Independent Study Notes Spring 2019

Brandon Foltz (brandon@brandonfoltz.com)

Paper Summaries

Alrawi O, Lever C, Antonakakis M, Monrose F. SoK: Security Evaluation of Home-Based IoT Deployments. InSoK: Security Evaluation of Home-Based IoT Deployments 2019 (p. 0). IEEE.

Introduction notes the state of IoT insecurity, and that vendors and researchers are often applying “ad-hoc” solutions without systematized/standardized solutions in place. Notes that services provided by IoT devices vary widely. This paper provides a systematized evaluation of 45 home-based devices and reviews their security properties.

Methodology provides an abstract graph-based model of typical IoT deployments, where devices/apps/cloud are nodes and their respective connections are edges. This model is used as a framework to describe vulnerabilities and potential mitigations existing in the system. This is done by a description of *Attack Vectors*, *Mitigations*, and *Stakeholders.* The scope of the evaluation covers an IP network attacker only.

Categories of Device attack vectors include:
- Vulnerable services
- Weak authentication
- Default configuration

Categories of Mobile app attack vectors
- Permissions
- Programming (eg. bugs/vulnerabilities)
- Data protection (sensitive information)

Communication component attack vectors
- Encryption (or lack thereof)
- MITM

Cloud endpoint shares some attack vectors above

- Vulnerable services
- Weak authentication
- Encryption

Page 4 has a table of related works and indications of which works discuss the various attack vector categories, mitigations, and stakeholders as listed above.

Device take away: platform permissions, unauthenticated services, insecure config, software/hardware bugs contribute to device insecurity and combinations amplify the insecurity. System patching can address most issues but is not perfect. Main stakeholder for device is the vendor. Similar problems exist in general purpose computers.

Mobile app take away: Inherent trust is given to mobile applications. This trust can be abused, vendors and users should ensure proper deployments for security by following best practices. Limiting mobile access to the device can reduce attack impact.

Cloud endpoint take away: These show insecure deployment through configuration and API implementations. These can be addressed with existing tools. Some endpoints suffer from over-privileged access and privacy issues. IoT vendors are shifting to managed IoT endpoint infrastructure (eg. Amazon IoT), and more research is needed on these systems to understand their weaknesses.

Communication take away: Devices rely on insecure protocols that are mitigated by higher level protocols such as TLS/SSL. Some of these protocols have flaws and many devices do not use encryption at all. Custom protocols used by managed IoT platforms need auditing.

The remainder of the paper discusses specific vulnerabilities on specific devices that were evaluated. I will not copy these here for brevity & because it would omit important details.

Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y. Generative adversarial nets. InAdvances in neural information processing systems 2014 (pp. 2672-2680).

This is the seminal paper on Generative Adversarial Networks as they are commonly known today. GANs are generative networks trained by an adversarial process to (hopefully) recover the distribution of data that the training samples come from, such that new unseen samples from the same distribution can be produced on demand.

The training process involves two networks, a Generator and a Discriminator. The training process is described as a game whereby the Generator tries to generate examples that fool the Discriminator in to classifying them as real, and the Discriminator tries to learn to tell the difference between fake (generated) examples and real ones from the training set. It is

analogous to the game a currency counterfeiter and the police play, whereby the counterfeiter always wants to increase the quality of their fakes so that they cannot be distinguished from real currency, and the police want to improve their methods of discovering these fake examples.

It is a short paper with full derivations of the loss function and proofs that convergence is possible. However in practice and as many follow up papers showed, one of the main problems with Vanilla GANs (as described in Goodfellows paper) are that they are far from guaranteed to converge, and they have numerous failure modes. This makes them very difficult to train in practice.

Ostensibly a GAN could be used to generate any type of data (sequential, text, image, etc) though this paper focuses on generating images of digits which mimic those from the MNIST dataset.

Unpublished paper from Professor Payton by Fenglong Ma et. al, "Fake is the New Real: Predicting Rare Diseases with Deep Generative Networks and Reinforcement Learning"

Existing methods for predicting diseases focus on common maladies as there is sufficient EHR to analyze them. This is not the case for rare diseases, which makes prediction more difficult. This paper discusses a novel system for rare disease prediction which generates high-quality task specific data.

