2019

Computational Modeling of Trust Factors Using Reinforcement Learning

C. M. Kuzio
Old Dominion University, ckuzi001@odu.edu

A. Dinh
Old Dominion University

C. Stone
Old Dominion University

L. Vidyaratne
Old Dominion University, lvidyara@odu.edu

K. M. Iftekharuddin
Old Dominion University, kiftekha@odu.edu

Original Publication Citation

C. M. Kuzio, A. Dinh, C. Stone, L. Vidyaratne, and K. M. Iftekharuddin "Computational modeling of trust factors using reinforcement learning", Proc. SPIE 11136, Optics and Photonics for Information Processing XIII, 111360J (6 September 2019); https://doi.org/10.1117/12.2531060.

Computational modeling of trust factors using reinforcement learning

C. M. Kuzio, A. Dinh, C. Stone, L. Vidyaratne, K. M. Iftekharuddin

C. M. Kuzio, A. Dinh, C. Stone, L. Vidyaratne, K. M. Iftekharuddin, "Computational modeling of trust factors using reinforcement learning," Proc. SPIE 11136, Optics and Photonics for Information Processing XIII, 111360J (6 September 2019); doi: 10.1117/12.2531060

SPIE.

Event: SPIE Optical Engineering + Applications, 2019, San Diego, California, United States

Computational Modeling of Trust Factors Using Reinforcement Learning

C. M. Kuzio[a], A. Dinh[a], C. Stone[a], L. Vidyaratne[a], and K. M. Iftekharuddin[a]

[a]Vision Lab in Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, VA 23529

ABSTRACT

As machine-learning algorithms continue to expand their scope and approach more ambiguous goals, they may be required to make decisions based on data that is often incomplete, imprecise, and uncertain. The capabilities of these models must, in turn, evolve to meet the increasingly complex challenges associated with the deployment and integration of intelligent systems into modern society. Historical variability in the performance of traditional machine-learning models in dynamic environments leads to ambiguity of trust in decisions made by such algorithms. Consequently, the objective of this work is to develop a novel computational model that effectively quantifies the reliability of autonomous decision-making algorithms. The approach relies on the implementation of a neural network based reinforcement learning paradigm known as adaptive critic design to model an adaptive decision making process that is regulated by a quantitative measure of risk associated with each possible decision. Specifically, this work expands on the risk-directed exploration strategies of reinforcement learning to obtain quantitative risk factors for an automated object recognition process in the presence of imprecise data. Accordingly, this work addresses the challenge of automated risk quantification based on the confidence of the decision model and the nature of given data. Additionally, further analysis into risk directed policy development for improved object recognition is presented.

Keywords: Risk Factorization in Machine Learning, Trust in Decision Models, Decision Making under imprecise data, Reinforcement Learning, Adaptive-Critic Design, Object Recognition, Deep Learning

1. INTRODUCTION

Autonomous systems have seen widespread adoption in fields such as navigation,[1,2] data classification, and predictive modeling.[3,4] As these systems grow in scope, popularity and complexity, the notion of reliability or 'trust' in these systems has become of increasing importance. This is especially true when these systems utilize some form of machine learning since models in machine learning are often not easily interpretable by humans.

Prior attempts to quantify trust have been seen in the contexts of online marketplaces,[5,6] search engine optimization,[7] social networking[8,9] and peer-to-peer networks.[10,11] For example, statistical models similar to that of Rettinger, Nickles, and Tresp[12] quantify the trustworthiness of an online marketplace offer from seller to buyer based on factors such as item condition, account feedback, account age, shipping rates, etc. Using a combination of reinforcement learning and hidden markov models (HMM), the authors of Ref. 12 are able to calculate a trust factor in dynamic environments. Another example would be the framework of Zhan and Fang,[9] wherein the trust value for an entity is based on the outcomes of past interactions with the entity. More specifically, they classify information as either reliable, neutral, or unreliable. They then calculate their trust value for a piece of information based on the proportion of similar past classifications being reliable.[9] The authors of Ref. 9 also discuss how this framework complements the common use of naive Bayes classifiers in electronic mail filters. Other examples of trust in a machine learning context, especially that of reinforcement learning, can be seen in Ref. 6 and Ref. 13. In the study by Law,[13] the author discusses a generalized, model-based method of computing the risk for the action of an agent and performing risk-directed exploration. It should be noted that in this paper, we consider trust to be the inverse of risk. Thus, we consider methods of calculating risk to be equivalent to computing trust. In the conventional sense of the word, 'trust' is a subjective concept formed by experience. In an analogous manner, many of the trust-quantification methods in the previously mentioned approaches attempt to replicate this conventional definition by basing the trust-quantification on not only the object to be trusted, but also prior interactions and the environment the object is situated in.

