CS 229, Fall 2019 Final Report

William Chong (wmchong@stanford.edu)

Introduction

Multi-agent formation control laws typically depend on the graph state of the robot formation to calculate control inputs for each robot in a decentralized method to reach a goal formation [6]. If malicious agents are present, then those actors can impede the formation control. This activity is called "spoofing" the network, and occurs in many instances of decentralized control of multiple nodes such as in formation control and in other contexts such as network transactions. It is important to classify these malicious agents to mitigate spoofing in order to ensure that the network behaves properly and is robust against these spoofing attacks. Otherwise, the system will be dysfunctional and most likely fail.

This work attempts to train a malicious agent classifier in robot formation control that has no knowledge of the system dynamics, control law, and connectivity of the graph. Feature inputs to the network consist of robot positions, velocities, and position FFT data along their trajectories. Three networks are designed and implemented (LSTM, CNN, baseline MLP), and evaluated against each other. The general network design is two layers where the first layer is a parallel structure (depending on the architecture) of 2 LSTM/CNN/MLP (position, velocity time series) and 1 MLP (position FFT), and the concatenated outputs are fed into an MLP for the second layer. The output is a 10x1 binary vector indicating whether each robot is malicious/not malicious (0/1). Results show high precision, recall, and balanced accuracy with a slightly worse specificity across all networks, with the CNN network slightly outperforming the rest.

Related Work

Similar work involve detection through robust control law strategies and designing outlier detectors. An example of the first approach is [1], where the author develops control laws with time-varying weights to reduce the influence of malicious agents to a certain epsilon formation convergence. Examples of the second approach are [2][3], where fault detection in the multi-robot network is performed with outlier detection from state estimators or neural networks. The first class of approaches is excellent if the state dynamics are known and the user has full control over the robots since the malicious agent classification through the weighting dynamics is performed through active exploration with control. The second class of approaches is excellent if the state and observation dynamics are known, and a probabilistic/learned outlier detector can be used. The main difference between this work and other works is that this work assumes no knowledge of the underlying state dynamics and control, and has no control authority over the system. Thus, this work assumes that the classifier only has 3rd party information from observing the trajectories. This scenario is quite common in cases where system dynamics are not provided, and the detector has to infer relevant features from just the trajectories.

In broader terms, this work falls under the category of time-series/sequence classification, where the objective is usually to assign the time series a certain class. In these works, the two most common techniques are using CNNs and LSTMs over the time sequence data. In the CNN approach, the spatial features are usually collapsed to the same dimension and the resulting sequence is fed directly into the CNN structure. An example approach is [4] where spatial-temporal information through recorded tactile feedback is classified, and a texture is assigned to the material. In the LSTM approach, the time sequence is directly inputted into the network, and the last output of the hidden state is used as a feature. An example approach is [5] where human-activity recognition classification is performed with measured body accelerations. Both works showed good classification metrics since CNNs and LSTMs work extremely well in extracting features from time sequence data due to how the networks extract features (explained in Methods section), and thus this work investigates both models separately. The general network structure used in this work follows the aforementioned works with some difference in data pre-processing and network parameters. In addition, this work adopts a parallel structure for the first layer, and feature concatenation into a second layer MLP for classification.

Dataset and Features

The generated dataset consist of trajectories (x,y positions) of 10 robot positions over 5 seconds sampled at 100 Hz with random initial seeds for each dataset. The robots were initially placed at a random position within a certain radius from each other, node indices were assigned to each robot, and a certain number of malicious agents were assigned (0-5). 6000 example trajectories were generated, which were split into the training, testing, and validation datasets (50-25-25 split). Each split dataset contains an equal number of trajectories for each different number of malicious agents (0-5).