Components discussed:
MaskEHR: Generates diverse EHR data based on data from patients with a given disease. Does so by randomly removing parts of visits. Data generation goal is then to fill in these missing visits.
RL-Selector: Reinforcement learning based data selector to choose high quality samples generated by MaskEHR.
Prediction: NN prediction component.

EHR data features (see figure 1 in paper):

- Discrete "visits" which are time-ordered.
- Each visit contains numeric diagnosis codes corresponding to diseases/other. These are discrete and unordered. The number of codes varies per visit.

Idea is to use a generative model to directly create case patient data (data for patients that have the given disease). Don't care much about control patient data as this is readily available.

Challenges:

- Temporal characteristics (should use many previous visits to generate the following visit)
- Need diverse data that is not just a replication of the input
- How to obtain high-quality data that improves rare disease prediction is main challenge

Data pipeline:
MaskEHR: Randomly removing parts of visits.
RNN-based Generator: Fills in the missing visits, using RNN because it can model the temporal characteristics among visits
Discriminator: Goal is to identify the real patient data, versus the “fake” data given by the generator.
RL-Selector: Reinforcement learning system to select high quality data. State: current generated data and all past chosen data (based on policy function and binary action (selected, not selected)).
NN-Predictor: Using the chosen dataset and original training data, a neural-network predictor is developed to identify the label of each input patient. It returns feedback (reward) to the RL-Selector component for updating the policy function, and guides MaskEHR to generate diverse and high quality case patient data

Note: It’s unclear to me in the paper whether MaskEHR is specifically referring to the masking component of the above described framework, or if it is referring to the data pipeline as a whole. I *think* it’s referring to the masker/encoder/generator/discriminator part of the data pipeline, as seen in Figure 3 of the paper.

The system above is tested on “three real datasets”, presumably the datasets representing patients in the three disease categories covered.

Inputs to the prediction component are the real case patient data and the selected data.

In the masking component, the first visit is never masked (deleted). For generation of visit *t*, the previous 0..*t-1* visits are used as input to the RNN based generator, which contains some hidden state and as such can take multiple inputs which update this hidden state. Their RNN consists of Gated Recurrent Units.

Potdar K, Pardawala TS, Pai CD. A comparative study of categorical variable encoding techniques for neural network classifiers. International Journal of Computer Applications. 2017 Oct;175(4):7-9.

Paper covers various techniques for encoding categorical data. Shows one-hot and binary encoding to perform the same on a classification task. There are some other encoding schemes used as well but don’t seem as universally applicable, though I didn’t completely understanding.

My takeaway is that I can use a binary encoding for categorical values in a NN input and expect similar classification performance as I would with one-hot, while getting the advantages of a more dense representation.

Borji A. Pros and cons of gan evaluation measures. Computer Vision and Image Understanding. 2019 Feb 1;179:41-65.

This paper describes some of the difficulties with GAN evaluation, and outlines some desired characteristics of GAN evaluation measures. These measures fall into two categories, quantitative and qualitative measures. It goes on to describe 29 metrics for evaluation of GAN output, 24 of which are quantitative and 5 of which are qualitative measures. Some of these measures are only applicable to certain domains (eg. image generation) but most are general enough to be used for other kinds of data.

I won't go in to describing all the metrics as some are rather detailed, but the desiderata outlined provide a general idea of what the author is considering "high quality output":

1. Favor models that generate high fidelity samples (ability to distinguish generated from real ones; discriminability)
2. Favor models that generate diverse samples (metric should be able to detect overfitting, mode collapse, mode drop)
3. Favor models with disentangled latent spaces (controllable sampling)
4. Have well defined bounds
5. Be sensitive to image distortions and transformations. These transformations do not change the semantic meaning on things like images, so the measure should be invariant to these transformations.
6. Agree with human perceptual judgement.
7. Have low sample & computational complexity.

Of the methods outlined in the paper, I chose to focus on two as they address two of the most common problems with GANs being mode collapse and memorization:

Number of Statistically Different Bins: Gather a dataset of N real and N fake examples. Use K-Means clustering to cluster the real examples into K clusters. Then determine which clusters the fake examples fall in to. If the fake distribution closely matches the real distribution, then the difference between the number of real and fake samples that fall into each cluster should be statistically insignificant. This method easily allows one to identify mode collapse or mode dropping.

