CS230: DEEP LEARNING

Adversarial Weakness Recognition for Efficient Black-Box Validation

Robert J. Moss
Computer Science
Stanford University
mossr@cs.stanford.edu

Abstract

When validating a black-box system, exhaustively evaluating over the entire validation dataset may be computationally intractable. The challenge then becomes to intelligently automate selective validation given knowledge of the system failures experienced so far. We propose an adaptive black-box validation framework that will learn system weaknesses over time and exploit this knowledge to propose validation samples that will likely result in a failure. We use a low-dimensional encoded representation of inputs to train an adversarial failure classifier to intelligently select candidate failures to evaluate. Experiments were run to test our approach against a random candidate selection process and we also compare against full knowledgeable of the true system failures. We stress test a black-box neural network classifier trained on the MNIST dataset. Results show that using our framework, the adversarial failure classifier selects failures about 3 times more often than random.

1 Introduction

Finding failures in a validation dataset may be computationally expensive if we search over the entire dataset. Then the challenge becomes how to intelligently select candidate inputs that are likely to lead to failures. We are also interested in finding failures in black-box systems, i.e. systems where we can only pass inputs and observe outputs—without any knowledge of the inner-workings of the system. The motivation to find such candidate failure inputs is to reduce the need to exhaustively evaluate an entire validation dataset, especially for black-box systems that may be computationally expensive to call. Each input sample in the dataset may also be high-dimensional, therefore we also want to learn a low-dimensional representation of the dataset and use that representation to learn features that caused failures. An adversarial failure classifier will input the low-dimensional representation and determine if a sampled input will likely result in a failure. Our proposed framework combines the components of a dataset encoder, an adversarial failure classifier, and a candidate failure selector to propose dataset inputs that will likely result in failure, all to reduce the computational cost of evaluating the system under test and to focus only on evaluating predicted failures. The framework is available online.[1]

1.1 Related Work

Current approaches to validate black-box systems focus their search over input disturbances to find failures [Corso et al., 2020]. A black-box reinforcement learning approach known as adaptive stress testing (AST) has recently been used to find the most likely failures in aircraft collision avoidance

[1] https://github.com/sisl/FailureRepresentation.jl

CS230: Deep Learning, Autumn 2020, Stanford University, CA. (LaTeX template borrowed from NIPS 2017.)

systems [Lee et al., 2015, 2018], aircraft flight management systems [Moss et al., 2020], and autonomous vehicles [Koren et al., 2018, Koren and Kochenderfer, 2019]. An underlying assumption of the AST problem formulation is that the system under test can be modeled as a sequential decision making process with explicitly defined states. This assumption limits the application of AST for validation over a dataset and ultimately creates *more* data to validate due to applying input disturbances. Other approaches use model-based clustering to efficiently sample large datasets, but rely on tuned initialization parameters for good performance [Wehrens et al., 2004]. Bayesian methods have been used to efficiently sample data from large hierarchical datasets using techniques to fit clusters over randomly partitioned subsets of the data [Huang and Gelman, 2005]. To be effective, these techniques assume hierarchical structure in the data. Compression-based approaches have also been applied to reduce data size without loss of useful information [Ferrari et al., 2013], but are generally domain specific. The most straightforward approach is to evaluate the system exhaustively over the entire validation dataset, which may be expensive and is the main motivation of our proposed framework.

2 Dataset and Features

We are using the MNIST handwritten digit dataset [LeCun et al., 2010] as our collection of inputs we want to selectively sample. We chose MNIST because it is a well-known machine learning dataset with many benchmarks, and is small enough to quickly iterate our framework without worrying about computational concerns. The MNIST training data contains $60{,}000$ gray-scale images of handwritten digits, each represented as 28×28 pixels. The test dataset contains $10{,}000$ gray-scale images with the same 28×28 pixel dimensions. For feature extraction, we left that up to the autoencoder described in Section 3.1.

2.1 Black-Box System Under Test

To test our framework, we trained an MNIST classifier to be our black-box system under test $\mathcal{S}$. We are using the Julia machine learning package `Flux.jl` [Innes et al., 2018] for building the neural network model and training. The black-box classifier $\mathcal{S}$ consists of two dense layers and a ReLU activation, mapping the input size of $28 \times 28 = 784$ to 32 activations, and then an output layer of size 10 (for each digit class). We use the logit cross-entropy loss, which is equivalent to the cross-entropy loss after applying the $\mathrm{softmax}$ function to the predicted output $\hat{\mathbf{y}}$:

$$\mathcal{L}_{\mathcal{S}}(\mathrm{softmax}(\hat{\mathbf{y}}), \mathbf{y}) = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(\hat{y}_i)$$

Figure 1: Example failure (i.e. misclassification) which classified this image as a 1 instead of a 7.