Optics and Photonics for Information Processing XIII, edited by Khan M. Iftekharuddin, Abdul A.S. Awwal, Victor H. Diaz-Ramirez, Andrés Márquez, Proc. of SPIE Vol. 11136 111360J · © 2019 SPIE · CCC code: 0277-786X/19/$21 · doi: 10.1117/12.2531060

Actor-critic methods are a form of temporal-distance learning where a critic network evaluates an action generated by the actor network in terms of an estimated future cost-to-go. Trust quantification using actor-critic methods have been explored previously by Yu et al.[6] to quantify the trustworthiness of (simulated) online testimonies. As mentioned in Ref. 6, this feedback from critic to actor is well-suited for supplementing existing trust-quantification frameworks.

Our proposed approach combines an actor-critic model with a risk-adjusted policy with the ultimate goal of improving classification performance based on the generated risk-values. Through repeated interactions with the environment, our system is able to intelligently learn the features that indicate an input is likely to be misclassified, quantify a corresponding risk value, and construct an exploration policy that minimizes this risk. More specifically, we utilize an actor network in conjunction with a risk-adjusted policy whose parameters are learned on-line in order to identify and navigate the environment with the goal of minimizing risk. We also utilize a critic network whose objective is to evaluate the quality of the actor's decision in terms of a cumulative future expected reward and provide feedback to the actor network. The remainder of this study is organized as follows. Section 2 discusses the framework of our actor-critic system and risk-based policy. Section 3 presents and discusses the results. Section 4 presents our conclusions.

2. METHODOLOGY

The general structure of our proposed risk-based actor-critic system can be seen in Fig. 1. The main components of this system are the actor network, the critic network, and the risk-adjusted policy. The role of the plant in this setup is merely to provide an input or state $x(t)$ to both the actor and critic at each time step t. The $x(t)$ selected by the plant is directly determined by the risk-adjusted policy.

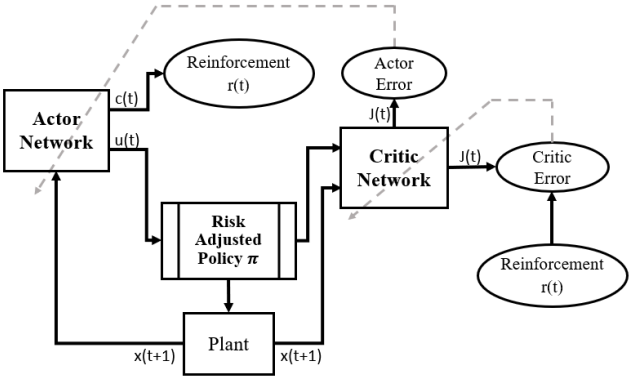

Figure 1: Actor-critic system structure

2.1 Actor Network

Given an input $x(t)$, the function of the actor network is two-fold. Firstly, the actor is to classify the given input, as represented by the output $c(t)$. A reinforcement signal $0 \leq r(t) \leq 1$ is then calculated based on the output c(t). In the training phase, the normalized mean-squared error of c(t) is used as r(t). In the testing phase, $r(t)$ is determined as follows:

$$r(t) = \max(1 - c(t)). \tag{1}$$

The second function of the actor is to output a set of actions u(t) that should indirectly minimize the cumulative future reinforcement R(t) given by:

$$R(t) = r(t+1) + \alpha r(t+2) + \cdots \tag{2}$$

where α is a constant between 0 and 1.

To calculate $c(t)$ and $u(t)$, we use a two-branch structure for the actor as seen in Fig. 2. We denote the set of feedforward layers used to calculate $c\,(t)$ as decision branch and the set of feedforward layers used to calculate $u(t)$ as the action branch. Note that while the feed-forward layers within the action branch are initialized with random weights, the decision branch along with the convolutional layers will utilize weights pretrained for classification.