Wasserstein Critic / Wasserstein Distance Approximation:
Train a classifier *f* (typically a neural network with clipped weights such that it has a bounded derivative) to output values close to 1 for real samples and 0 for fake samples. The metric is then calculated as follows:

$W(x_{real},x_{fake}) = (1/N)\text{sum}(f(x_{real})) - (1/N)\text{sum}(f(x_{fake}))$

A good score is close to 0 (critic can not easily distinguish between real and fake), a bad score is close to 1 (trivially distinguishable).

Fedus W, Goodfellow I, Dai AM. Maskgan: Better text generation via filling in the_. arXiv preprint arXiv:1801.07736. 2018 Jan 23.

This paper opens by pointing out some of the weaknesses of existing techniques for generative language modeling, translation, and related tasks. Typically these tasks are carried out by a seq2seq model or similar RNN. However the authors point out that these models cannot be conditioned on sequences that were not in the training set. The authors propose to use a GAN to increase the quality of these types of language modeling tasks.

The authors introduce a text generation model which is trained on in-filling redacted text (MaskGAN). They go on to discuss actor-critic architectures in large action spaces as well as evaluation metrics for determining the quality of synthetic data.

The MasGAN architecture considers pairs of input and target tokens (xi, yi). There is a special <m> token to denote a masked token, whereby the original token is replaced by “a hidden token” (pretty sure this is the <m> token they’re referring to). The generator produces a filled-in token from this input which is then passed to the discriminator. Important to note about their architecture is that the discriminator provides a decision for *each* token in the sequence, and not just the whole sequence. Their model is a seq2seq model whereby the input sequence is encoded by an encoder network, and then the hidden state is decoded by the decoder network which predicts the actual infilled character. See the paper for diagram.

Training is done by estimating the gradient for the generator by using policy gradients. This is necessary due to the sampling operations on the generator’s probability distribution causing the output to not be fully differentiable. The authors note that reinforcement learning was used with GANs for language modeling by Yu et al. 2017. See the paper for full derivation of the loss function and training algorithm. The authors used the Penn Treebank dataset as well as an IMDB movie review dataset for training their model and evaluating its output.

Evaluation of generative models is difficult and is still an open research area. The authors chose to use the BLEU score as one metric for evaluating their model. The authors chose specifically not to focus on optimizing for validation perplexity which other papers have used in the past as an evaluation metric.

One evaluation metric that can be directly calculated is the degree of mode collapse. The model does show some mode collapse as it has a reduced number of unique bigrams/trigrams/quadgrams as compared to other state of the art models. However when human evaluation was done via mechanical turk, the MaskGAN model was evaluated considerably more favorably than existing models on criteria such as gramaticality, topicality, and overall.

Lipton ZC, Berkowitz J, Elkan C. A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019. 2015 May 29.

This is a lengthy paper which discusses the motivations for creating Recurrent Neural Networks, how they work, what they can be used for, and more. It begins by discussing neural networks in general and then continuing on to describe how recurrent networks are developed from them. The paper discusses recent developments such as Long Short Term Memory cells and their benefits, as well as Gated Recurrent Units and others. A full list of topics is outlined below:

- Feedforward Neural Networks

- Backpropogation
- Recurrent Neural Networks (early designs)
- Training of Recurrent Neural Networks
- Modern RNN Architecture(s)
- Long Short Term Memory cells
- Bidirectional RNN (BRNN)
- Neural Turing Machines
- Applications of LSTM and BRNN
- Representations of inputs and outputs
- Evaluation methods
- Applications including translation and image captioning

I will not describe these topics in detail because that's essentially the point of the paper. It provides sufficient background for someone unfamiliar with Recurrent Neural Networks to start using them for practical applications and research. Highly recommended reading!

Arora S, Zhang Y. Do gans actually learn the distribution? an empirical study. arXiv preprint arXiv:1706.08224. 2017 Jun 26.

This paper is most often cited for introducing the idea of using the "Birthday Paradox" to estimate the support size of the generated distribution. This can be used to evaluate how well the generated distribution matches that of the desired training distribution.