All robots follow first order dynamics $\dot{p}_i = u_i : p \in \mathbf{R}^{20x1}$ (robot (x,y) positions). The discrete control law following [6] is $p_i(l+1) = (1-h)p_i(l) + \frac{h}{d_i}(\sum_{j \in N_i} a_{ij} p_j(l) + b_i) + wh; w \sim \mathcal{N}(0, Q)$. The malicious agents follow a Brownian motion with $u_i \sim \mathcal{N}(0.1, Q)$. The malicious agent strategy was chosen such that the process noise is the same as the good agents,

and the noise mean generated feasible trajectories similar to the good agents. This work assumes no coordination among malicious agents since neighbor communication among malicious agents is not guaranteed based on the initialized graph.

The features included the position trajectory, the velocity trajectory, and the position FFT. The position and velocity trajectories are inputs to the LSTMs/CNNs to learn temporal relationships for the robot trajectories. The position FFT are inputs to the MLP to extract frequency content correlation among good agents and among bad agents. Example trajectory and fft plots are provided. The effect of the malicious agents can be seen where the final formation is not attained. The fft plots show the relative difference in amplitude spectrum magnitude between the good and bad agents.

Pre-processing of the data only involved min/max scaling of the position and velocity trajectories such that the data is contained in the range [-1, 1]. The FFT data was obtained from calculating the position trajectory FFT and extracting the amplitude spectrum across frequency. Both operations were performed with scikit.

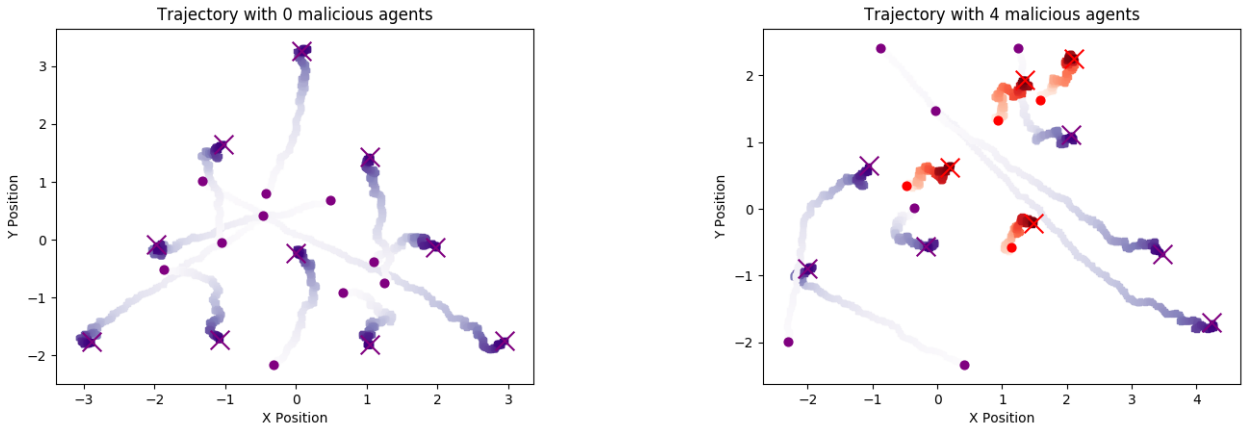

Figure 1: Sample trajectories with and without malicious agents (purple is good agent, red is malicious agent). Crosses indicate final positions, and dots indicate starting positions. The desired formation is a pyramid.

