Differentiable Channels Are All You Need

Corey D. Cooke
Jeremy Till

May 2022

Approved for public release. Distribution is unlimited.
DOCUMENT AVAILABILITY

Reports produced after January 1, 1996, are generally available free via OSTI.GOV.

Website: www.osti.gov/

Reports produced before January 1, 1996, may be purchased by members of the public from the following source:

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
Telephone: 703-605-6000 (1-800-553-6847)
TDD: 703-487-4639
Fax: 703-605-6900
E-mail: info@ntis.gov
Website: http://classic.ntis.gov/

Reports are available to DOE employees, DOE contractors, Energy Technology Data Exchange representatives, and International Nuclear Information System representatives from the following source:

Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831
Telephone: 865-576-8401
Fax: 865-576-5728
E-mail: report@osti.gov
Website: https://www.osti.gov/

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DIFFERENTIABLE CHANNELS ARE ALL YOU NEED

Corey D. Cooke
Jeremy Till

May 2022

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831
managed by
UT-Battelle LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725
ABSTRACT

Mobile wireless communication systems must adapt to and compensate for time-varying propagation paths. The process of performing compensation in the receiver for said effects is known as equalization. We seek an improved method for optimizing the reliability of said wireless links to minimize outage probability. A method based on machine learning, specifically deep learning, has the potential to allow for greater flexibility and robustness against unforeseen channel conditions. In this report we investigate the performance of a prior method in the literature with minor modifications.

1 INTRODUCTION

Next generation wireless systems (5G, etc) have an immense societal and economic value. Optimizing said systems will improve the quality of life for all. Advancements such as 5G allow a wider range of carrier frequencies and antenna configurations to be employed. However, these systems, particularly massive MIMO systems, use model-based signal detection and estimation methods and may experience suboptimal performance in certain adverse or time-varying channels. Deep learning can be utilized to learn the functions of encoder and decoder in a wireless transceiver.

This study explores the usage of deep neural networks to learn the functions of transmitter and receiver. In particular, we attempted to implement the method of Ye [1] with minor modifications. Section 2 covers previous work on the topic of machine learning and wireless communication. Section 3 outlines the technical details for code used in the study. Section 4 shows the dataset used for testing and how the study was implemented. Section 5 contains the results. Section 6 contains the final conclusions.

2 PREVIOUS WORK

Several studies have applied deep learning to waveform shaping and estimation in wireless systems [1–4]. In the broader machine learning literature, generative adversarial networks (GANs) have been utilized to generate realistic looking draws from a data distribution [5]. Applied to wireless systems, the generator can be configured to represent the channel and allows the receiver to learn the channel distribution from observations of received pilot symbols that travel over the real channel. We plan to study the ability of ML-based physical layer implementations to mitigate the harmful effects of multipath propagation and other channel impairments in wireless systems [6].

3 TECHNICAL APPROACH

3.1 Signal and System Model

Propagation of radio frequency (RF) signals in a mobile radio system involves many physical propagation paths where electromagnetic (EM) waves scatter off objects such as buildings, the ground/street, vegetation, vehicles, etc. The scattered waves from the multiple paths impinge on the receive antenna and can cause constructive of destructive interference according to the principle of wave superposition. As
well, one or both of the radio notes are often in motion during a communications session. This means that these propagation paths as well as their relative path gains are effectively random and varying with time.

Mathematically, these phenomena can be modeled by treating the environment as a random linear time-varying (RLTV) system [6]. Under this model, the propagation medium is modeled by an equivalent lowpass time-varying impulse response \( h(t, \tau) \), and the complex baseband received signal \( y(t) \) is computed by:

\[
y(t) = \int_{-\infty}^{\infty} h(t, \tau) x(t - \tau) d\tau + n(t),
\]

where \( x(t) \) is the transmitted complex baseband signal, and \( n(t) \) is AWGN distributed according to \( n(t) \sim \text{CN}(0, N_0) \), where \( \text{CN}(0, N_0) \) denotes the circularly-symmetric complex normal distribution with zero mean and variance \( N_0 \). The superposition integral in (1) can be thought of as a convolution (across lag variable \( \tau \)) with an impulse response that is changing with absolute time \( t \). In the case where the channel is not changing with time, the system would be a linear time-invariant (LTI) system and would just be expressed as \( h(\tau) \). (i.e. no dependence on \( t \) in the LTI case.)

Because the propagation environment is rapidly changing and difficult to predict a priori, \( h(t, \tau) \) is treated as a stochastic process. Usually it is assumed that \( h(t, \tau) \) is a wide-sense stationary (WSS) stochastic process with respect to \( t \) and that the system response for different values of lag \( \tau \) exhibit uncorrelated scattering (US); together these assumptions form what is called a WSSUS stochastic process.