The authors start by posing the question "Do GANs actually learn the distribution they were trained with" eg. the target distribution. Some past proposed methods for evaluation are discussed and the birthday paradox test is proposed as a new metric. Full mathematical derivation of the metric and it's motivations are included. Additionally results of using this metric to evaluate the output of several GAN architectures as compared to other evaluation metrics in use at the time are shown.

Theis L, Oord AV, Bethge M. A note on the evaluation of generative models. arXiv preprint arXiv:1511.01844. 2015 Nov 5.

This paper discusses different training objectives for GANs, as well as different methods for evaluating their output quality. The training objectives discussed include Kullback-Leibler divergence, Maximum Mean Discrepancy, and Jensen-Shannon divergence. The important point about training GANs with these objectives are that they are not equivalent, and training under different objectives can produce very different results. Essentially the objective chosen is application specific, as an objective that works well for one application will not necessarily be good for another.

Regarding evaluation of generative model output, the main takeaway is "qualitative as well as quantitative analyses based on model samples can be misleading about a model's density estimation performance". First discussed is the common measure of log-likelihood and how you can have a very high log-likelihood but still produce very poor samples, so this quantitative measure is not sufficient for evaluation.

The authors go on to show that nearest neighbors comparison of samples and training data is not a good way to detect overfitting, because even minute changes in an image (for example) can significantly

change the euclidean distance from it to a visually similar image. Thus nearest neighbors is not suitable to detect overfitting except in the most extreme cases.

Finally, the authors suggest that Parzen window estimates be avoided altogether since good models perform worse under this evaluation method than far less desirable models.

Kusner MJ, Hernández-Lobato JM. Gans for sequences of discrete elements with the gumbel-softmax distribution. arXiv preprint arXiv:1611.04051. 2016 Nov 12.

The authors introduction covers problems with training GANs to generate discrete sequences due to the non-differentiability of the sampling operation. This propose a differentiable sampling operation which can be used to alleviate this problem. The authors show its use in an LSTM based GAN for generating short text sequences. The paper covers the derivation of this “gumbel-softmax distribution” as well as the process followed to train the GAN using said distribution.

Salimans T, Goodfellow I, Zaremba W, Cheung V, Radford A, Chen X. Improved techniques for training gans. InAdvances in neural information processing systems 2016 (pp. 2234-2242).

This paper provides a series of “techniques that are heuristically motivated to encourage convergence” when training GANs. It is known by this time (though reviewed here) that training GANs is very difficult and not necessarily guaranteed to converge.

Some of the methods discussed include:

Feature Matching

Intended to prevent a generator from over-fitting on the current discriminator, this method sets a new objective for the generator to match various statistical features of the data. The discriminator is used to choose which statistical features are “worth” matching.

Minibatch discrimination

Intended to combat mode collapse (a common GAN failure mode where the same point is always output), this technique is a general one. Rather than show the discriminator a single point at a time (where in the case of mode collapse, the discriminator cannot distinguish identical “real” and “fake” points), many examples (a minibatch) should be “discriminated” at once to provide a more useful gradient for training the generator. See the paper for implementation details.

Historical Averaging

Adds an additional term to the loss function which is the average value of parameters over some number of past time steps. The authors claim this method allows for convergence in some situations where gradient descent alone fails.

One-sided label smoothing

Instead of providing labels of 0 and 1, for binary classes, replace 1 with 0.9 Encourages convergence.

Virtual batch normalization

A method to combat some of the problems with batch normalization due to output of a network for a given input being highly dependent on other examples in the same batch.

The paper goes on to show results of training with some of these techniques and compares them with existing results on a number of evaluation criteria.

Arjovsky M, Chintala S, Bottou L. Wasserstein generative adversarial networks. InInternational Conference on Machine Learning 2017 Jul 17 (pp. 214-223).

Proposes “Wasserstein distance” aka “earth movers distance” between distributions as training objective which avoids many problems with traditional GANs such as mode-collapse.

