# A Communication Channel Density Estimating Generative Adversarial Network

Aaron Smith, and Joseph Downey
Cognitive Signal Processing Branch
NASA Glenn Research Center
Cleveland, OH 44135
{aaron.smith, and joseph.a.downey}@nasa.gov

***Abstract*—Autoencoder-based communication systems use neural network channel models to backwardly propagate message reconstruction error gradients across an approximation of the physical communication channel. In this work, we develop and test a new generative adversarial network (GAN) architecture for the purpose of training a stochastic channel approximating neural network. In previous research, investigators have focused on additive white Gaussian noise (AWGN) channels and/or simplified Rayleigh fading channels, both of which are linear and have well defined analytic solutions. Given that training a neural network is computationally expensive, channel approximation networks—and more generally the autoencoder systems—should be evaluated in communication environments that are traditionally difficult. To that end, our investigation focuses on channels that contain a combination of non-linear amplifier distortion, pulse shape filtering, intersymbol interference, frequency-dependent group delay, multipath, and non-Gaussian statistics. Each of our models are trained without any prior knowledge of the channel. We show that the trained models have learned to generalize over an arbitrary amplifier drive level and constellation alphabet. We demonstrate the versatility of our GAN architecture by comparing the marginal probability density function of several channel simulations with that of their corresponding neural network approximations.**

***Index Terms*—machine learning, neural network, generative adversarial network, communication, channel approximation.**

## I. Introduction

The autoencoder-based communication system (see Fig. 1) has recently gained popularity in the research community. This system was first proposed in [1], where the authors interpret the transmitter as an encoder that maps incoming messages $s$ to transmitted symbols $x$. These symbols then pass through a channel function $h(x)$, after which the received symbols $y$ are decoded to recover the original message. Using this framework, a modulation scheme can be learned by training the transmitter and receiver networks to minimize a message reconstruction error metric. To do this, loss gradients are passed backwardly from the output layer of the receiver to the input layer of the transmitter.

In practice, the gradients of the physical channel are unknown and this prevents the transmitter network from receiving updates during training. This problem can be circumvented by assuming channel models with known analytic expressions, such as additive white Gaussian noise (AWGN) and Rayleigh fading channels. However, these channels fail to justify the

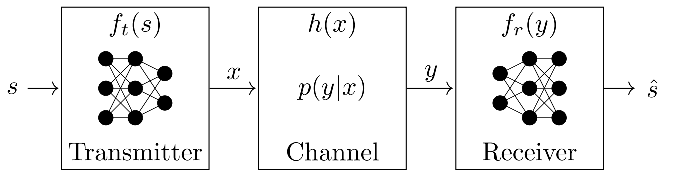


Fig. 1. An autoencoder-based communication system.

increased computational cost involved with training an autoencoder system. Ideally, the autoencoder system would be deployed in communication scenarios where no appropriate closed form representation exists. In these environments, the autoencoder could be used to learn unique modulation schemes that improve standard metrics such as Bit Error Rate (BER) or data throughput.

A solution to the missing channel gradients is to first approximate the channel response using a neural network. The neural network channel model can then act as a surrogate for the physical channel when training the autoencoder system. An early example of this method was demonstrated in [2], where the authors minimize the Mean Squared Error (MSE) between the real channel response and the response of the neural network model. This method results in a neural network that approximates the means of the conditional distributions of the channel. In [3], the authors trained a stochastic channel model using a Generative Adversarial Network (GAN). Though they only demonstrated this on AWGN and Rayleigh fading channels, they did show that the stochastic model could be used to train an autoencoder system. In [4], the authors used a GAN to approximate a stochastic channel that included non-linear distortions and non-Gaussian statistics. In general, their results demonstrate the potential in using a GAN to learn a difficult channel, but the channel contains no memory effects and the marginal Probability Density Function (PDF) of the learned channel model differs from the PDF of the simulated channel.

In this work, we propose and evaluate a new GAN architecture that can learn non-linearities, memory effects, and non-Gaussian statistics. In Section II, we describe our channel simulations, GAN architecture, and various implementation details. In Section III, we present and discuss our evaluation of the GAN. We provide concluding remarks in Section IV.

## II. Methods

In Subsection II-A, we describe the four channel simulations that are used to test the versatility of the GAN. Each of our channels can potentially contain Intersymbol Interference (ISI), due to the combination of a Root Raised Cosine (RRC) pulse-shaping filter and a non-linear power amplifier. Additionally, tight filtering of the signal, to ensure compliance with a spectral mask, causes the channel to be dispersive and provides another source of ISI. We also include a Finite Impulse Response (FIR) based multipath channel, as well as an uncorrelated phase noise channel. In Section II-B, we provide a detailed description of the GAN architecture. In Section II-C we discuss several aspects of our implementation such as: the channel sounding distribution, extending the generator input to support multiple symbols, data feeding, and hyper-parameter selection.

### A. Channel Models

In the autoencoder-based communication system (see Fig. 1), the "channel" includes all the processes between modulation and demodulation. Therefore, our channel simulations contain elements, such as pulse shaping, that are not traditionally considered part of the channel. We assume that the transmitter is using an amplitude and phase-shift keying modulation scheme such that symbols $x \in \mathbb{C}$. In this work, we construct channel models that present unique challenges to a traditional communication system. These include non-linearities induced by a transmit amplifier, dispersive channel elements such as a high-order spectral masking filter, multipath, and non-Gaussian statistics.

