Attack-Inspired Data Generation

Deep transfer learning to detect cyber-attacks on critical water infrastructure

Kayla Cummings, Operations Research Center

1 Introduction

Modern public infrastructure streamlines and regulates physical system events with computing capabilities. This emergent cyber layer facilitates timely and economical delivery of critical resources, but it also exposes our infrastructure to cyber attacks. Hackers have released raw sewage into waterways and have also altered the levels of chemicals used to treat tap water [1, 2]. Smart water distribution networks remotely monitor hydraulics via strategically placed sensors and meters. Program Logic Controllers (PLCs) collect sensor readings, which trigger automatic control actions to regulate system events. An operator centrally controls the PLCs via a Supervisory Control and Data Acquisition (SCADA) system [3]. Adversaries may coerce PLCs into deploying malicious control actions by altering sensor readings.

We use deep transfer learning to detect cyber attacks in SCADA outputs. Attacks are rare, so classical supervised learning yields a model that fails to generalize. However, even the most creative approaches in the literature are static [3, 4, 5, 6, 7]. Sophisticated models that report high accuracy on realistic test sets are seldom constructive in the case of failure on real world SCADA outputs. We have identified a need for a customizable, explainable attack detection paradigm that anticipates likely attack behavior. Our adversarial data augmentation model represents a hacker who directly manipulates sensor readings to trigger automatic control responses that disrupt system hydraulics. We generate synthetic, attack-inspired data to train a deep convolutional neural network (CNN) that can classify real attacks.

This work extends a project that began during a summer internship at Oak Ridge National Lab with Jason Laska. Our initial attack generation model used naïve data augmentations such as swapping and shuffling within observations to train the attack detection model [8]. While we achieved some success, this approach leads to redundant training data. This report develops three new model-based approaches to generating synthetic training data, and implements two of them.

2 Methods

We describe the raw data and three new synthetic attack generation models. We also provide a brief summary of a previously developed attack-detection model [8].

2.1 Data

The raw data are training sets I and II from the Battle of the Detection Algorithms (BATADAL) [9], respectively containing 8761 and 4177 hourly, simulated readings from 43 sensors in a water distribution system. These are observations that represent physically accurate network flows under typical operating conditions in a real, medium-sized network [10]. Training set I is entirely clean, while training set II contains 7 realistic cyber-attacks. Each cyber-attack is between 24 and 110 hours long and constitutes approximately 12% of training set II. SCADA records measurements of flow levels and switch statuses for pumps and valves, pressure levels at the junctions before and after pump stations, and the water levels in each tank [9].

We remove seven low-variance features, and then scale them to be between 0 and 1. Then we roll the data so that each observation is a $w \times 36$ window of w consecutive readings. For now, we set $w = 24$. We perform this operation to facilitate temporal pattern recognition while training the attack detection model, and we parameterize window length instead of implementing an LSTM-RNN because we wear the hat of a real-time protagonist. In other words, we assume that we do not need to scan months into the past to ascertain whether an attack happens now.

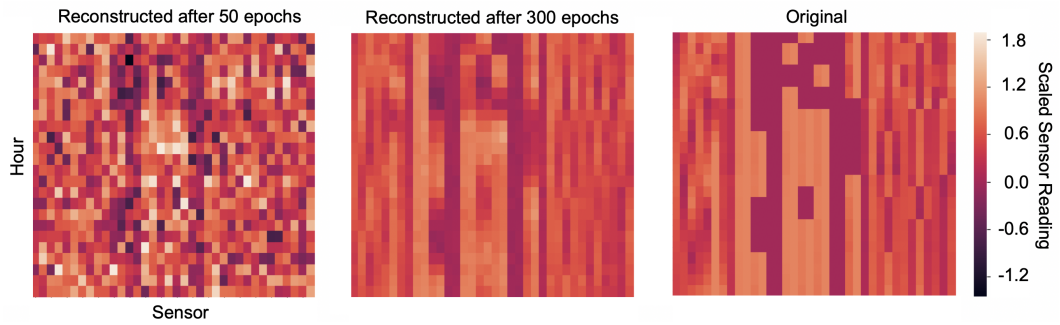

Figure 1: Example of sensor reading window reconstructed by VAE after 50 and 300 epochs.