Datasets

AWS Honeypot data: https://www.kaggle.com/casimian2000/aws-honeypot-attack-data

SSH Brute Force attack data: https://www.kaggle.com/lako65/ssh-brute-force-ipuserpassword

IoT Device Setup Captures: https://www.kaggle.com/drwardog/iot-device-captures

KDD99: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

Malware Mixed Captures: https://www.stratosphereips.org/datasets-overview

Publicly Available PCAP files: https://www.netresec.com/?page=pcapfiles

UNSW-NB15 Dataset:
https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/

Tutorials / Links / Definitions

Active learning definition: https://en.wikipedia.org/wiki/Active_learning_(machine_learning)

N-gram: https://en.wikipedia.org/wiki/N-gram

PyTorch GAN tutorial:
https://medium.com/@devnag/generative-adversarial-networks-gans-in-50-lines-of-code-pytorch-e81b79659e3f

PyTorch master documentation: https://pytorch.org/docs/stable/index.html

PyTorch DCGAN Tutorial: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

PyTorch Dropout:
https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/dropout_layer.html

Overview of GAN varieties:
https://medium.com/jungle-book/towards-data-set-augmentation-with-gans-9dd64e9628e6

PacketGAN / Paper Brain Dump

Code Repository: https://github.com/JayWalker512/PacketGAN

Introduction / Description

Most of this work is based (or at least inspired by) an unpublished paper titled “Fake is the new real …. “ in which the authors trained a Generative Adversarial Network to generate novel examples of Electronic Health Records from “case patients” who exhibit a rare disease. The goal (as with any GAN) was to generate new examples of data from the same distribution as the training data. If this can be done, it may be possible to generate as much training data as is needed to improve performance of a classifier on the task of predicting these rare diseases.

In this research, we are interested in generating examples of network packets (or at least their metadata) form the same distribution as “attack packets” from some type(s) of network attacks. These may include DoS, Exploits, Reconnaissance and so on. We wish to do this because examples of such attacks are hard to gather, given that they are typically low footprint attacks (few packets) and do not typically announce themselves as “attack” packets (hard to recognize).

I chose the same approach as the authors of “Fake is the new real”, with some variations. Namely their approach was much more of a “data generation pipeline” involving multiple components for selecting data. My work thus far has just been to train a vanilla Generative Adversarial Network (Goodfellow) and evaluate the quality of generated data.

GAN Design

To produce such a generator network as described in Goodfellow’s paper, we must have two neural networks dubbed the Generator and the Discriminator. The Generator output has the same shape as the data you wish to generate. The Discriminator is a classifier with two classes, “real” or “fake”. The training procedure is a min-max game whereby the Generator network tries to generate output which fools the Discriminator, and the Discriminator tries to learn to distinguish between examples of real data and data provided by the Generator.

My Generator is a two-layer network, the first of which consists of GRU cells and the second is a simple fully-connected layer with tanh activation function (note, this might have changed by the time someone else reads this. The network architecture is not fixed it stone and

it might be helpful to try many different ones). The output of the network is run through a sigmoid activation function so that all outputs are in the range (0,1).

~~The input to the network is a sequence of network packets that have been encoded into a suitable feature representation (more on that later). Each element of this sequence is concatenated with a vector of noise prior to being fed into the network. The reason for this is that the generator must have some seed of randomness, otherwise it will simply learn to reproduce the same examples that are being input.~~

Currently the input to the Generator network is just a vector of normally distributed noise. This is consistent with many common GAN implementations. I moved to this implementation rather than the "masking" implementation from the "Fake is the new real" paper and "MaskGAN" because it is simpler, and allows me to eliminate some complexity that I haven't shown is actually useful.

The training objective is to minimize the "Earth movers distance" (aka Wasserstein distance) between the distribution of real data and "fake" data. This is the training objective proposed by the paper "Wasserstein generative adversarial networks" as it (supposedly) avoids many of the problems with traditional GANs such as mode-collapse. I got some reference code from this repository (and you may find many other useful GAN implementations there, WGAN-GP is one I wanted to try but don't quite understand yet): https://github.com/eriklindernoren/PyTorch-GAN

Feature Encoding

The training data (network packets) consists of features which are either categorical or numeric. The input to our network can be only numeric, so we must convert these features into suitable numeric representations.

For those features which are categorical in nature, I encode the integer corresponding to their category as a binary vector. This was chosen instead of a one-hot representation because it allows for the network to be more dense (have fewer parameters) and thus a decreased training time. Furthermore this choice is supported by a paper (Potdar) comparing feature encodings on a classification task which showed that one-hot encoding and binary-encoding of classes performed nearly identically.