Fig. 2, shows the block diagram of our four channel simulations. We identify each channel by its subfigure label. Channel $h_a$ represents our most basic channel model and serves as the basis for each of the other channels. This channel begins by interpolating the incoming symbol stream using a 129 tap RRC pulse-shaping filter, with roll-off factor $\alpha$. After pulse shaping, the resulting sample stream is passed through a memoryless Saleh amplifier model, which is shown in Fig. 3. After amplification, we apply an AWGN process, perform matched filtering, and recover the received symbols. In each of our channel models, we assume that the proper procedures are in place for carrier removal and symbol timing acquisition. The channels contain an implicit linear low-noise receive amplifier with a constant gain factor.

In channel $h_b$, we simulate an environment where a transponder seeks to maximize throughput for a given bandwidth and utilizes a bandpass filter to ensure compliance with a spectral mask. A sharp filter cutoff would maximize the users usable spectrum, but can also induce a frequency-dependent group delay due to a large filter order. We simulate this with an eighth-order lowpass Infinite Impulse Response (IIR) filter. In Fig. 4 we show the filter's normalized amplitude response and overlay the normalized Power Spectral Density (PSD) of a Binary Phase Shift Keying (BPSK) signal at two different operating points. In Fig. 5, we show the group delay of the filter as a function of frequency. In channel $h_c$ we evaluate

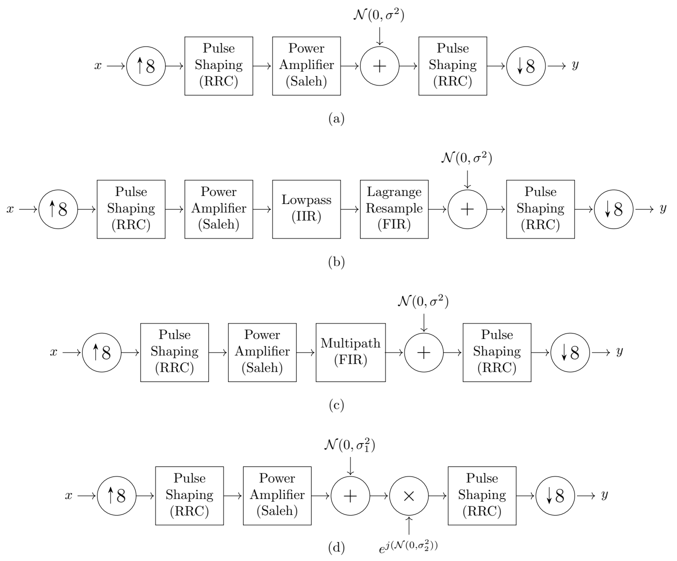


Fig. 2. A block diagram of the four channel simulations used in this work.

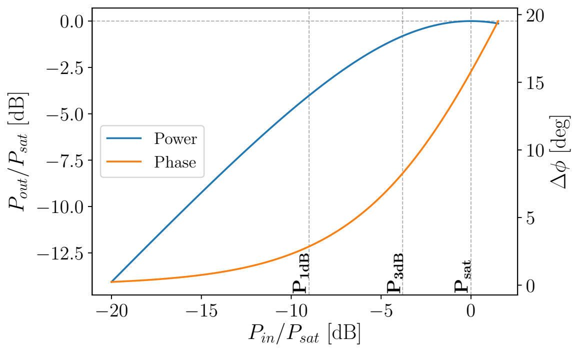


Fig. 3. The amplitude and phase response for our normalized Saleh amplifier model.

a FIR based multipath model with constant taps and channel $h_d$ demonstrates a non-Gaussian distribution by applying an uncorrelated phase noise process.

### B. GAN Architecture

The purpose of the GAN is to train a generator that approximates the output of an arbitrary channel such that they have the same conditional PDF. In image generation, several researchers have investigated similar problems, where a conditioning input stimulates a distribution of possible outputs. In this work, we leverage that body of research by adapting a modern GAN architecture to our channel estimation problem.

In [5], the authors train a generator that uses an input image and a vector of latent variables $\mathbf{z} \sim p(\mathbf{z})$ to produce a distribution of output images. In that work, the authors attempt to enforce a robust utilization of the latent vector by making the connection between the output and the latent code invertible. Their model combines the information cycles of a Variational Autoencoder (VAE)-GAN [6], with a dual cycle—inspired by the work in [7], [8], and [9]—which reconstructs

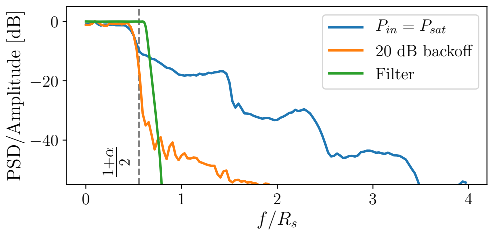


Fig. 4. The amplitude response of the spectral masking filter used in channel $h_b$ and the normalized PSD of a BPSK signal at two operating levels.