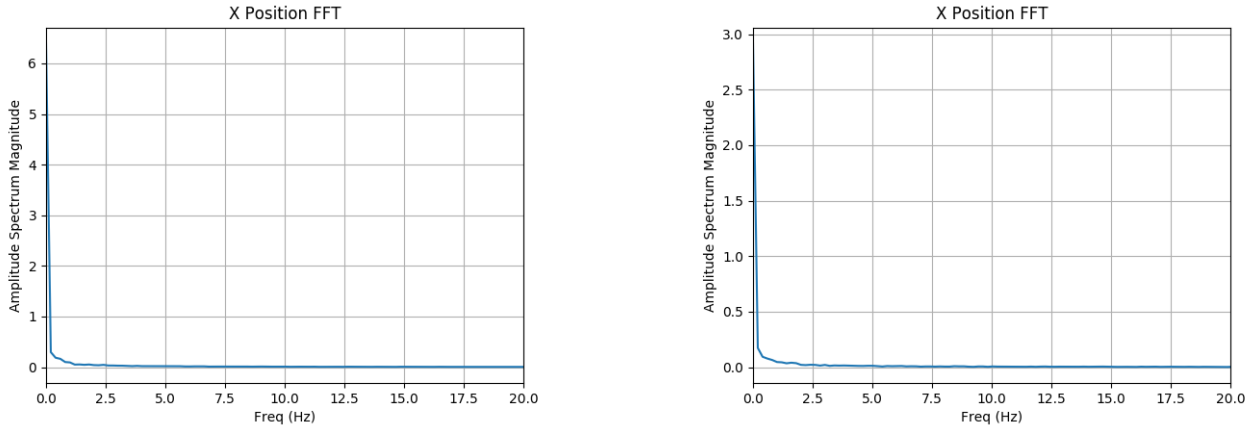

Figure 2: Sample FFT plots (left is good agent, right is bad agent).

Methods

The first method considered is using CNNs across the 2D spatial features (without concatenation) since the trajectory plots can be considered as images. However, this approach will not work well for several reasons. First, spatial correlation among robots are most likely not contained locally since robot neighbors may initially be far away from each other. Second, the position trajectory does not encode information about relative time history, and thus the time dependencies of the trajectories based on the control will potentially be lost. For example, parts of two trajectories may be close to each other on the trajectory plot, but actually very uncorrelated since they occur at completely different times.

The next approaches considered extracting features from the time series data directly. Two popular approaches to time-series classification are using CNNs and using LSTMs. For the CNN approach, the network performs convolution across the time history along with non-linear operations, which can extract temporal relationship features. For the LSTM approach, the time sequence is input directly into the network, and the LSTM cell structure (described below) allows information transfer utilizing states at different points in the sequence to extract relevant features.

The overall network was designed in two layers where the first layer consists of 3 parallel blocks that each take a separate feature. The first two parallel blocks are LSTM/CNN/MLP blocks that take the position and velocity trajectories. The third parallel block is an MLP with the position FFT feature. Feature selection was performed to finalize the full network. Figures 3-4 show the CNN and LSTM network structure (the baseline is not shown, but follows a similar structure to the LSTM but with MLP blocks instead). Batch norm regularization and dropout were added to reduce overfitting. The network parameters were refined with parameter iteration and trading off between complexity and performance.

The loss function used was the binary cross-entropy loss (specifically BCEWithLogitsLoss in Pytorch) since the task is binary classification. The loss function is $l_n = -p_c y_n log\sigma(x_n) - (1 - y_n)log(1 - \sigma(x_n))$ where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function to produce output probabilities in range [0,1]. x_n is the input feature to the sigmoid activation, and y_n is the true label. p_c is used to adjust the positive and negative class weighting for class imbalance in the training set.

Briefly describing CNN and LSTM structure in the following two paragraphs. CNNs consist of a convolutional layer, pooling layer, and a fully-connected layer in that sequence. The convolutional layer uses a sliding window filter that slides over the input volume and computes dot products between the entries of the filter and the input at any position. This sliding operation produces an activation map that gives the filter response at every position, which can be learned to activate when some feature is seen. Position in this case can be treated as a point in a sequence.

LSTMs are a form of recurrent neural networks which consist of a cell (acts as the memory part) and 3 gates (input, output, forget) that regulates information flow inside the LSTM unit. At a high level, the cell keeps track of the correspondences between elements in the input sequence, the input gate controls how much a new value flows into the cell, the forget gate controls how much a value remains in the cell, and the output gate controls how much the value is used to compute the output activation of the unit. The reader should reference [7] for specific implementation details.