For the numeric features (which are conveniently all non-negative), I have simply scaled them relative to the maximum values in the dataset (per feature). When decoding integer features, we scale them back to the original range and round to the nearest integer.

Output Evaluation Methods

Evaluating the output of a GAN is one of the challenges of making such a "fake data" framework in the first place, but the paper "Pros and cons of GAN evaluation measures"

provides some good ideas. I've chosen two of these metrics to implement in my code thus far, described below.

Number of Statistically Different Bins: Gather a dataset of N real and N fake examples. Use K-Means clustering to cluster the real examples into K clusters. Then determine which clusters the fake examples fall in to. If the fake distribution closely matches the real distribution, then the difference between the number of real and fake samples that fall into each cluster should be statistically insignificant. This method easily allows one to identify mode collapse or mode dropping.

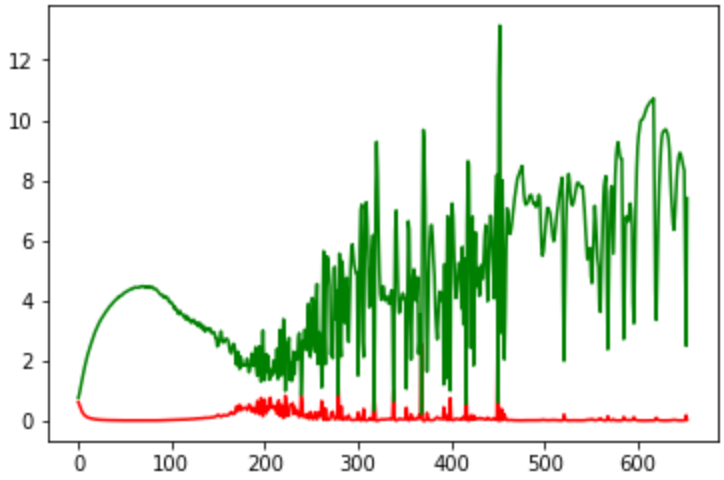
Wasserstein Critic / Wasserstein Distance Approximation: Train a classifier *f* (typically a neural network with clipped weights such that it has a bounded derivative) to output values close to 1 for real samples and 0 for fake samples. The metric is then calculated as follows:

$W(x_{real},x_{fake}) = (1/N)sum(f(x_{real})) - (1/N)sum(f(x_{fake}))$

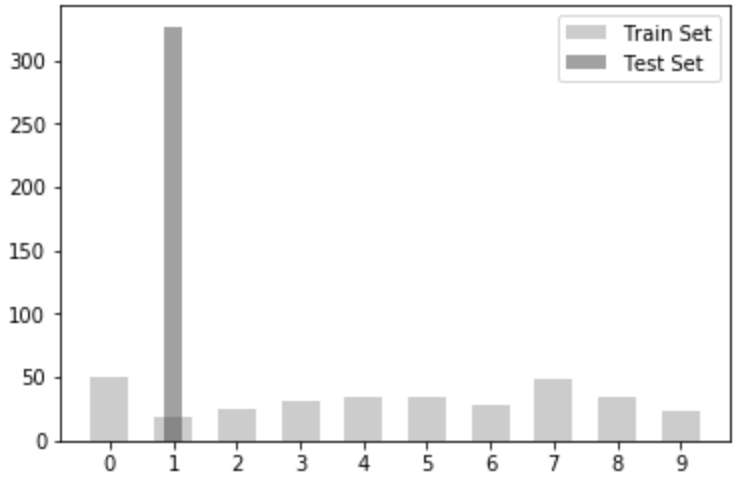
A good score is close to 0 (critic can not easily distinguish between real and fake), a bad score is close to 1 (trivially distinguishable).

I did not use the raw generator output as input to either of these metrics. Instead I use a latent representation produced by a GRU network, as these have shown to be very good at mapping semantically different sequences into lower dimensional vectors that can then be used as the inputs to classifiers or other algorithms which benefit from a lower dimensional representation.