2.2 Synthetic Attack Generation

We describe the baseline model and three new models to generate synthetic attacks.

2.2.1 Model 0: Swapping

We manually perform a swap in a clean observation as follows [8].

1. Sample a sensor type s uniformly at random from among four types: tank water level, pump/valve status (on or off), pump/valve flow, and suction/discharge pressure for each pump/valve.
2. Sample two features (f_1, f_2) uniformly at random and without replacement from the set of features with sensor type s.
3. Sample uniformly at random from start indices $i \in [0, w-2]$ and end indices $j \in [i+1, w-1]$ to obtain a contiguous, non-empty interval.
4. Ensure the attack has the potential to be visible to our model by repeating steps 1-3 until we achieve distinct readings of features f_1 and f_2 in the time range $[i, j]$, inclusive; then swap.

2.2.2 Model 1: VAE with Swapping

An autoencoder is a generative model that simultaneously trains a pair of neural networks, called an encoder and a decoder. The encoder learns a representation of the data in a low-dimensional feature space, while the decoder learns to accurately reconstruct the data from this latent space. Naturally, we use MSE loss. A variational autoencoder (VAE) equally weights reconstruction loss with a term that ensures a "smooth enough" encoding for the decoder to interpolate accurately during generation [11]. More specifically, we use Kullback-Leibler divergence of the latent space's distribution from a standard normal distribution. This ensures that the decoder is more likely to map a standard normal draw from the latent space to a realistically interpolated synthetic observation.

We train a VAE using clean training set I using architecture by Chandy *et al.* [7], and visualize an example of successful reconstruction in Figure 1. We apply Model 0 to the synthetic clean observations that we generate and label them as synthetic attacks.

2.2.3 Model 2: Adversarial VAE

Model 2 is a variation on the theme of Model 1. Using training set II, we train a VAE that includes a second branch from the latent space, which learns to classify the encoded observations using binary cross-entropy. In doing the latter, we encourage a fuzzily partitioned formulation of the latent space. We generate synthetic attacks by sampling from the latent space and discarding synthetic observations that the classifier labels as clean.

Figure 2 illustrates Model 2 architecture. Figure 3 plots training and validation loss during tuning of the loss weight parameter. Loss weights are given as ratios of VAE loss to binary cross-

entropy. We select the 1:3 weight, which yields an adequate balance of the two priorities. Figure 4 shows training and validation loss for each weight.

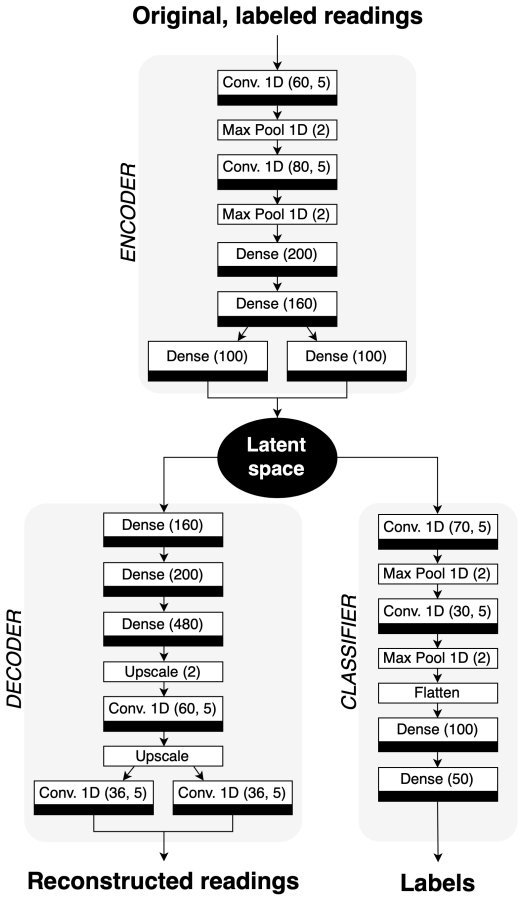

Figure 2: Model 2 trains a labelled attack embedding. A module married to a bold rectangle signifies a batch normalization layer. Module key: Conv. 1D (*num. filters, filter size*); Max Pool 1D (*resolution*); Dense (*dim.*); Upscale(*resolution*). VAE: [7].