We trained the system over 20 epochs, with a mini-batch size of 1024, and using the Adam optimizer [Kingma and Ba, 2017] with a learning rate of $\alpha = 3e^{-4}$. This classifier achieves around 93.2% accuracy, so there is room to find weaknesses to exploit failures, where we define a failure as a misclassification. Figure 1 shows an example failure where the system misclassified a particular digit.

3 Method

Our proposed framework consists of two major components: a dataset autoencoder, and an adversarial failure selector. These components are iteratively called within a *sampled validation loop*. The dataset autoencoder is used to sample m low-dimensional representations of the encoded input samples $\tilde{\mathbf{x}}$. We encode inputs into a lower-dimensional space for two reasons: 1) to reduce the potential high-dimensionality of the inputs $\mathbf{x}$ and 2) to learn features in this low-dimensional space that likely caused failures. We then split the m low-dimensional samples into a training set $\tilde{\mathcal{D}}_{\text{train}}$ and test set $\tilde{\mathcal{D}}_{\text{test}}$. The training set $\tilde{\mathcal{D}}_{\text{train}}$ is passed to an adversarial agent that learns characteristics of the low-dimensional feature representation that led to failures. We use a failure classifier as our adversary, and then predict which inputs led to failures over the test data $\tilde{\mathcal{D}}_{\text{test}}$, then map the predicted failures from the low-dimensional space back to the original representation, and then run the candidate inputs expected to result in a failure through the system under test. Figure 2 illustrates each step of the validation framework.

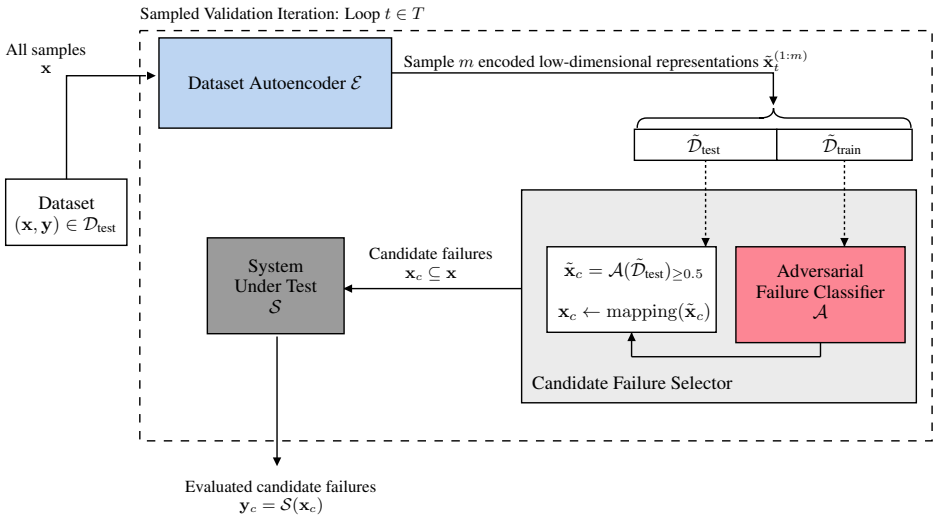


Figure 2: Validation framework—a dataset autoencoder $\mathcal{E}$ is trained on the entire validation dataset $\mathcal{D}_{\text{test}}$ consisting of input samples $\mathbf{x}$. Then m samples of encoded low-dimensional representations of the inputs $\tilde{\mathbf{x}}$ are selected for this iteration t, denoted $\tilde{\mathbf{x}}_t^{(1:m)}$ for all m samples. The low-dimensional representations are then split into a training and test dataset. The training dataset $\tilde{\mathcal{D}}_{\text{train}}$ is used to train an adversarial failure classifier $\mathcal{A}$ on the encoded representations. Then the test dataset $\tilde{\mathcal{D}}_{\text{test}}$ is used to select candidate failures $\tilde{\mathbf{x}}_c$ as predicted by the adversary. Finally, the candidate failures from the adversary $\tilde{\mathbf{x}}_c$ are mapped back to the original inputs $\mathbf{x}_c \subseteq \mathbf{x}$ and evaluated by the system under test $\mathcal{S}$.