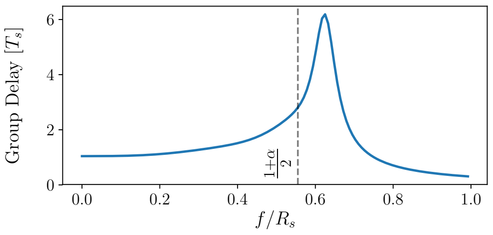


Fig. 5. Group delay of the spectral masking filter used in channel $h_b$.

the latent vector. Their claim is that this bijective approach assists in preventing mode collapse.

In Fig. 6a, we show our GAN training architecture. In this work, we adopt the dual cycle concept presented in [5], but modify the components of their architecture to perform channel approximation. In the upper portion of Fig. 6a, we implement the VAE cycle, which generates an output reconstruction loss metric $\ell_y$ that is used to stabilize the training process. In the lower portion of Fig. 6a, we implement the latent regression cycle, which similarly produces a latent reconstruction loss metric $\ell_z$. In the latent case, we use a sample estimate of $\mathbf{z}$ as opposed to the point estimate used in reconstructing $y$. On the right side of Fig. 6a, we show the discriminator signals. We use $f$ to signify fake inputs and $fe$ when the input utilized an encoded noise vector. In contrast to the implementation described in [5], we provide both the encoder and discriminator networks with our conditioning input. This gives both units the ability to either encode or discriminate based on knowledge of the transformation from $\mathbf{x}$ to $y$.

In this work, we construct our objective functions following the least squares approach [10]. Our discriminator objective function

$$\min_D V(D) = \mathcal{L}_d + \mathcal{L}_d^f + \mathcal{L}_d^{fe} \tag{1}$$

performs the standard binary classification problem between real and fake samples, with the addition of the encoded samples, as was done in [6]. We capture the classification error of real examples as

$$\mathcal{L}_d = \mathbb{E}_{\mathbf{x},y\sim p(\mathbf{x},y)}[(D(\mathbf{x},y)-1)^2] \tag{2}$$

with a real label target of one. The corresponding error due to fake samples is

$$\mathcal{L}_d^f = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}),\mathbf{z}\sim p(\mathbf{z})}[(D(\mathbf{x},G(\mathbf{x},\mathbf{z})))^2] \tag{3}$$

where the target label for fake samples is equal to zero. The last element in our discriminator objective function

$$\mathcal{L}_d^{fe} = \mathbb{E}_{\mathbf{x},y\sim p(\mathbf{x},y)}[(D(\mathbf{x},G(\mathbf{x},E(\mathbf{x},y))))^2] \tag{4}$$

represents the error associated with fake examples that were constructed using knowledge of the transformation between $\mathbf{x}$ and $y$.

The generator objective function

$$\min_G V(G) = \mathcal{L}_g^f + \mathcal{L}_{ge}^{fe} + \lambda_y\mathcal{L}_y + \lambda_z\mathcal{L}_z \tag{5}$$

combines the losses associated with discriminator labeling and the reconstruction losses from the symbol and latent variables. The first two components of the generator objective

$$\mathcal{L}_g^f = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}),\mathbf{z}\sim p(\mathbf{z})}[(D(\mathbf{x},G(\mathbf{x},\mathbf{z}))-1)^2] \tag{6}$$

$$\mathcal{L}_{ge}^{fe} = \mathbb{E}_{\mathbf{x},y\sim p(\mathbf{x},y)}[(D(\mathbf{x},G(\mathbf{x},E(\mathbf{x},y)))-1)^2] \tag{7}$$

are the standard discriminator label losses, though we again include the encoded version. The last two components of the generator objective relate to the reconstruction error where

$$\mathcal{L}_y = \mathbb{E}_{\mathbf{x},y\sim p(\mathbf{x},y)}\|y - G(\mathbf{x},E(\mathbf{x},y))\|_1 \tag{8}$$

is the expected $L_1$ distance between the true $y$ and the reconstructed $\hat{y}$ produced by the generator, using latent samples provided by the encoder network. This is the loss term associated with the VAE cycle discussed earlier and is approximated using the $\ell_y$ signal in Fig. 6a. The latent regression cycle is represented as

$$\mathcal{L}_z = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}),\mathbf{z}\sim p(\mathbf{z})}\|\mathbf{z} - \mathbb{E}[E(\mathbf{x},G(\mathbf{x},\mathbf{z}))]\|_1 \tag{9}$$

where we find there expected $L_1$ distance between an input latent vector and a vector of sample estimates from the encoder output. In Fig. 6a, we approximate this using the signal $\ell_z$.