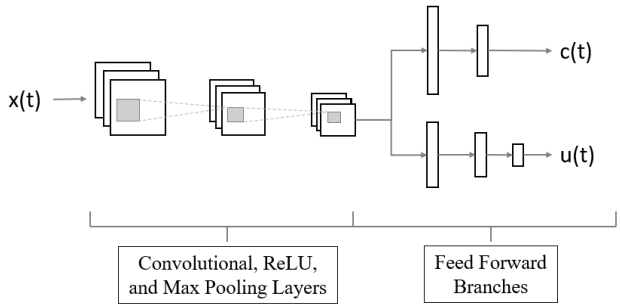

Figure 2: Proposed architecture of the actor network

To generate the weights for the convolutional layers, we first construct a convolutional neural network for classification consisting of 5 convolutional layers each followed by a max pooling layer. After the final convolutional layer is a single feedforward classification layer. Note that after each convolutional layer, a rectified linear unit (ReLU) activation function is applied followed by a batch normalization layer. Furthermore, the classification layer utilizes a softmax activation function. A cross-entropy loss function with a stochastic gradient descent (SGD) optimization scheme is used to train the network. Once adequately pretrained, the feedforward layer is removed and the remaining layers with the pretrained weights are used to construct the actor and critic networks. By making use of the features learned by the pretraining process of the CNN, we are able to reduce the time and computational resources required for training of the actor-critic network. This also serves to minimize the instability associated with typical actor-critic models.

The loss function to be minimized in the decision branch is simply the mean-squared error of $c(t)$ with respect to the correct classification. The loss function to be minimized in the action branch is:

$$E_a\,(t) = \frac{1}{2} e_a^2\,(t) = \frac{1}{2}\left(J\,(t) - U_c\,(t)\right)^2. \tag{3}$$

where $J\,(t)$ is the output of the critic network, which will be elaborated on in Sec. 2.2. U_c is the desired ultimate objective, as defined in Ref. 14. Note that U_c will be set to 0 for the remainder of this paper. Also note that when calculating the loss defined in Eq. (2) and performing backpropogation, only the weights in the action branch will be adjusted. This effectively preserves the learned features of the CNN at the pretraining step by inhibiting the abrupt gradient changes associated with initial exploration steps in the actor-critic training scheme.

For the action branch, the output layer utilizes the following activation function, following the example of Ref. 14:

$$u(t) = \frac{1 - e^{-g(t)}}{1 + e^{-g(t)}}. \tag{4}$$

where $g(t)$ denotes a weighted sum of inputs from the previous layer. The remaining feedforward layers in the action branch utilize leaky rectified linear units (Leaky ReLU) as activation functions. For the decision branch, the output layer utilizes a linear activation function while the remaining feedforward layer utilizes a Leaky ReLU activation function.

2.2 Critic Network

Similar to the actor network, the critic uses pretrained convolutional layers in conjunction with a set of randomly initialized feedforward layers, as seen in Fig. 3. For all layers within the critic network but the output layer, a Leaky ReLU is used as the activation function. For the critic's output layer, no activation function is used. The loss function to be minimized in the critic is defined as in Ref. 14:

$$E_c\left(t\right) = \frac{1}{2} e_c^2\left(t\right) \tag{5}$$

where the critic error $e_c(t)$ is given by:

$$e_c\left(t\right) = \alpha J\left(t\right) - \left[J\left(t-1\right) - r\left(t\right)\right]. \tag{6}$$

and where α is a constant between 0 and 1. Note that this implies the critic output $J\left(t\right)$ is an estimation of the total, discounted cost-to-go. As in the case of the action branch of the actor, the weights of the convolutional layers are not adjusted during the actor-critic training step.