3.1 Dataset Autoencoder

To get a low-dimensional representation of the inputs $\mathbf{x}$, we used an autoencoder network [Kramer, 1991]. We trained the autoencoder $\mathcal{E}$ on the MNIST test dataset $\mathcal{D}_{\text{test}}$ and sample from the low-dimensional representation $\tilde{\mathbf{x}}$ as inputs into our adversarial failure classifier. We use the MNIST test set because this is our input validation set—thus, we want our autoencoder to only have information about the validation set, without the need to have access to the training set used by the system under test. The autoencoder network maps the 28×28 input image $\mathbf{x}$ into a low-dimensional latent space of size 64 using a LeakyReLU activation. Then the decoder will take the 64-dimensional representation $\tilde{\mathbf{x}}$, again using a LeakyReLU activation layer, and attempt to recover the original input $\mathbf{x}'$. We pre-trained the autoencoder over 20 epochs, with a mini-batch size of 1000, and tuned the network parameters using the Adam optimizer with a learning rate of $\alpha = 1e^{-3}$. Training is unsupervised and we use the mean squared error loss function:

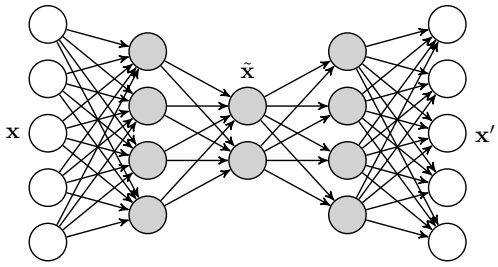
$$\mathcal{L}_{\mathcal{E}}(\mathbf{x}', \mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} (x_i' - x_i)^2$$

Figure 3a illustrates the autoencoder network architecture and Figure 3b shows samples of the true inputs and their output after encoding/decoding.

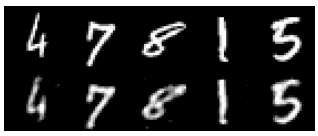
3.2 Adversarial Failure Classifier

To learn the low-dimensional features that are likely to cause failures, we train an adversary $\mathcal{A}$ in the validation loop to classify failures. The supervised adversary is trained on the partition $\tilde{\mathcal{D}}_{\text{train}}$ of the low-dimensional samples $\tilde{\mathbf{x}}$ and outputs a prediction that a given input would lead to a system failure. In order to get the target classifications $\mathbf{y}$, we use the system $\mathcal{S}$ to run the true inputs associated to the encoded inputs which are part of the training data $\tilde{\mathcal{D}}_{\text{train}}$. This gives us the targets we can now train our adversary on. Our adversarial loss function is the binary cross-entropy loss:

$$\mathcal{L}_{\mathcal{A}}(\hat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{m} \sum_{i=1}^{m} y_i \log(\hat{y}_i) - (1 - y_i) \log(1 - \hat{y}_i)$$



(a) Dataset autoencoder architecture, with inputs $\mathbf{x} \in \mathbb{R}^{28\times28}$, encoded through a dense $\mathrm{LeakyReLU}$ layer of size $\frac{28\times28}{2}$ to a low-dimensional representation $\tilde{\mathbf{x}} \in \mathbb{R}^{64}$, then decoded through a $\mathrm{LeakyReLU}$ layer of size $\frac{28\times28}{2}$, outputting $\mathbf{x}' \in \mathbb{R}^{28\times28}$.



(b) Sampled output from the MNIST autoencoder: true input is on top and the recovered input on bottom.

Figure 3: Dataset autoencoder architecture and sample decoded output.

The adversarial network architecture consists of 3 dense layers which map the low-dimensional representation of $\tilde{\mathbf{x}} \in \mathbb{R}^{64}$ through a ReLU layer of size 128, another ReLU layer of size 64, and finally an output sigmoid layer to map the predictions to a probability. For each sampled validation iteration t (shown in Figure 2), we retrain the adversary $\mathcal{A}$ for 20 epochs using the Adam optimizer with learning rate $\alpha = 3e^{-5}$. Notice that our learning rate is very small, this is so we do not overfit to early iterations in t and can generalize across different samples of the low-dimensional space. The adversary will use the test data partition $\tilde{\mathcal{D}}_{\text{test}}$ to select the encoded input that it predicted would lead to a failure. We use the threshold of $\mathcal{A}(\tilde{x}) \geq 0.5$ to indicate the input $\tilde{x} \in \tilde{\mathcal{D}}_{\text{test}} \subset (\tilde{\mathbf{x}}, \mathbf{y})$ led to a failure. All encoded inputs in the test dataset that led to a failure are considered candidate failure scenarios, and we denote them as $\tilde{\mathbf{x}}_c$. We use a mapping from the encoded inputs $\tilde{\mathbf{x}}_c$ to the original inputs $\mathbf{x}_c \subseteq \mathbf{x}$, and finally pass the failure candidates to the true system $\mathcal{S}$ for actual evaluation.