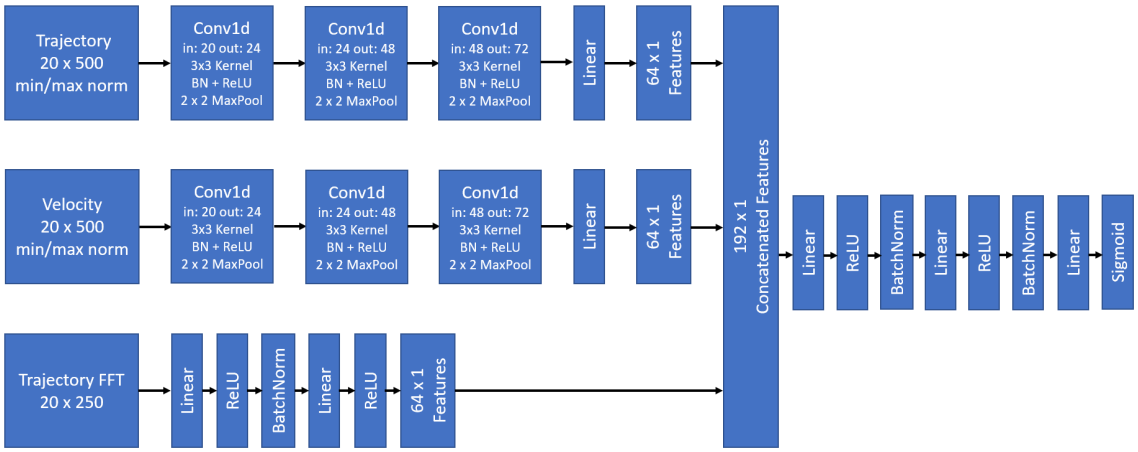

Figure 3: CNN network structure

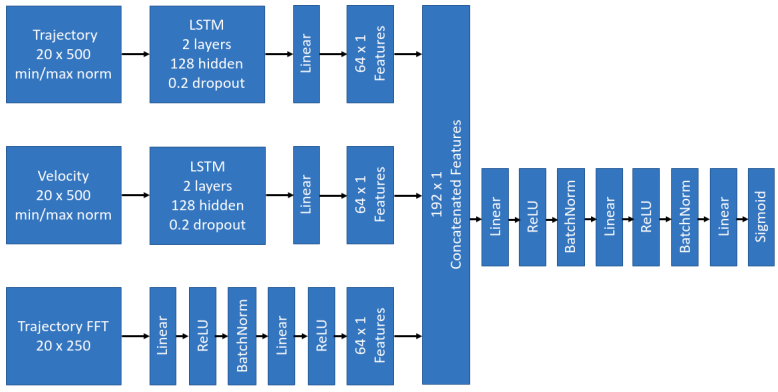

Figure 4: LSTM network structure

Experiments/Results/Discussion

Training parameters include mini-batch size of 100 examples and learning rate of 1e-3 (Adam). p_c is set to 0.333 for class imbalance weighting (average ratio of malicious to good agents). The optimizer and learning rate are standard and resulted in good training/testing curves. The mini-batch size was chosen from running training sessions with different batch sizes and selecting a number that provides fast convergence with not much overfit and good validation statistics.

The primary evaluation metrics are precision (ratio of true positives to all positive classifications), recall (true positive rate), specificity (true negative rate), and balanced accuracy ($0.5 * (recall + specificity)$) where balanced accuracy is used since the classes are imbalanced. The balanced accuracy normalizes true positive and true negative predictions by the number of positive and negative examples. The following table shows performance statistics on the validation dataset:

Network/Features	Precision	Recall	Specificity	Balanced Accuracy	Accuracy	Average Precision Score
Baseline	0.933	0.954	0.793	0.874	0.915	0.924
CNN, all	0.956	0.972	0.867	0.919	0.945	0.951
CNN, fft + pos	0.964	0.978	0.891	0.935	0.959	0.960
CNN, fft + vel	0.937	0.954	0.807	0.880	0.914	0.928
CNN, pos	0.948	0.961	0.842	0.901	0.933	0.940
CNN, vel	0.870	0.853	0.617	0.735	0.790	0.852
LSTM, all	0.960	0.961	0.878	0.920	0.938	0.951
LSTM, fft + pos	0.957	0.958	0.870	0.914	0.934	0.948
LSTM, fft + vel	0.851	0.895	0.530	0.713	0.805	0.840
LSTM, pos	0.880	0.919	0.625	0.772	0.845	0.870
LSTM, vel	0.826	0.926	0.415	0.670	0.800	0.820