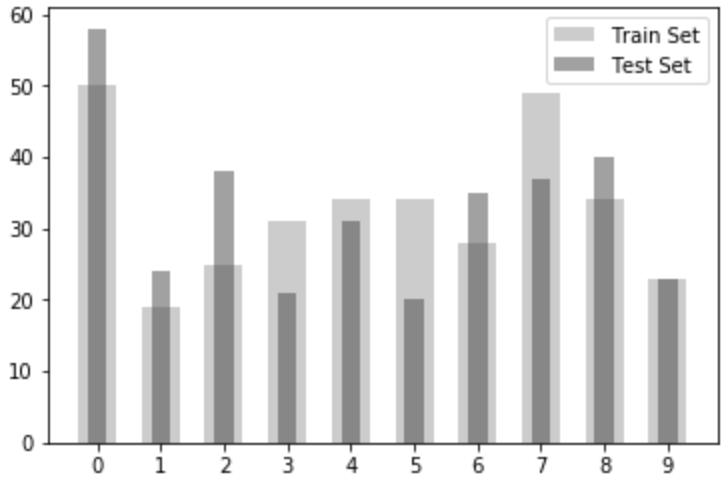
This "mapping of a sequence into a lower dimensional vector" seems initially suspect and unintuitive, but it is exactly what the generator (or any recurrent neural network) is doing internally; the latent representation is simply the hidden state of the network which is normally not exposed to the user or output directly. In the case of the generator, the hidden state is a very high dimensional vector (higher than the number of input features) since we wish to extract unseen information from the sequence. In the case of evaluation, we reduce the dimensionality for easier discrimination and computational efficiency. Additionally, euclidean distance between sequences doesn't necessarily make sense and so it can't be considered directly, but the euclidean distance between latent representations does actually mean there is a semantic difference between the sequences mapped to those points [citation needed, I'm assuming most of this because it's what people have done in other papers eg. "Fake is the new real" but they don't explain the motivation, it's just a "we use a latent representation because it works" kind of deal].



This is a learning curve from a short GAN training session which quickly fell in to mode-collapse. You can see that early in the training, the generator (green) loss dropped quickly and likewise the discriminator loss rose. However, the discriminator loss dropped shortly after since it recognized that the outputs being produced by the generator were fake, and the generator completely failed to recover for the rest of the session. This learning curve was from the session which produced the below bin-comparison. In this case the generator was always producing the same output, and so the discriminator easily recognized it as fake.



The above image is what you might see with a particularly bad case of mode-collapse. In this case (mode collapse), the generator is always producing the same output which it has learned to fool the discriminator with, and so logically all of it's output falls into the same bin.



The above image is what you should expect to see with a particularly good bin-comparison. In fact the train and test sets in this image are drawn from the same distribution, but the sample size was small which explains the variance.

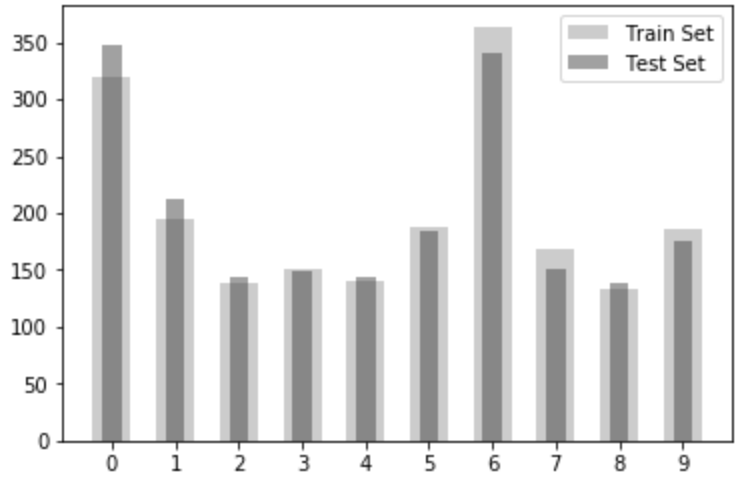
WaveGAN Experiment

I performed a short experiment to test whether my procedure and evaluation methods were useful with a simple and easily human-understandable dataset: sine waves. I created a data set of sine waves all of unit amplitude and the same frequency, but with a random phase shift for each training example. I then used this dataset as the “training set” for the very same PacketGAN framework described previously. That code is general enough that a number of different types of sequential datasets could be substituted for testing.