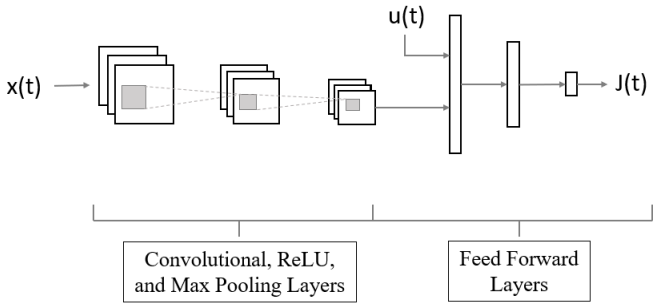

Figure 3: Proposed architecture of the critic network

2.3 Risk-Adjusted Policy

In our system, the ultimate goal of the risk-adjusted policy is to navigate the state-space in such a way that balances a 'risk' measure with an estimated future reward. More specifically, at each value of t, the policy should select the action that causes the plant to transition to the state which will result in the best outcome in terms of the immediate and future rewards. In our case, the immediate reward is given by $r\left(t\right)$ and the future reward is given by the critic output $J(t)$. The actions in this case are represented by the vector of actor outputs $u\left(t\right)$. Each individual action u_i represents the transition to a different state within the state-space.

To calculate the risk-adjusted policy given t and a state-space S with n unique states, we utilize the following equations adapted from the work in Ref. 13:

$$\text{Risk}\left(s, u_i\right) = \lambda\left(t\right) H_{u_i}\left(s\right) - \left(1 - \lambda\left(t\right)\right) E[R_{ss'}], \tag{7}$$

$$H_{u_i}\left(s\right) = -P_{ss'}^{u_i} \log\left(P_{ss'}^{u_i}\right), \tag{8}$$

$$E\left[R_{ss'}\right] = \sum_{s'} P_{ss'}^{u_i} R_{ss'}, \tag{9}$$

$$U_r(s, u_i) = p(t)\left(1 - \text{Risk}(s, u_i)\right) + \left(1 - p(t)\right) J(s, u(t)), \tag{10}$$

and

$$\pi\left(s, u_i\right) = \frac{e^{\frac{U_r\left(s, u_i\right)}{\tau}}}{\sum_{j=1}^{n} e^{\frac{U_r\left(s, u_j\right)}{\tau}}} \tag{11}$$

where τ is a constant and $\lambda(t)$, $p(t)$ are bounded between 0 and 1. Note that $P_{ss'}^{u_i}$ and $R_{ss'}$ are vectors representing the probability and reward of transitioning from the current state s to each possible state s' given that the action u_i is taken. Also note that $J(s, u(t))$ here denotes the value of critic output $J(t)$ should the input to the critic network be s and $u(t)$.

In particular, $R_{ss'}^{u_i}$ is given by the value of $1 - r(t)$ should the current input $x(t)$ be s'. For example, suppose each of the states in S are indexed from $k = 1$ to $k = n$. Then the k-th entry of $R_{ss'}$ would equal the value of $1 - r(t)$ should $x(t) = s_k$.

Similarly, the k-th entry of $P_{ss'}^{u_i}$ represents the probability of transitioning from state s to s_k given that action u_i is taken. For our system, we define initial values for $P_{ss'}^{u_i}$ and then modify the values of $P_{ss'}^{u_i}$ according to the sign and magnitude of u_i. More specifically, suppose the action u_k represents the decision to transition from state $s = s_{k_0}$ to state s_k. Then $P_{ss'}^{u_k}$ would be defined as follows:

- If $u_i \leq 0$, define the probability p_{k_0} of transitioning from state s to s_{k_0} (i.e. staying in the same state) as $p_{k_0} = \beta(t) * p_0 * |u_i|$. Otherwise, if $u_i > 0$, define p_{k_0} as:

$$p_{k_0} = \beta(t) * (p_0 + (1 - p_0) * |u_i|).$$

- Define the probability p_k of transitioning from state s to s_k as $p_k = 1 - p_{k_0}$.
- All other entries are initialized as 0.

In essence, a positive value of u_i indicates the actor's aversion to moving away from state s and vice-versa for the negative case. Note that p_0 is a constant between 0 and 1. Additionally, to urge early exploration and allow the actor network time to learn, we define the parameter $\beta(t)$ as:

$$\beta(t) = \frac{1}{k_\beta} \min\left(\frac{t}{\min(t_{\max}, C_1)}, 1\right) \tag{12}$$

where t_{max} is the maximum value of t and $C_1 \in \mathcal{N}$ is a constant. For the same reasons, we define $\lambda(t)$ and $p(t)$ as:

$$\lambda(t) = \frac{1}{k_\lambda} \min\left(\frac{t}{\min(t_{\max}, C_1)}, 1\right), \tag{13}$$

and

$$p(t) = 1 - \frac{1}{k_p} \min\left(\frac{t}{\min(t_{\max}, C_1)}, 1\right). \tag{14}$$