From the above metrics, the best CNN architecture configuration was **CNN with FFT and position** input features, and the best LSTM architecture configuration was **LSTM with FFT, position, and velocity** input features. For both configurations, there is high precision and recall, and slightly lower specificity and balanced accuracy, which shows that there are a noticeable amount of false positive classifications. The baseline yielded respectable results since the FFT feature was excellent for agent classification. The following plots show the relevant network performance evaluation plots and curves. Note the train/dev curve behavior since dropout is not applied during evaluation.

CNN Architecture

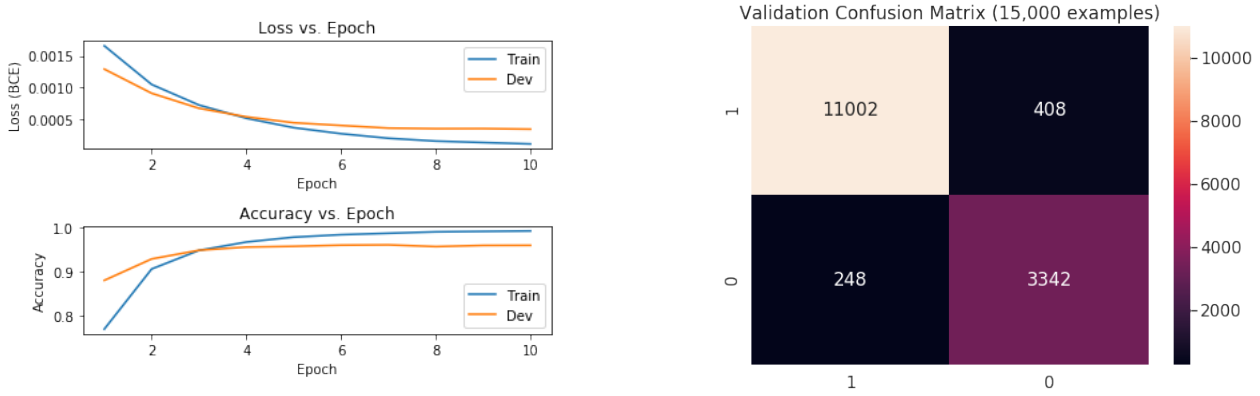

Figure 5: Training curve and confusion matrix for CNN architecture. **X axis of confusion matrix is true label and y axis of confusion matrix is predicted label. 1 is good agent, 0 is malicious agent.**

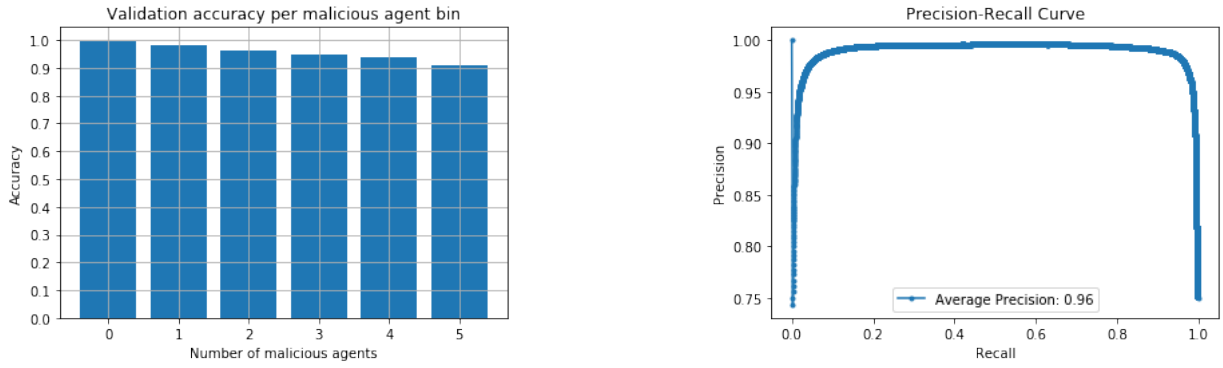

Figure 6: Validation accuracy per malicious agent bin, and precision-recall curve for CNN architecture.