2.2.4 Model 3: GAN

A generative adversarial network is also a pair of simultaneously trained neural networks, in which a generator produces synthetic observations that seem realistic enough to fool a discriminator that grows increasingly sophisticated [12]. The discriminator trains with respect to classical $0-1$ loss, while the generator's loss is based on whether it successfully fools the discriminator. We use the decoder of Chandy's VAE as a warm-started generator. The idea is to allow the generator to learn to convert clean windows into attack-inspired observations, rather than training it to perform the more difficult task of generating attacks from random noise. The discriminator's architecture is loosely based on the attack detection model described in section 2.3. We provide the discriminator with examples of realistic attacks from training set II.

Unfortunately, this approach failed due to a vanishing gradient problem. The generator was too sophisticated for the discriminator, due to the preliminary step of priming the generator, as well as the limited availability of realistic attack examples. The code for this approach is submitted, but no results are reported.

2.3 Attack Detection with Convolutional Neural Network

We select a convolutional neural network (CNN) for attack detection because its architecture explicitly assumes some translational invariance in the data. We use 1D-convolutional layers to detect temporal patterns by rolling sensor data into windows of consecutive readings. We follow each convolutional layer with other layers that regularize parameters, prevent overfitting, and reduce computational intensiveness. In particular, we use 1D max-pooling followed by batch normalization. The $\max$ operation selects the node firing with the strongest signal, thereby letting the most important information filter through the rest of the network. The batch normalization layer transforms the distribution of the inputs so that their new mean is 0 and their standard deviation is close to 1, mitigating the difficulties that arise when parameters fluctuate during training [13]. After convolving, we flatten the inputs and stack dense layers, followed again by batch normalization. Dense layers of dimension D fully connect the input nodes to the outputs, with ReLU activation. The output dense layer is followed by softmax activation [8].

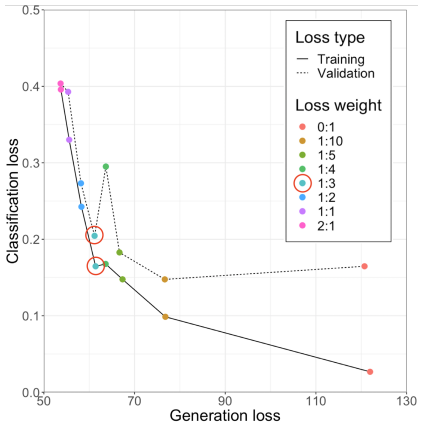

Figure 3: Loss weight selection to balance realistic reconstruction with accurate classification.

We use stochastic gradient descent to optimize our CNN with binary cross-entropy loss. For training data, we augment BATADAL training set II with synthetic attacks generated from Models 0, 1, or 2, until the two classes are balanced, reserving a stratified 30% for model validation.

3 Results and Conclusions

Confusion matrices in Figure 5 demonstrate performance on the official BATADAL test set, containing 2089 hourly readings and 7 realistic attacks comprising 27.5% of the data. As a baseline, we train the attack detection model using the unaltered BATADAL training set II with class weights that balance their classification priority in the loss function, achieving 54.4% precision and 20.0% recall. Model 1 offers no meaningful advantage over the baseline, while Model 2 exhibits increased precision and recall of 66.3% and 22.7%. In the hybrid model, we generated equal numbers of synthetic attacks from Models 1 and 2 to augment training set II, and see gains over the baseline proportional to the number of Model 2 synthetic attacks.