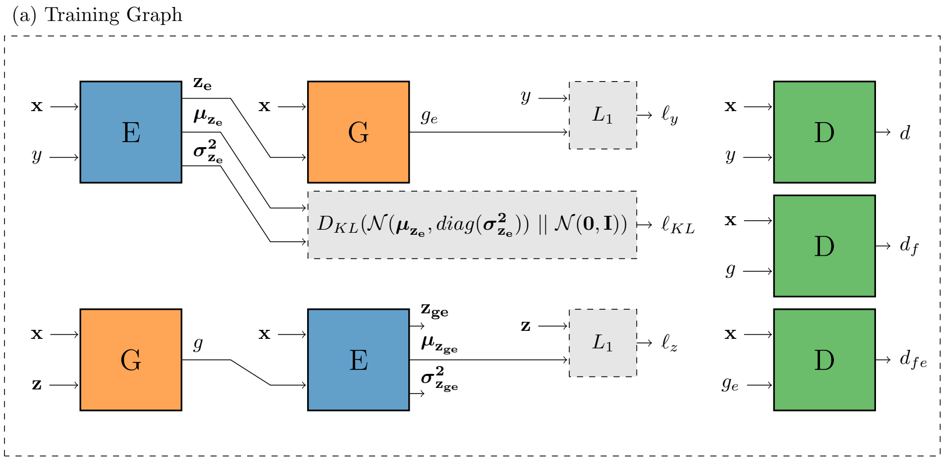
The encoder objective function

$$\min_E V(E) = \mathcal{L}_{ge}^{fe} + \lambda_y\mathcal{L}_y + \lambda_z\mathcal{L}_z + \lambda_{KL}\mathcal{L}_{KL} \tag{10}$$

is similar to the generator, though it excludes $\mathcal{L}_g^f$ and adds the Kullback-Leibler (KL)-divergence from the encoder output distributions to a vector of normal random variables as

$$\mathcal{L}_{KL} = \mathbb{E}_{\mathbf{x},y\sim p(\mathbf{x},y)}[\mathcal{D}_{KL}(E(\mathbf{x},y)\|\mathcal{N}(\mathbf{0},\mathbf{I}))] \tag{11}$$

where $\mathcal{D}_{KL}(p\|q) = -\int p(z)\log\frac{p(z)}{q(z)}dz$. Given that the generator is a deterministic network, we produce a stochastic output by sampling the latent space according to a prior distribution $p(\mathbf{z}) \sim \mathcal{N}(\mathbf{0},\mathbf{I})$. Therefore, during training, we use the expected KL-divergence to encourage the encoder to produce samples with a distribution similar to those used during inference. We calculate the KL-divergence in our training graph using the signal $\ell_{KL}$ in Fig. 6a.


(a) Training Graph
x
y
E
$\mathbf{z_e}$
$\boldsymbol{\mu}_{\mathbf{z_e}}$
$\boldsymbol{\sigma}^2_{\mathbf{z_e}}$
x
G
$g_e$
$y$
$L_1$
$\ell_y$
$D_{KL}(\mathcal{N}(\boldsymbol{\mu}_{\mathbf{z_e}}, diag(\boldsymbol{\sigma}^2_{\mathbf{z_e}})) \,||\, \mathcal{N}(\mathbf{0},\mathbf{I}))$
$\ell_{KL}$
x
z
G
$g$
x
E
$\mathbf{z_{ge}}$
$\boldsymbol{\mu}_{\mathbf{z_{ge}}}$
$\boldsymbol{\sigma}^2_{\mathbf{z_{ge}}}$
z
$L_1$
$\ell_z$
x
y
D
$d$
x
$g$
D
$d_f$
x
$g_e$
D
$d_{fe}$

(b) Loss Components

$$\begin{aligned}
\mathcal{L}_d &= \overline{(d-1)^2}\\
\mathcal{L}_d^f &= \overline{d_f^2}\\
\mathcal{L}_d^{fe} &= \overline{d_{fe}^2}\\
\mathcal{L}_g^f &= \overline{(d_f-1)^2}\\
\mathcal{L}_{ge}^{fe} &= \overline{(d_{fe}-1)^2}\\
\mathcal{L}_y &= \overline{\ell_y}\\
\mathcal{L}_z &= \overline{\ell_z}\\
\mathcal{L}_{KL} &= \overline{\ell_{KL}}
\end{aligned}$$

(c) Optimizer Losses

$$\begin{aligned}
\mathcal{L}_D &= \mathcal{L}_d + \mathcal{L}_d^f + \mathcal{L}_d^{fe}\\
\mathcal{L}_G &= \mathcal{L}_g^f + \mathcal{L}_{ge}^{fe} + \lambda_y\mathcal{L}_y + \lambda_z\mathcal{L}_z\\
\mathcal{L}_E &= \mathcal{L}_{ge}^{fe} + \lambda_y\mathcal{L}_y + \lambda_z\mathcal{L}_z + \lambda_{KL}\mathcal{L}_{KL}
\end{aligned}$$

Fig. 6. (a) A diagram of the GAN training graph. (b) Loss component approximation from graph signals. (c) A summary of the optimizer losses.

In Fig. 6b, we show our approximations of the loss components, where we estimate the expected value by averaging over a batch of examples. In Fig. 6c, we combine the loss components of Fig. 6b, to estimate the objective functions in equations (1), (5), and (10). We then use these signals in three separate optimizers that are jointly minimized during training.

In Fig. 7, we diagram the generator, discriminator, and encoder networks. Each of these elements utilize a symbol energy calculation, denoted by $|\cdot|$. Hidden and output layers follow the same notation, for example, "FC : 128 : Lin" signifies that the layer contains 128 fully connected neurons with a linear activation function. There is an implied concatenate operation whenever a layer is shown to have multiple inputs. We implement the variational aspect of the encoder by interpreting the output of two separate layers as vectors of mean and variance values, which are then used to produce noise samples. Our actual implementation returns $\log(\sigma^2)$, as is common in literature, with the values being exponentiated as needed.