LSTM Architecture

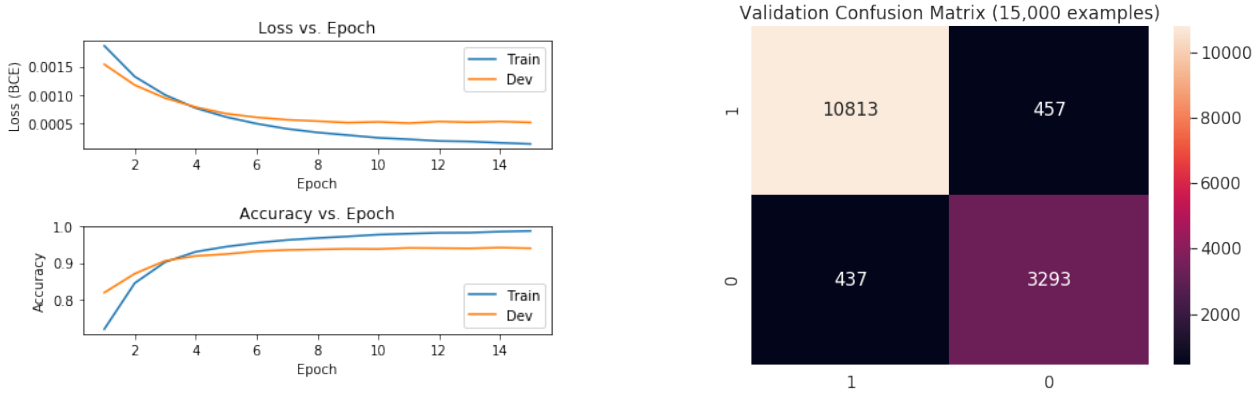

Figure 7: Training loss curves and validation confusion matrix for LSTM architecture. **X axis of confusion matrix is true label and y axis of confusion matrix is predicted label. 1 is good agent, 0 is malicious agent.**

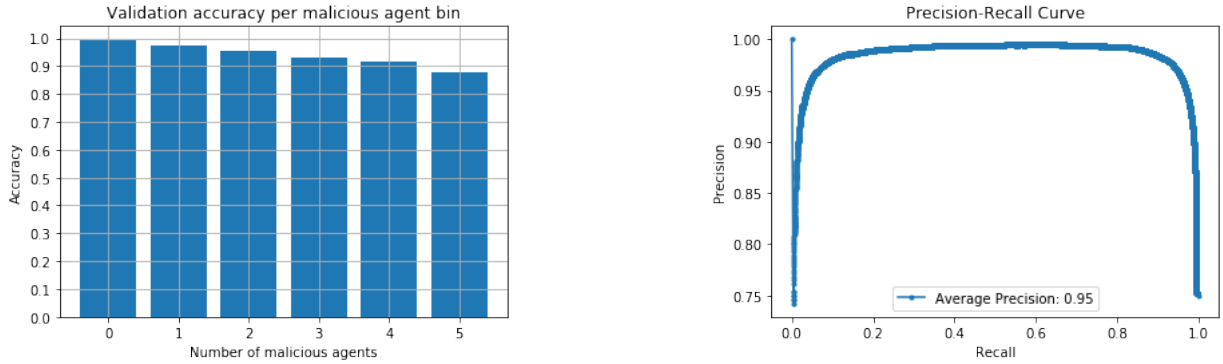

Figure 8: Validation accuracy over malicious agent bins and precision-recall curve