4 Experiments and Results

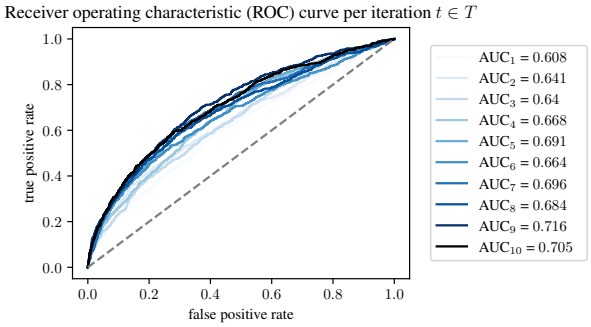
To evaluate our approach, we ran $T = 10$ sampled validation iterations, sampling $m = 500$ random encodings and partitioning these samples in half into $\tilde{\mathcal{D}}_{\text{train}}$ and $\tilde{\mathcal{D}}_{\text{test}}$ for the adversary. Because the failure rate for our system under test $\mathcal{S}$ is about 0.0677, we augment the training dataset by duplicating the known failures 10 times after running each training set through the system $\mathcal{S}$ to get the true outputs $\mathbf{y}_{\text{train}}$. We use two main metrics to evaluate the performance of the adversary: *precision* and *recall*. During each iteration t, we save off the current adversary $\mathcal{A}_t$ and evaluate the *area under the ROC curve* (AUC) as shown in Figure 4a. The ROC curve highlights incremental improvement of the adversary after each iteration. Note that we retrain the new adversary $\mathcal{A}_t$ starting from the network weights learning by the previous adversary $\mathcal{A}_{t-1}$. To balance between precision and recall, we swept over the prediction threshold to determine which threshold value to select (see Figure 4b). Based on this tuning sweep, we chose a threshold of $\hat{y} \geq 0.5$ to indicate a positive failure prediction by the adversary.

During each iteration t, the adversary selects k candidate inputs predicted to be failures. For comparison, we employ a random selection of k candidates and evaluate the precision and recall metrics of the random scheme. This allows us to compare our adversarial learning approach to a baseline. Table 1 quantifies the evaluation metrics for the adversary and random candidate selector.

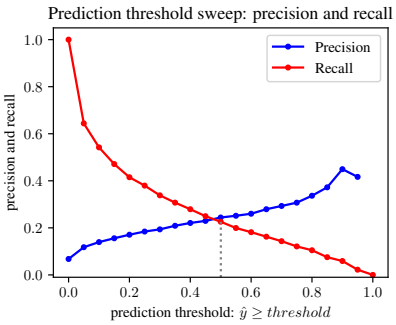
Table 1: Evaluation Metrics

Failure Selector	**Precision**[*]	**Recall**[*]	**Sampled Precision**[†]	**Sampled Recall**[†]
Adversary $\mathcal{A}$	0.2441	0.2260	0.2374 ± 0.11	0.3244 ± 0.17
Random	0.0647	0.4712	0.0618 ± 0.04	0.0910 ± 0.07

[*] Run over $\mathcal{D}_{\text{test}}$ only calculated for the "failure" class.
[†] Calculated from $T = 10$ iterations of the *sampled validation loop*.



(a) ROC curve and AUC for each adversary $\mathcal{A}_t$.



(b) Tuning prediction threshold.

Figure 4: ROC curve and evaluation metrics.

5 Analysis and Discussion

Shown by the confusion matrix in Figure 5, the adversarial failure classifier achieves about 0.24 in precision and 0.23 in recall. Compared to random, the precision rate is about 3 times better using the adversary. Random recall—as expected—is around 0.5. These results show that our approach learned the low-dimensional representation of a failure over the validation set, and based on this information it selected candidate failures to be evaluated. To see which element of the low-dimensional feature vector contributed the most to a likely failure as predicted by the adversary, we encoded a one-hot vector over $\mathbb{R}^{28\times28}$ (i.e. the input space) and plotted the output likelihood of failure for each of the pixels, shown in Figure 6.