In the case of a discrete-time system (i.e. any system using digital signal processing), we can concatenate time-sampled values of the transmitted and receive signals into vectors, i.e.

\[
\tilde{x} = [x(T) \ x(2T) \ \ldots \ x(KT)] \\
\tilde{y} = [y(T) \ y(2T) \ \ldots \ y(KT)]
\]

where \( T \) is the sample period in seconds. This means that \( \tilde{x}, \tilde{y} \in \mathbb{C}^K \). The real components of these vectors correspond to the in-phase channel (I) and the imaginary components correspond to the quadrature channel (Q). Because neural networks are usually formulated to work with real numbers, we will reshape the data into tensors \( x, y \in \mathbb{R}^{K \times 2} \) and treat the I and Q components each as real-valued channels.

In information theoretic terms, the channel is defined by the conditional probability density function (pdf) of the received vector conditioned on the transmitted vector, i.e. the channel pdf is given by \( p(y|x) \). This is the most general case and encompasses any possible perturbation that the channel can apply to the input signal, including EM propagation effects, additive noise, multiplicative noise, and nonlinearities. The proposed channel model in (1) is a subset of this in that it is a linear model with additive noise only, but it is representative of a wide range of practical cases. The modulation and coding scheme is represented by the pdf of the transmitted signal, i.e. \( p(x) \). The optimal modulation and coding scheme in terms of bitrate while providing for reliable communication is the one that maximizes the mutual information \( I(x; y) \), defined by

\[
I(x; y) = -\mathbb{E}[\log_2 p(y)] + \mathbb{E}[\log_2 p(y|x)],
\]

where \( \mathbb{E}[-] \) denotes expectation. For simple channels, such as AWGN with no multipath, the optimization problem in (4) can be solved analytically. However, the optimal modulation and coding scheme is, in general, unknown and requires a method for characterizing the channel pdf as well as a computationally tractable algorithm for performing the encoding and decoding operation in near real-time. This remains an open problem in the wireless communications community.
3.2 Neural Network Approach

Fundamentally a communications system operates as an autoencoder in the sense that the transmitter serves the role of the encoder, the receiver serves the role of the decoder, and the channel is a perturbation that is applied to the latent space. If one had a differentiable model for the channel, backpropagation could be performed directly and both the transmitter and receiver could be optimized by minimizing the end-to-end loss [4]. If the output layer of the decoder is using sigmoid activations (one bit per output neuron with no softmax), then the appropriate end-to-end loss would be binary cross-entropy between the transmitted bits and the sigmoid outputs.

For a codeword representing a fixed number of bits $N$ and codeword length $K$, a pair of feed-forward neural networks could be trained using the above procedure and approach a near-optimal solution if trained appropriately. However, this requires knowledge of the channel state information (CSI) at both the transmitter and receiver to perform the backpropagation operation. In terms of concepts we have discussed, the CSI would consist of knowledge of the finite impulse response (FIR) filter taps $h$, representing sampled values of $h(t, \tau)$ for lags $\tau \in \{T, 2T, \ldots, KT\}$. Because the channel is time-varying, the taps of this filter are changing rapidly; the approximate amount of time $t$ over which the CSI remains valid is called the channel “coherence time” and is denoted $T_c$. In this work we will assume that $T_c > MKT$, where $M$ is the batch size used during training. This means that if, during training, an entire batch of symbols were concatenated together and transmitted in one burst, the channel would remain approximately constant during this time.

In real-life scenarios, the channel is not accessible to the user, as it represents a physical environment outside of the computer, thus it is impossible to back-propagate derivatives across it. In the absence of an appropriate substitute channel model that is differentiable, one author has proposed the use of a conditional generative adversarial network (GAN) model to provide a differentiable path for derivatives to propagate [1].

The machine learning approach is proven using existing differentiable channel models and then using a generative adversarial network (GAN) to learn arbitrary unknown channel models. One appeal of this approach is that it allows for the GAN to be trained on measured data from unusual channel types that might be difficult to characterize statistically; afterwards this GAN may be then used to train the transmitter and receiver in an offline fashion. In the referenced paper, the GAN is a conditional GAN, and the conditioning information is the transmitted symbol $x$ concatenated with the pilot symbol output $y_p$, which will be described later.