Both CNN and LSTM networks are better than the baseline. In comparing the LSTM and CNN architectures, they are extremely similar in terms of performance, however the CNN architecture is slightly better in all performance metrics. The main difference between the two networks is the training time, where the CNN architecture reaches convergence faster and has faster computation time than the LSTM architecture. There is some overfit to the training data as seen in the training curves where the test (dev) curve plateaus at an earlier epoch than the training curve, which I attempted to correct with batch norm regularization and dropout in the network structure. The LSTM confusion matrix is slightly more balanced across false positives/negatives than the CNN confusion matrix. There is a similar trend in decreasing validation accuracy across increasing malicious agent bins which is expected since the network attempts to generalize across all possible numbers of malicious agents, and more malicious agents produce more convoluted trajectories with less correlated features which may differ by a lot between different bins. The precision-recall curve is similar in both networks, and shows a large trade-off slope near the edges. The LSTM and CNN average precision scores (AUPRC, not interpolated between points like trapezoidal rule) are very good (0.95, 0.96), and shows that the classifiers have good output quality. Note that the precision-recall curve is used instead of ROC since there is a class imbalance.

Conclusion/Future Work

This work presents two main learning architectures to perform time series classification for malicious agents in robot formation control. Since the network assumes no knowledge of the system dynamics, control, and graph connection, the input features to the networks are the position, velocity, and position FFT data for all 10 robots. Networks were trained with a training set of 3000 simulations, and testing and validating on 1500 simulations each. From the test results, both CNN and LSTM architectures yielded similar performances across precision, recall, specificity, and balanced accuracy, where the CNN slightly outperforms the LSTM architecture (both were better than the MLP baseline). The CNN network converged in fewer epochs and is computationally faster to train than the LSTM network since CNN operations are optimized. These networks were effective for the time-series classification task since the network extracted useful temporal relationship features using their respective learning structures. The CNN performed slightly better than the LSTM since the convolution operation extracted more relevant features expressing the robot control interactions. For future work, a combined CNN-LSTM network would be explored, as well as training classifiers across different malicious strategies. A VAE would also be explored to learn latent features and provide stochasticity to improve classification.

References

[1] Gil, Baykal and Rus. "Resilient Multi-Agent Consensus Using Wi-Fi Signals." 2018 IEEE Control Systems Letters (2018): 126-131. doi: 10.1109/HUMANOIDS.2018.8624987.

[2] Khalastchi E, Kalech M. "Fault Detection and Diagnosis in Multi-Robot Systems: A Survey". Sensors (Basel). 2019;19(18):4019. Published 2019 Sep 18. doi:10.3390/s19184019.

[3] Zool Hilmi Ismail and Nohaidda Sariff. "A Survey and Analysis of Cooperative Multi-Agent Robot Systems: Challenges and Directions, Applications of Mobile Robots." 2018 IntechOpen. doi: 10.5772/intechopen.79337.

[4] A. Tulbure and B. Bäuml, "Superhuman Performance in Tactile Material Classification and Differentiation with a Flexible Pressure-Sensitive Skin," 2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids), Beijing, China, 2018, pp. 1-9.

[5] Yu Zhao, Rennong Yang, Guillaume Chevalier, Ximeng Xu, and Zhenxing Zhang, "Deep Residual Bidir-LSTM for Human Activity Recognition Using Wearable Sensors," Mathematical Problems in Engineering, vol. 2018, Article ID 7316954, 13 pages, 2018. https://doi.org/10.1155/2018/7316954.

[6] Mondal, Arindam et al. "A novel multi-agent formation control law with collision avoidance." IEEE/CAA Journal of Automatica Sinica 4 (2017): 558-568.

[7] S. Hochreiter and J. Schmidhuber. "Long short-term memory". Neural computation, 9(8):1735–1780, 1997.

[8] Pedregosa et al. "Scikit-learn: Maching Learning in Python". Journal of Machine Learning Research, vol. 12, 2011, pp. 2825-2830.

[9] Paszke et. al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". Advances in Neural Information Processing Systems 32, 2019, pp. 8024-8035.

Code/Contributions

All code can be found in the final project folder of https://github.com/chongwilliam/cs229/. The repo is private, so please send a request to view the code. I worked on the entirety of this project alone.