3. RESULTS AND DISCUSSION

In this section, we present the performance of the actor-critic system on various subsets of the custom ship dataset selected via a risk-based threshold. Unless noted otherwise, the parameters used for our actor-critic system are as follows. Both the actor and critic are initialized with 5 convolutional layers with rectified linear units (ReLU) followed by a number of feedforward layers. Other hyper-parameters for the system are configured as follows: In the case of the actor's classification branch, two feedforward layers with 2048 / 5 nodes are used and in the actor's action branch, three feedforward layers with 2048 / 800 / 5 node configuration are used. Similarly, for the critic, three feedforward layers with 2053 / 800 / 1 node are used. The actor's learning rate l_a is set to 0.01 whereas the critic's learning rate l_c is set to 0.001. For α, a value of 0.95 is used. For the risk-directed exploration, we assign $C_1 = 100$, $p_0 = 0.6$, $k_\beta = 5$, $k_\lambda = 5$, $k_p = 3$ with $t_{max} = 400$.

Our dataset for this study consists of a total of 3800 images of ships, split between 5 classes and sourced from shipspotting.com.[15] The 5 classes used are aircraft carriers, cargo ships, cruise ships, rescue vessels, and

Figure 4: Example dataset images. Image classes from top-left to bottom-right: work boat, work boat, rescue vessel, cruise ship, cargo ship, aircraft carrier

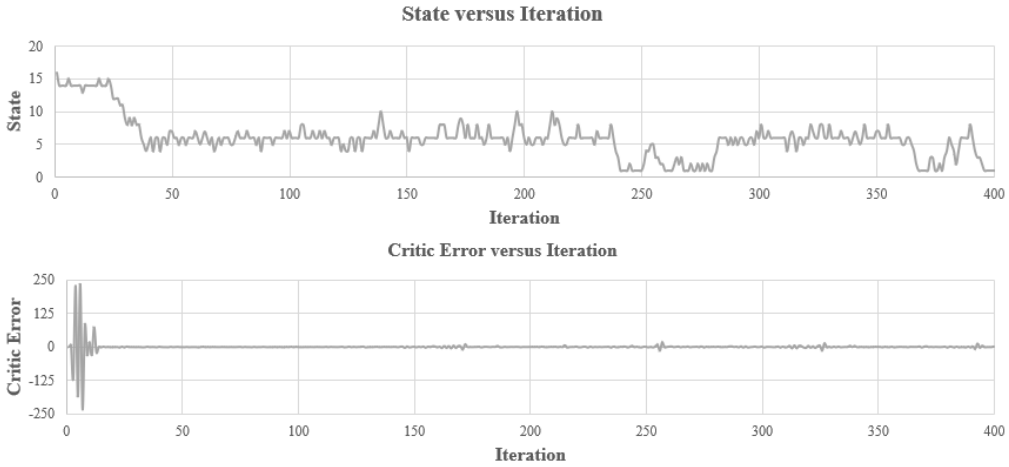

Figure 5: Example sequence run, top: system state versus iteration, bottom: critic error versus iteration

work boats. All images are converted to grey-scale, resized, and cropped to 128 by 128 pixels as seen in Fig. 4. Note that only images where the entirety of the starboard or port side are visible are selected for the analysis. Images taken from onboard or within the ship and images of mainly the bow or stern are discarded due to the apparent lack of vital information for the recognition task. Within each class, the images are further sub-divided randomly into sequences of 20 for ease of computation.

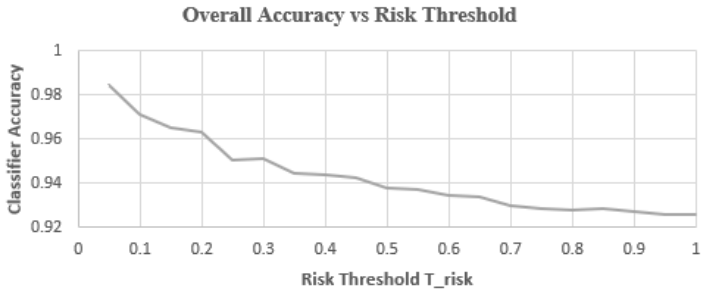

Figure 6: Classifier accuracy on images filtered by risk.