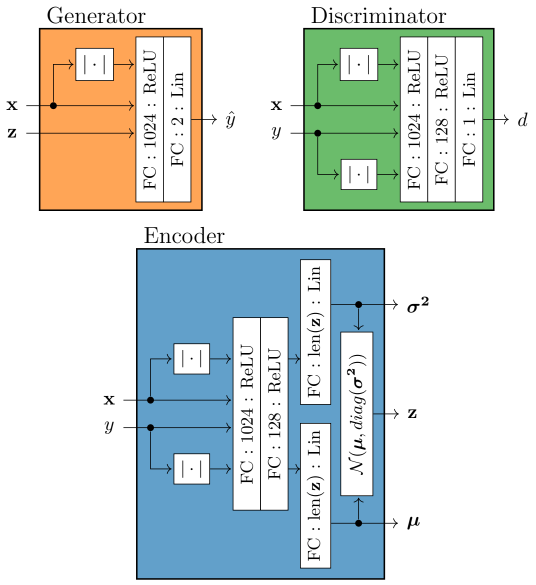


Fig. 7. A diagram of the Generator, Discriminator, and Encoder networks.

### C. Implementation Details

Each of our channels (see Fig. 2), include a non-linear amplifier that will induce ISI when the amplifier is driven into compression. Additionally, channel $h_b$ will cause ISI due to the frequency-dependent group delay associated with the IIR spectral masking filter. In an ideal case, without ISI, the output symbol is only dependent on the time aligned input symbol. However, ISI causes a dependency on the surrounding transmit symbols. Due to this, we construct training examples $\mathbf{x} \in \mathbb{C}^N$, which are centered on the current time-step transmit symbol. For example, if transmit symbol $x_i$ results in the time aligned channel output $y_i$, then our example vector would be $\mathbf{x} = [x_{i-\frac{N-1}{2}}, ..., x_i, ...x_{i+\frac{N-1}{2}}]$, where $N$ is odd integer.

To construct a training set $\{(\mathbf{x} \in \mathcal{X} \subset \mathbb{C}^N, y \in \mathcal{Y} \subset \mathbb{C})\}$, which defines a joint probability $p(\mathbf{x}, y)$, we must choose a sounding distribution $p(\mathbf{x})$. Our sounding distribution should canvas the usable complex domain such that the generator produces realistic results for an arbitrary set of input symbols. We assume that the input symbols are independent and identically distributed and use a combination of the following $p(x)$ distributions:

- Uniform magnitude - $x \sim \mathbb{U}(0,a) \times e^{j\mathbb{U}(0,2\pi)}$
- Uniform disk - $x \sim \sqrt{\mathbb{U}(0,a)} \times e^{j\mathbb{U}(0,2\pi)}$
- Gaussian - $x \sim \mathcal{N}(0,\sigma^2) + j\mathcal{N}(0,\sigma^2)$.

Using a uniform random variable for the magnitude component of a complex value results in a higher sample density near the origin of the complex plane due to the area element $dA = 2\pi r dr$. Normalizing such that the complex plane is sampled with uniform density results in a bias toward high energy symbols in the training set. As mentioned, we only train on the usable complex plane, which was determined by the maximum output power of our amplifier model.

In this work, we use continuous random variables when building a training set to ensure that the generator generalizes

TABLE I
HYPER-PARAMETERS USED IN THIS WORK.

| Parameter | Value |
|---|---|
| Batchsize | 250 |
| $len(\mathbf{z})$ | 6 |
| $\lambda_y$ | 2.0 |
| $\lambda_z$ | 1.0 |
| $\lambda_{kl}$ | 0.008 |

to an arbitrary set of inputs. Due to this, the generator will not have the opportunity to train on sequences of symbols with repeating elements. For instance, a BPSK-like modulation that randomly selects one of two symbol values. In the future, it may be worthwhile to test a more structured training set that includes common features like constant energy rings (PSK constellations) and sequences that draw symbols from a small set.

Using our $p(x)$ sampling function, we construct stimulus vectors of length $M$, where $M >> N$, and pass that through our channel model. Then, due to the filtering processes in our channels, we trim transient symbols and realign the stimulus and response vectors. We then develop training sets by slicing out examples $(\mathbf{x}, y)$ from the stimulus and response vectors at random indexes.

We have been describing symbols as complex values, but these are processed by the neural networks as 32-bit floating point values. This is done by simply concatenating the real and imaginary terms together. In the generator network, shown in Fig. 7, the final layer contains two neurons. These two neurons are interpreted as the real and imaginary components of the output symbol $y$.

In Fig. 6a, the diagram shows multiple instances of the generator, encoder, and discriminator elements. Tensorflow [11] similarly implements these as additional elements in the resulting graph structure. However, instances of the same element share a common set of weights and biases. Therefore, we interpret there to be a single generator, with the various outputs corresponding to different input signals. Similarly, we interpret there to be a single discriminator and encoder element.

Regardless of the channel model, we maintain the same graph structure. In Table I, we list the various parameters used during training. Weight updates are performed using the Adam optimization algorithm [12], which minimizes the losses summarized in Fig. 6c. In general, we based the length of the input vector $\mathbf{x}$, on the length of the pulse shaping filter. Given that we use a 129 tap RRC filter, with eight samples per symbol, we primarily used a $len(\mathbf{x})$ of 17 complex symbols, though we do investigate the impairments associated with shortening $\mathbf{x}$ in the next section.