The results indicate some promise in model-based data generation to supplement imbalanced classes. To further improve, we can post-process the reconstructed data using simple domain-specific knowledge. For example, Figure 1 shows some binary features are binary are reconstructed as continuous, so we might round these outputs. Additionally, we can fit a different classifier to the embedding and use it to label our synthetic attacks. Including the classifier in Model 2 encourages a partitioned embedding, but because the classifier is jointly trained with the decoder, we might be losing some potential classification accuracy. In conjunction with more refined hyperparameter tuning, we expect the results to improve even more.

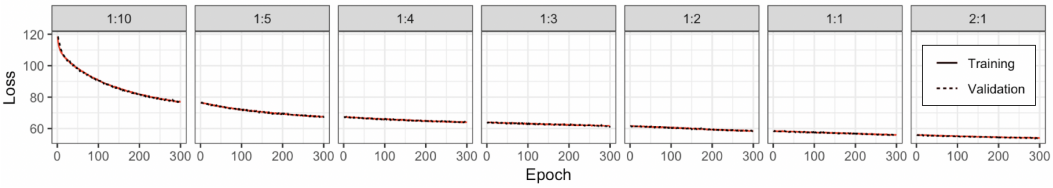

Figure 4: Un-weighted sums of generation and classification loss for loss weight selection.

		Baseline		Model 1		Model 2		Hybrid	
True	Clean	1402.8	95.2	1407.2	90.8	1432.4	65.6	1420.6	77.4
	Attack	454.6	113.4	457.8	110.2	439	129	445.6	122.4
		Clean	Attack						
		Predicted							

Figure 5: Confusion matrices for test set, averaged over 5 trials.

4 Acknowledgements

Code uses Tensorflow [14] and Keras [15]. GAN code builds on code available in the tutorial by [16].

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internship program. Managed by UT Battelle, LLC under Contract No. DE-AC05-00OR22725 for the U.S. Department of Energy.

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 1122374. The opinion, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the opinions of the National Science Foundation.

References

[1] J. Slay and M. Miller, *International Federation for Information Processing*, pp. 73–82. Boston: Springer, 2008.

[2] M.-A. Russon, "Hackers hijacking water treatment plant controls shows how easily civilians could be poisoned." International Business Times, 2016. Accessed 24 July 2018.

[3] C. M. Ahmed, C. Murguia, and J. Ruths, "Model-based attack detection scheme for smart water distribution networks," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 101–113, 2017.

[4] M. Housh and Z. Ohar, "Model based approach for cyber-physical attacks detection in water distribution systems," in *World Environmental and Water Resources Congress*, pp. 727–736, 2017.

[5] J. Goh, S. Adep, M. Tan, and L. Z. Shan, "Anomaly detection in cyber physical systems using recurrent neural networks," in *18th IEEE International Symposium on High-Assurance Systems Engineering*, pp. 140–145.

[6] R. Taormina and S. Galeli, "Real-time detection of cyber-physical attacks on water distribution systems using deep learning," in *World Environmental and Water Resources Congress*, 2017.

[7] S. E. Chandy, A. Rasekh, Z. A. Barker, and M. E. Shafiee, "Cyberattack detection using deep generative models with variational inference," *Journal of Water Resources Planning and Management*, May 2018.

[8] K. Cummings, "Attack-Inspired Data Augmentation to Protect Critical Infrastructure," tech. rep., Science Undergraduate Laboratory Internship program at Oak Ridge National Laboratory, August 2018.

[9] R. Taormina, S. Galelli, *et al.*, "The Battle Of The Attack Detection Algorithms: Disclosing Cyber Attacks On Water Distribution Networks," *Journal of Water Resources Planning and Management*, 2018.

[10] A. Ostfeld, E. Salomons, L. Ormsbee, J. G. Uber, and *et al.*, "Battle of the water calibration networks," *Journal of Water Resources Planning and Management*, vol. 138, no. 5, pp. 523–532, 2012.

[11] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," May 2014. arXiv 1312.6114.

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2672–2680, Curran Associates, Inc., 2014.

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[14] M. Abadi, A. Agarwal, *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from `https://www.tensorflow.org/`.

[15] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[16] "Deep convolutional generative adversarial network tutorial." `https://www.tensorflow.org/alpha/tutorials/generative/dcgan`, 2019. Accessed 17 May 2019.