Training seemed to go well, the Generator loss was low (and stayed low, strangely). Below are the somewhat perplexing outputs.

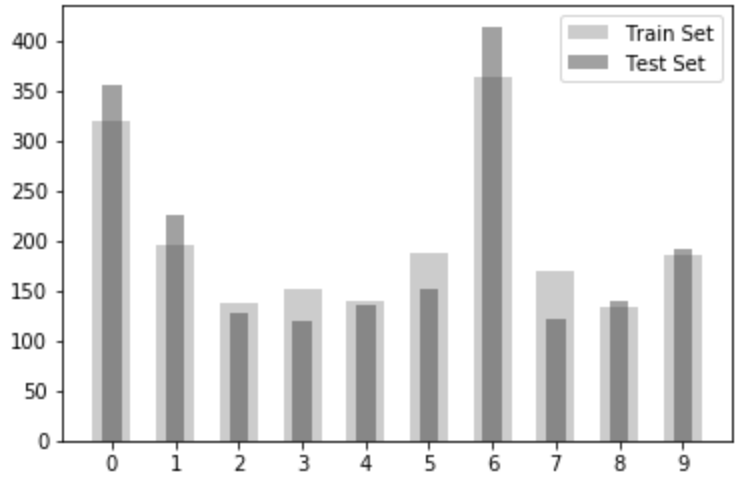
First, the cluster binning results for the real training & test sets.

Training set:



Next the cluster binning results for the generated (fake) data from the trained Generator (my chart labels are incorrect below. The “test set” is actually the generated set). The clustering looks to be nearly the same as the real data. That seems great!

Generated set:

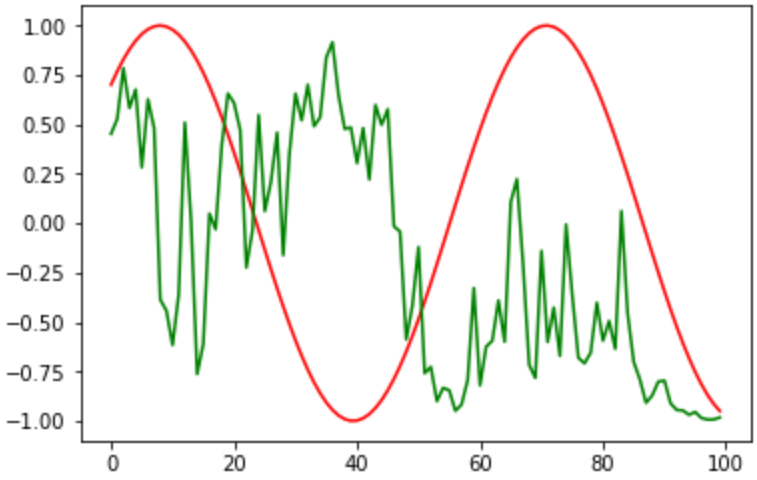


Furthermore, our Wasserstein Distance metric and Classification Accuracy metrics are near the best possible values (zero and 0.5 respectively). So we’ve succeeded in the best possible way, right?

Evaluation:
Wasserstein Distance: 0.00022685853764414787
Classification Accuracy: 0.5

Well, when we compare the real sine wave (red) to the generated sine wave (green), a human can trivially see that they are not nearly the same! Not all is lost because it does appear that the generator learned to mimic the “up and down” motion of the wave in a very noisy way, but it leaves a lot to be desired.

Actual output:



So how can this happen? Our metrics tell us the result is perfect but we can clearly see that it is not. Well, I'm thinking that the problem must lie in the latent space representation used to prepare the input for each of these metrics. Both of them use a GRU network to map a sequence of data to a flat (vector) representation for clustering and classification tests. So it seems that in this particular case, both of these waves are getting mapped to very similar representations where in my previous experiments (see the "Latent Space Classifier" notebook) I had not encountered this. So the next step in this particular experiment would be to determine if that is in fact the cause of these mistakenly good-looking metrics, and find a way to correct them. Ideally we want metrics that can tell us if the distribution of the generated data (which is not often easily visualized or understandable eg. in the case of network packets) factually does match the distribution of the training data. This is a hard problem, and is an open area of research.