## III. RESULTS

After training, the generator network is extracted from the training graph and used as shown in Fig. 8. During inference, the generator uses a symbol sequence $\mathbf{x}$ and noise samples $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ to produce a guess of the channel output. We

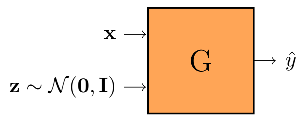


Fig. 8. Operation of the Generator during inference.

evaluate the generators by comparing the marginal PDF of a channel $p(y) = \sum_{\mathbf{x}} p(y|\mathbf{x})p(\mathbf{x})$ with that of a trained generator. We approximate the PDFs using a two-dimensional histogram and present those results visually. For these comparisons, we choose input distributions $p(x)$ that should be familiar to the reader, i.e. a Quadrature Phase-Shift Keying (QPSK) constellation, though the generator will respond to an arbitrary set of inputs and has not learned anything specific to these standard constellations.

For visualization, we define the following modulation alphabets: BPSK, QPSK, 8-PSK, 16-QAM, 16-APSK, 32-APSK, and 64-QAM. We construct transmit signals by drawing symbols from a constellation with equal probability. Normalized signal power is calculated as

$$\frac{P_{\mathbf{x}}}{P_{sat}} \; [dB] = 20 \log_{10}(\beta) - 10 \log_{10}(PAPR) \qquad (12)$$

where $\beta \in (0, 1]$ is an amplitude scale factor and $PAPR$ is the Peak-to-Average Power Ratio of the constellation.

First, we analyze a generator approximation of channel $h_a$, with a RRC roll-off factor $\alpha = 0.35$ and a normalized AWGN noise power $P_n/P_{sat} = -35$ dB. In the upper portion of Fig. 9, we feed a QPSK signal into both the channel and the generator. As $\beta$ is increased, the power amplifier is driven deeper into saturation which causes ISI and distorts the output. This demonstrates that the generator has learned to approximate the channel response over a range of input power levels. In the lower portion of Fig. 9, we hold $\beta$ constant, but vary the input constellation, showing that the generator has learned an appropriate response to an arbitrary modulation alphabet.

In Fig. 10, we show that a generator is able to learn various roll-off factors and AWGN powers. In this case, we train separate generators for each of the selected parameters. This is in contrast to the previous test, where one generator had generalized over an arbitrary input. We found that the generator generally performed better in low signal-to-noise ratio (SNR) channels when a small number of hidden layers was used.

Our final analysis of channel $h_a$ tests the impact of varying the input vector length $len(\mathbf{x})$. We test this using an approximation of the total variation distance $\delta(P, Q) = \frac{1}{2}\|P - Q\|_1$, where $P$ and $Q$ are binned estimates of the channel and generator marginal PDFs. Channel parameters are held constant, with $\alpha = 0.1$ and $P_n/P_{sat} = -35$ dB. The input is a 16-QAM signal with $\beta = 0.4$. We train a separate generator for each $len(\mathbf{x}) \in \{1, 3, 5, 17\}$ and normalize the results based on $len(\mathbf{x}) = 1$. In Table II, we show that the majority of our gain comes from the inclusion of just a few

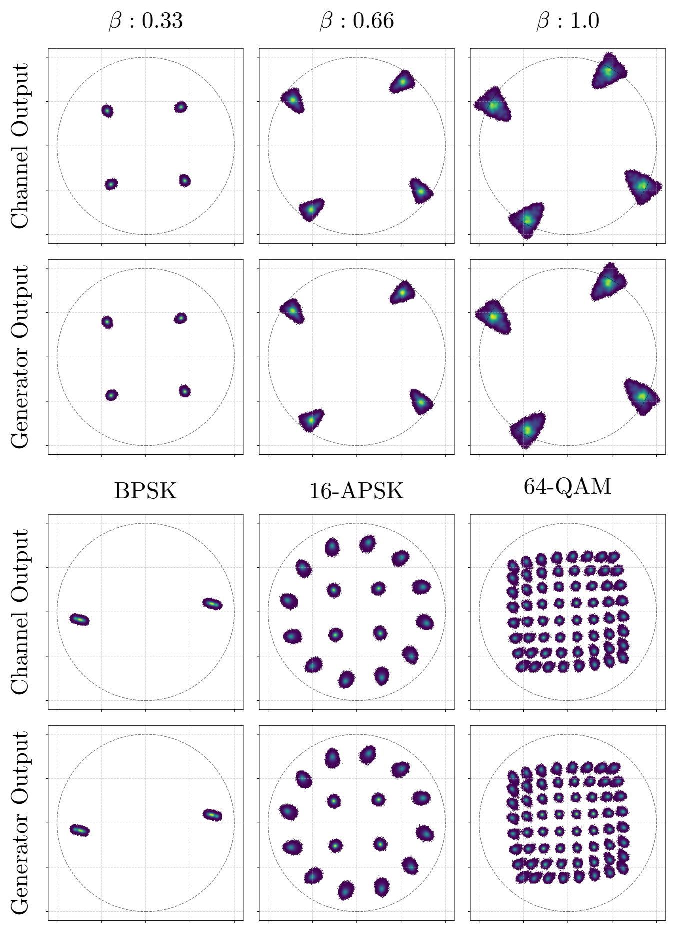


Fig. 9. An evaluation of the $h_a$ channel approximation results. We show that the generator has learned to produce realistic outputs over arbitrary input power levels and modulation alphabets.

