Information Processing Systems 2, San Mateo, CA. Morgan Kaufmann, pp.550-557, **(1990)**

[2] L. jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," arXiv:1902.06162, **(2019)**

[3] Z. Ghahramani, "Unsupervised learning," In: Bousquet O., Von Luxburg U., Rätsch G. (eds) Advanced Lectures on Machine Learning. ML 2003. Lecture Notes in Computer Science, vol.3176. Springer, Berlin, Heidelberg. **(2004)** DOI: 10.1007/978-3-540-28650-9_5
[4] D. Helmbold and R. Schapire, "Predicting nearly as well as the best pruning of a decision tree. Machine learning 27," pp51-68, **(1997)**, DOI: 10.1023/A:1007396710653
[5] J. Kennedy and R. Eberhart, "Swarm intelligence," A book, Morgan Kaufmann Publishers Inc. 340 Pine Street, Sixth Floor San Francisco, CA, United States, ISBN:978-1-55860-595-4, **(2001)**
[6] L. Ke and M. Jitendra, "Learning to optimize," arXiv:1606.01885, **(2016)**
[7] L. Ke and M. Jitendra, "Learning to optimize neural nets," arXiv:1703.00441v2, **(2017)**
[8] H. Yao, W. Xian, Z. Tao, Y. Li, B. Ding, R. Li, and Z. Li, "Automated relational meta-learning," in Proceedings of ICLR 2020 Conference, **(2020)**
[9] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," arXiv:2004.05439, **(2020)**
[10] M. Andrychowicz, M. Denil, S. Gomez, M. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. Freitas, "Learning to learn by gradient descent," Neural and Evolutionary Computing, arXiv preprint arXiv:1606.04474, **(2016)**
[11] K. Li and J. Malik, "Learning to optimize," CoRR,abs/1606.01885, **(2016)**
[12] D. Kingma and A. Jimmy, "A method for stochastic optimization," arXiv preprintarXiv:1412.6980, **(2014)**
[13] J. Duchi, E. Hazan, and Y. Singer, "Adaptive sub-gradient methods for online learning and stochastic optimization," Journal of Machine Learning Research, vol.12(Jul) pp.2121-2159, **(2011)**
[14] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," COURSERA: Neural Networks for Machine Learning, vol.4, no.2, **(2012)**
[15] E. Bonabeau, M. Doringo, and G. Theraulaz, "Swarm intelligence: From natural to artificial systems," New York: Oxford University Press, **(1999)**
[16] R. Caruana, D. Silver, J. Baxter, T. Mitchell, L. Pratt, and S. Thrun. "Learning to learn: Knowledge consolidation and transfer in inductive systems," NIPS workshop on Learning to Learn: Knowledge Consolidation and Transfer in Inductive Systems, **(1995)**
[17] S. Thrun and L. Pratt, "Learning to learn," Springer Science & Business Media, **(2012)**
[18] P. Brazdil, G. Carrier, C. Soares, and R. Vilalta, "Meta learning: Applications to data mining," Springer Science and Business Media, Springer Verlag, Berlin Heidelberg, pp.17-42, **(2008)**
[19] C. Blum and A. Roli, "Meta heuristics in combinatorial optimization: Overview and conceptual comparison," ACM Computing Surveys (CSUR), vol.35, no.3, pp.268-308, **(2003)**
[20] J. Schmidhuber, "Optimal ordered problem solver," Machine Learning, vol.54, no.3, pp.211-254, **(2004)**
[21] S. Hochreiter, S. Younger, and P. Conwell, "Learning to learn using gradient descent," in International Conference on Artificial Neural Networks, pp.87-94, Springer, **(2001)**
[22] K. Kishor and P. Suresh, "GCS technique to improve the performance of neural networks," Journal of Intelligent Systems, vol.29, no.1, DOI: 10.1515/jisys-2017-0423, **(2019)**
[23] J. Travis, T. Desell, S. Clachar, J. Higgins, and B. Wild, "Evolving deep recurrent neural networks using ant colony optimization," 15th European Conference on Evolutionary Computation in Combinatorial Optimization, pp.235, **(2015)**, DOI: 10.1007/978-3-319-16468-7_8
[24] H. Bottee and E. Bonabeau, "Evolving ant colony optimization," Advanced Complex System., vol.1, no.2/3, pp.149-159, **(1998)**
[25] S. Haykin, "Neural networks: A comprehensive foundation," Prentice Hall PTR, Upper Saddle River, New Jersey, United States, **(1998)**, ISBN: 978-0-13-273350-2
[26] G. Kevin, "An introduction to neural networks," CRC Press, Computers pp.234, **(1997)**
[27] C. Tianfeng and R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)," Geoscientific Model Development Discussions, vol.7, no.1, **(2014)**, DOI: 10.5194/gmdd-7-1525-2014

[28] M. Dorigo and M. Gambardella, “Ant colony system: A cooperative learning approach to the traveling salesman problem,” IEEE Transactions on Evolutionary Computation, vol.1, no.1, pp.53-66, **(1997)**, DOI: 10.1109/4235.585892

[29] G. Di Caro and M. Dorigo, “AntNet: A mobile agent’s approach to adaptive routing,” Technical report IRIDIA/97-12, IRIDIA, Universite´ Libre de Bruxelles, Brussels, **(1997)**

[30] O. Martin, S. Otto, and E. Felten, “Large-step Markov chains for the TSP incorporating local search heuristics,” Operations Research Letters, vol.11, pp.219-224, **(1992)**

[31] D. Johnson and L. McGeoch, “The travelling salesman problem: A case study in local optimization,” in Local Search in Combinatorial Optimization, New York: Wiley and Sons, **(1997)**

[32] M. Fredman, D. Johnson, L. McGeoch, and G. Ostheimer, “Data structures for traveling salesmen,” Journal of Algorithms, vol.18, pp.432-479, **(1995)**

[33] R. Vilalta and Y. Drissi, “A perspective view and survey of meta-learning,” Artificial Intelligence Review, vol.18, no.2, pp77-95, **(2002)**