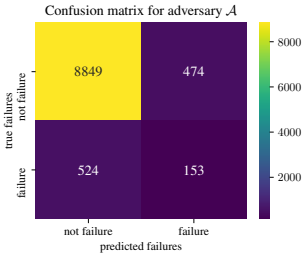


Figure 5: Confusion matrix.

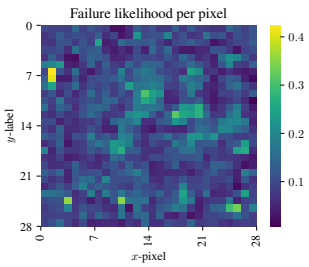
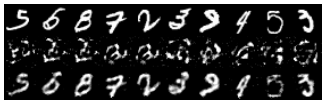


Figure 6: Failure likelihood.

Figure 7 compares the failures classified by the adversary. For the true positives in Figure 7a, the top row of digits are the 10 digits with the highest predicted failure likelihood that were true failures, the middle row shows the feature that had the strongest influence on the failure classification (decoding a one-hot vector representation of the maximum of a $\mathrm{softmax}$ over low-dimensional inputs), and the bottom row shows the reconstructed digits after passing through the autoencoder. Similarly for the false negatives in Figure 7b, the top row are the 10 digits with the lowest predicted failure likelihood that were true failures, the middle row shows the strongest influential feature, and the bottom row shows the output of the autoencoder. Notice that certain features in the middle row are shared among the other digits, indicating features that play a larger role in classifying failures.

6 Conclusion and Future Work

To avoid exhaustively searching an entire validation set for failures, we show that an iterative framework that uses an adversarial failure classifier trained on low-dimensional representations of the inputs can select failures about 3 times more likely than randomly choosing candidates to evaluate. To extend this framework, we could investigate different adversarial architectures, particularly around deep reinforcement learning. We could also explore how to take the true failures found, automatically improve the system under test, and then in a continual learning approach we could rerun this validation loop to use prior knowledge of previous system failures to find new system failures.



(a) Adversarial true positives.

(b) Adversarial false negative.

Figure 7: MNIST failure predictions from the adversary.

7 Contributions

Robert Moss consulted regularly with Bernard Lange, who is taking CS330 Meta Learning, regarding the project ideas. Originally, Bernard and Robert were going to collaborate on this project but took different directions as the quarter progressed. All of the work seen here (literature review, framework design, coding, training, analysis, plotting, and writing) were completed solely by Robert Moss.

References

Anthony Corso, Robert J. Moss, Mark Koren, Ritchie Lee, and Mykel J. Kochenderfer. A survey of algorithms for black-box safety validation, 2020.

Ritchie Lee, Mykel J Kochenderfer, Ole J Mengshoel, Guillaume P Brat, and Michael P Owen. Adaptive stress testing of airborne collision avoidance systems. *Digital Avionics Systems Conference (DASC)*, 2015.

Ritchie Lee, Ole J. Mengshoel, Anshu Saksena, Ryan Gardner, Daniel Genin, Joshua Silbermann, Michael Owen, and Mykel J. Kochenderfer. Adaptive stress testing: Finding likely failure events with reinforcement learning, 2018.

Robert J. Moss, Ritchie Lee, Nicholas Visser, Joachim Hochwarth, James G. Lopez, and Mykel J. Kochenderfer. Adaptive stress testing of trajectory predictions in flight management systems. *Digital Avionics Systems Conference (DASC)*, 2020.

Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J Kochenderfer. Adaptive stress testing for autonomous vehicles. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1–7. IEEE, 2018.

Mark Koren and Mykel J Kochenderfer. Efficient autonomy validation in simulation with adaptive stress testing. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 4178–4183. IEEE, 2019.

Ron Wehrens, Lutgarde MC Buydens, Chris Fraley, and Adrian E Raftery. Model-based clustering for image segmentation and large datasets via sampling. *Journal of Classification*, 21(2):231–253, 2004.

Zaijing Huang and Andrew Gelman. Sampling for Bayesian computation with large datasets. *Available at SSRN 1010107*, 2005.

Carlotta Ferrari, Giorgia Foca, and Alessandro Ulrici. Handling large datasets of hyperspectral images: Reducing data size without loss of useful information. *Analytica chimica acta*, 802:29–39, 2013.

Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable modelling with Flux. *CoRR*, abs/1811.01457, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.