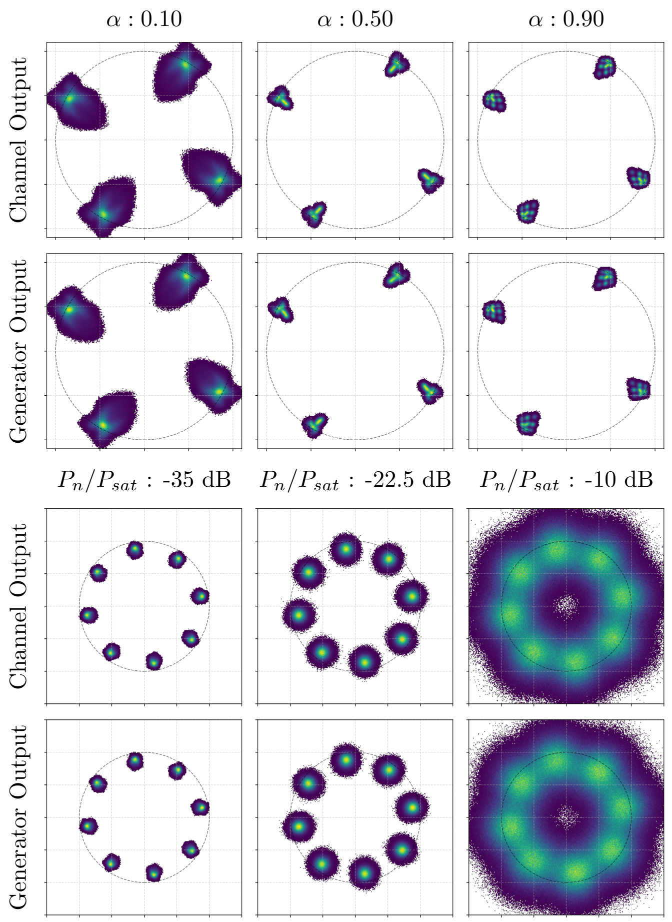


Fig. 10. Using channel $h_a$, we demonstrate the versatility of the GAN architecture by varying the RRC roll-off factor and channel AWGN power.

TABLE II
APPROXIMATED TOTAL VARIATION DISTANCE BETWEEN THE CHANNEL AND GENERATOR PDFS AS A FUNCTION OF $len(\mathbf{x})$.

| len($\mathbf{x}$) | $\delta$ |
|---|---|
| 1 | 0 dB |
| 3 | -4.79 dB |
| 5 | -6.81 dB |
| 17 | -7.78 dB |

neighboring symbols, though we expect that the neighboring symbols become increasingly important as the amplifier is driven toward saturation.

In Fig. 11, we analyze channel $h_b$, where a sharp spectral masking filter causes a frequency-dependent group delay (see Figs. 4-5). In this case, we use a small roll-off factor $\alpha = 0.1$ to simulate the scenario where high spectral efficiency is desired. Additionally, we set $P_n/P_{sat} = -35$ dB, $len(\mathbf{x}) = 17$, $\beta = 0.5$. In Fig. 11, we can see that the filter has caused an AWGN like distribution around the symbols. In an eye diagram, we can also see this as a closing of the eye. The results of our final two channels $h_c$ and $h_d$ are shown in Figs. 12 and 13. These results show that the GAN is able to learn both the multipath environment and an estimate of phase noise. To limit the number of figures, we forgo a demonstration over various roll-off factors, AWGN power levels, and $\beta$ scale factors—for channels $h_b$, $h_c$, and $h_d$. However, as was shown for channel $h_a$, those same generalizations apply to each of the other learned channels.

In this work, the majority of the results are presented as a visual comparison of probability densities. This qualitative analysis is useful for a quick verification of a GAN architecture, but without a quantitative metric, it is difficult to directly compare competing architectures. In Table II, we used a distance metric to compare the results of generators that had been trained using various input vector lengths. In that case, we could directly compare distance results because we maintained all aspects of the input distribution $p(x)$, the conditional channel distribution $p(y|\mathbf{x})$, and approximating assumptions such as bin locations and the number of samples. However, a GAN should be able to learn a variety of channels over arbitrary inputs. This is a difficult capability to quantify and would likely involve the standardization of a set of channel simulations.

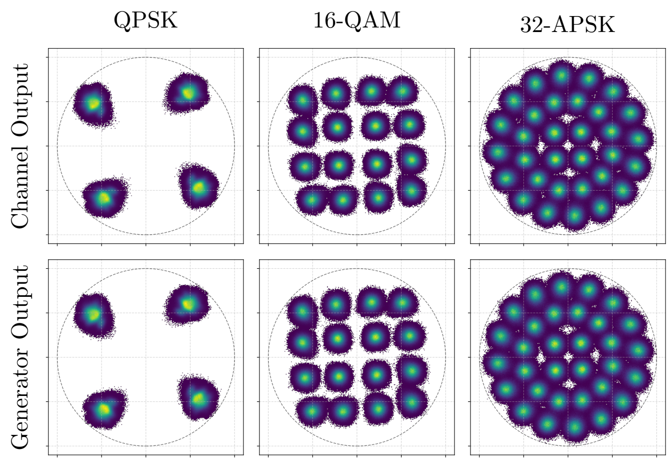


Fig. 11. An approximation of the dispersive channel $h_b$.