The performance of the system is analyzed using a 10 fold cross-validation procedure. For each fold, a classifier network is pretrained on the training set and used to initialize part of the actor and critic weights. During the training phase, the system is allowed to explore each sequence in the training set for t_{max} iterations. A typical

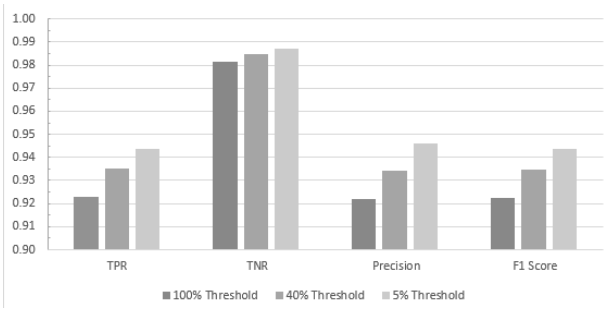

Figure 7: TPR, TNR, precision, and F_1 score at $T_{risk} = 100\%$, $T_{risk} = 40\%$, and $T_{risk} = 5\%$

100% Threshold		Actual Class				
		Class 1	Class 2	Class 3	Class 4	Class 5
Predicted Class	Class 1	618	9	13	35	18
	Class 2	14	657	3	7	6
	Class 3	13	0	934	0	4
	Class 4	22	13	3	589	52
	Class 5	13	1	7	49	720

40% Threshold		Actual Class				
		Class 1	Class 2	Class 3	Class 4	Class 5
Predicted Class	Class 1	256	5	5	10	8
	Class 2	4	261	0	2	2
	Class 3	2	0	378	0	0
	Class 4	6	6	0	242	18
	Class 5	4	0	1	18	292

Figure 8: Confusion matrices, left: $T_{\text{risk}} = 100\%$, right: $T_{\text{risk}} = 40\%$

run of the actor-critic system for a single sequence is shown in Fig. 5. Note that although the system tends to hover around a specific state as in Fig. 5, it is not uncommon for the system to oscillate between multiple states that are identified to have low-risk. After exploring each sequence in the training set, the system is then allowed to explore each of the remaining sequences in the test set for t_{max} iterations. After the exploration concludes, the risk values generated by Eq. (6) are recorded and averaged for each state. Then a percentage of images with the lowest average risk are sampled from each sequence and the performance of the classifier is evaluated on this set of samples. We will refer to this percentage as the risk-based threshold T_{risk}. The accuracy of the classifier for this set is also calculated and averaged over each fold, as in Fig. 6. We also present the true positive rate (TPR), true negative rate (TNR), precision and F_1 score of the system. These values are averaged over each class, for various risk-based thresholds as seen in Fig. 7. The corresponding confusion matrix is shown in Fig. 8. Note that we use the following definitions for TPR, TNR, precision, and F_1 score:

$$\text{TPR} = \frac{\sum \text{ True Positives}}{\sum \text{ True Positives} + \sum \text{ False Negatives}},$$

$$\text{TNR} = \frac{\sum \text{ True Negatives}}{\sum \text{ True Negatives} + \sum \text{ False Positives}},$$

$$\text{Precision} = \frac{\sum \text{ True Positives}}{\sum \text{ Positive Predictions}},$$

and

$$\text{F}_1 \text{ Score} = 2 \times \frac{\text{Precision} \times \text{TPR}}{\text{Precision} + \text{TPR}}.$$

The average accuracy of the classifier on the entirety of each testing fold is found to be 92.58%. By using our actor-critic system to filter out images deemed higher risk for the task, the average classifier accuracy is effectively increased to 98.42% (at $T_{\text{risk}} = 5\%$). Furthermore, the system is also shown to improve the classifier's

TPR, TNR, precision and F1 score, as shown in Fig. 7. At a $T_{\text{risk}} = 5\%$, the system shows a 2.3% increase in the TPR, TNR, and F_1 score each and at $T_{\text{risk}} = 40\%$, a 1.3% increase. In the case of the TNR, the system shows a marginal improvement of less than 0.5% as T_{risk} decreases.

4. CONCLUSION AND FUTURE WORK

We propose an actor-critic model with a risk-adjusted policy to quantify risk in a complex recognition task, and complement a classifier's performance. Our experimental results show an all-around performance improvement in terms of accuracy, TPR, TNR, precision and F_1 score of the classifier.

Possible future work would include research into reduction in the number of experimentally derived parameters such as k_β, k_λ, and k_p, as used in Eqs. (12), (13), and (14). These parameters are pre-adjusted for the system's tolerance for high-risk actions and serve to adjust the degree of control that the actor's action branch has over the state transitions. It may be beneficial to obtain an online learning scheme for these parameters alongside the actor critic training setup. Although the structure of our actor and critic is specific to the classifier used, the concept of adding feedforward layers to create an action branch / critic branch to the network may easily be adapted for other classifiers. One future avenue of investigation would be to evaluate how a similar setup performs when the architecture of the classifier is altered. Another constraint of the current approach is the limited number of actions the system may take, regardless of the size of the state space. This implies that as the number of possible states grows in number, the number of iterations required for the system to satisfactorily explore the state-space also grows, increasing the computational cost. Therefore, we also propose further research into augmenting the action space of the system to improve the overall efficiency and convergence.

ACKNOWLEDGMENTS

This work would not have been possible without the financial support of the Office of Naval Research (ONR) under Grant N00014-18-1-2682.

REFERENCES

[1] Bonin-Font, F., Ortiz, A., and Oliver, G., "Visual navigation for mobile robots: A survey," *Journal of intelligent and robotic systems* **53**(3), 263–296 (2008).

[2] Chao, H., Cao, Y., and Chen, Y., "Autopilots for small unmanned aerial vehicles: a survey," *International Journal of Control, Automation and Systems* **8**(1), 36–44 (2010).

[3] Agrawal, S. and Agrawal, J., "Neural network techniques for cancer prediction: A survey," *Procedia Computer Science* **60**, 769–774 (2015).

[4] Vijiyarani, S. and Sudha, S., "Disease prediction in data mining technique–a survey," *International Journal of Computer Applications & Information Technology* **2**(1), 17–21 (2013).

[5] Rettinger, A., Nickles, M., and Tresp, V., "A statistical relational model for trust learning," in [*Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*], 763–770, International Foundation for Autonomous Agents and Multiagent Systems (2008).

[6] Yu, H., Shen, Z., Miao, C., An, B., and Leung, C., "Filtering trust opinions through reinforcement learning," *Decision Support Systems* **66**, 102–113 (2014).

[7] Jøsang, A., Ismail, R., and Boyd, C., "A survey of trust and reputation systems for online service provision," *Decision support systems* **43**(2), 618–644 (2007).

[8] Kuter, U. and Golbeck, J., "Sunny: A new algorithm for trust inference in social networks using probabilistic confidence models," in [*AAAI*], **7**, 1377–1382 (2007).

[9] Zhan, J. and Fang, X., "A computational trust framework for social computing (a position paper for panel discussion on social computing foundations)," in [*2010 IEEE Second International Conference on Social Computing*], 264–269, IEEE (2010).

[10] Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H., "The eigentrust algorithm for reputation management in p2p networks," in [*Proceedings of the 12th international conference on World Wide Web*], 640–651, ACM (2003).

[11] Zhou, R., Hwang, K., and Cai, M., "Gossiptrust for fast reputation aggregation in peer-to-peer networks," *IEEE Transactions on Knowledge and Data Engineering* **20**(9), 1282–1295 (2008).
[12] Moe, M. E. G., Tavakolifard, M., and Knapskog, S. J., "Learning trust in dynamic multiagent environments using hmms," in [*Proceedings of the 13th Nordic Workshop on Secure IT Systems (NordSec 2008)*], (2008).
[13] Law, E. L., Coggan, M., Precup, D., and Ratitch, B., "Risk-directed exploration in reinforcement learning," *Planning and Learning in A Priori Unknown or Dynamic Domains* , 97 (2005).
[14] Si, J. and Wang, Y.-T., "Online learning control by association and reinforcement," *IEEE Transactions on Neural networks* **12**(2), 264–276 (2001).
[15] "Ship photos and ship tracker." http://www.shipspotting.com/. Accessed: 2018-05-29.