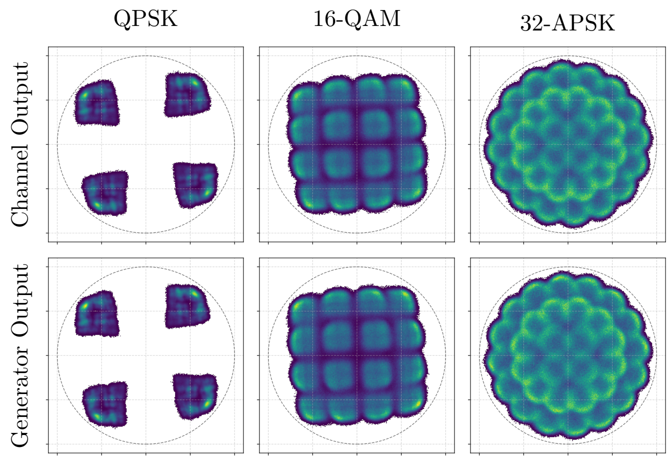


Fig. 12. An approximation of the multipath channel $h_c$.

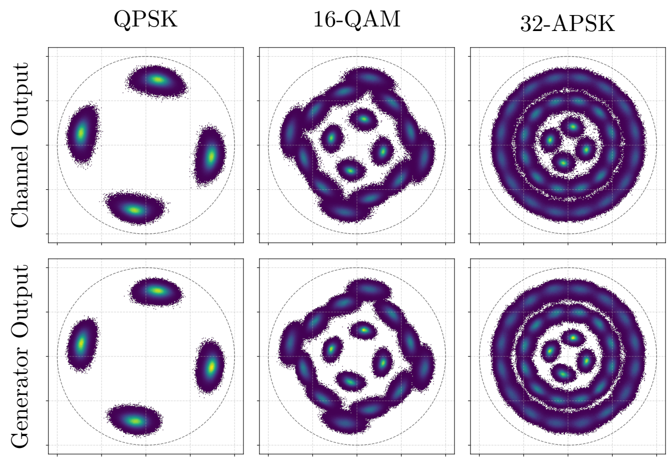


Fig. 13. An approximation of the uncorrelated phase noise channel $h_d$.

## IV. Conclusion

The autoencoder-based communication system presents a unique opportunity for a global optimization routine that combines previously independent processes. Passing loss gradients from a receiver to a transmitter can be accomplished by learning a neural network approximation of the physical channel. An ideal channel approximation algorithm should be capable of learning an arbitrary channel solely from channel measurements. Furthermore, the approximation algorithm should be capable of learning channels that are traditionally difficult to model, as this would enable an autoencoder system to operate in environments where no standard solution is available.

In this work, we use non-linearities, memory effects, non-Gaussian statistics to test the capabilities of a new GAN architecture. We present our results as a visual comparison between the marginalized PDFs of the channel and a trained generator. Our results show that the proposed GAN is able to train a generator that well approximates our four channel simulations. Furthermore, we show that a trained generator can produce high fidelity approximations of a channel over an arbitrary input power level and modulation alphabet.

## References

[1] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, Dec 2017.

[2] T. J. O'Shea, T. Roy, N. West, and B. C. Hilburn, "Physical layer communications system design over-the-air using adversarial networks," in *2018 26th European Signal Processing Conference (EUSIPCO)*, Sep. 2018, pp. 529–532.

[3] H. Ye, G. Y. Li, B. F. Juang, and K. Sivanesan, "Channel agnostic end-to-end learning based communication systems with conditional GAN," *CoRR*, vol. abs/1807.00447, 2018. [Online]. Available: http://arxiv.org/abs/1807.00447

[4] T. J. O'Shea, T. Roy, and N. West, "Approximating the void: Learning stochastic channel models from observation with variational generative adversarial networks," *CoRR*, vol. abs/1805.06350, 2018. [Online]. Available: http://arxiv.org/abs/1805.06350

[5] J. Zhu, R. Zhang *et al.*, "Toward multimodal image-to-image translation," *CoRR*, vol. abs/1711.11586, 2017. [Online]. Available: http://arxiv.org/abs/1711.11586

[6] A. B. L. Larsen, S. K. Sønderby, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," *CoRR*, vol. abs/1512.09300, 2015. [Online]. Available: http://arxiv.org/abs/1512.09300

[7] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," *CoRR*, vol. abs/1605.09782, 2016. [Online]. Available: http://arxiv.org/abs/1605.09782

[8] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, "Adversarially learned inference," *CoRR*, vol. abs/1606.00704, 2016. [Online]. Available: http://arxiv.org/abs/1606.00704

[9] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," *CoRR*, vol. abs/1606.03657, 2016. [Online]. Available: http://arxiv.org/abs/1606.03657

[10] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, and Z. Wang, "Multi-class generative adversarial networks with the L2 loss function," *CoRR*, vol. abs/1611.04076, 2016. [Online]. Available: http://arxiv.org/abs/1611.04076

[11] M. Abadi, A. Agarwal *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[12] J. B. Diederik Kingma, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980