3.3 Performance Metrics

Since our goal is reliable communication, the most important metric is the bit error rate (BER) which is the probability $P_b$ that a single bit will be in error, averaged over a large number of transmitted bits. In characterizations of communication systems it is ubiquitous to plot the system BER as a function of the received signal to noise ratio (SNR). Specifically, the x-axis quantity most commonly used is the “SNR per bit”, denoted $E_b/N_0$, defined by:

$$ \frac{E_b}{N_0} = \text{SNR} \cdot \frac{K}{N} $$

where $E_b$ is the energy per bit, $N_0$ is the one-sided noise power spectral density (PSD). The quantity $\eta = N/K$ gives the number of bits per second per Hertz of bandwidth, and is a measure of the spectral
efficiency of the modulation and coding scheme (higher is better). Using the SNR per bit as the x-axis value for the BER plots gives a better apples-to-apples comparison of the actual error resilience of modulations with different spectral efficiencies.

We used the quadrature phase shift keying (QPSK) modulation as our reference point to see how close to optimum our modulation scheme was. We will describe in a later section why this modulation was chosen.

4 DATASET AND IMPLEMENTATION

4.1 Dataset

The recently released Sionna library provides communication channel models that can be directly integrated into neural network dataflows to enable back-propagation of derivatives over the channel [8]. The additive white Gaussian noise (AWGN) and 3GPP Jakes channel models will be used for testing. The Sionna models allowed us to generate somewhat realistic channel outputs ad infinitum for training and testing.

The 3GPP standards organization publishes standarized channel models covering a variety of frequency ranges and propagation scenarios [9]. In our simulations, we study a simple but important case—narrowband Rayleigh fading. This was implemented by setting the delay spread and max speed to zero. This is a reasonable model in a non-line of sight flat fading channel. Each operation used to compute the channel output from the channel input supports automatic differentiation (AD), so it can be used on a
TensorFlow GradientTape to backpropagate derivatives across.

The AWGN model is a channel with complex white Gaussian noise with variance $N_0$ added between the input and output. The signal to noise ratio is specified when the channel model is initialized.

The data communicated between transmitter and receiver consists of arbitrary ones and zeros. The goal of the network is to perfectly reconstruct arbitrary binary messages. Separate random binary codewords were generated to validate the models.

### 4.2 Architectures

The transmitter network and receiver network architectures were implemented as shown in Tables 1 and 2. $N = 8$ per codeword were transmitted, which results in $K = 4$ complex samples per codeword. Each layer used a sigmoid activation except for the second dense layer of the transmitter, which used a linear activation. This spectral efficiency $\eta = N/K = 2$ bps/Hz is the same as quadrature phase shift keying (QPSK) with sinc pulse shaping, so we used this as the comparison for assessing how close to optimal our modulation and coding scheme was. It can be shown mathematically that given the constraints of a fixed energy per bit $E_b$ and spectral efficiency $\eta = 2$, that the QPSK modulation maximizes the mutual information $I(x; y)$ across all modulation and coding schemes in an AWGN-only channel. This is because under AWGN, the problem is reduced to choosing the signal constellation in the I-Q plane that maximizes the distance between symbol points for a fixed energy per bit $E_b$; this corresponds to four equispaced points on a circle of radius of $\sqrt{2E_b}$, which is exactly the QPSK constellation shape.

The main modifications made from the networks used in [1] is that we increased the number of neurons per layer in the TX and RX networks to improve performance and added BatchNormalization layers after each hidden layer, which was found to dramatically improve training speed.

**Table 1. Transmitter Network Architecture**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Shape</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>(None, 8)</td>
<td>0</td>
</tr>
<tr>
<td>Reshape</td>
<td>(None, 8)</td>
<td>0</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 256)</td>
<td>2,304</td>
</tr>
<tr>
<td>BatchNormalization</td>
<td>(None, 256)</td>
<td>1,024</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 256)</td>
<td>65,792</td>
</tr>
<tr>
<td>BatchNormalization</td>
<td>(None, 256)</td>
<td>1,024</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 8)</td>
<td>2,056</td>
</tr>
<tr>
<td>Reshape</td>
<td>(None, 4, 2)</td>
<td>0</td>
</tr>
<tr>
<td>TxNormalizer</td>
<td>(None, 4, 2)</td>
<td>0</td>
</tr>
</tbody>
</table>

The generator network and discriminator network architectures for the GAN were implemented as shown in Tables 3 and 4. The generator output represents the output of the propagation portion of the channel, and the discriminator categorizes if the input is received data generated from the real channel (true) or the generator (false). Once the generator is fully trained, it replaces the channel models from [8] in the neural network implementation. To test the ability of the GAN to learn the channel, the GAN is trained on the 3GPP channel from [9] so that the theoretical best performance of the communications neural network with the GAN channel should be identical to the performance with the 3GPP Sionna channel.
Table 2. Receiver Network Architecture

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Shape</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>(None, 8, 2)</td>
<td>0</td>
</tr>
<tr>
<td>Reshape</td>
<td>(None, 16)</td>
<td>0</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 256)</td>
<td>4,352</td>
</tr>
<tr>
<td>BatchNormalization</td>
<td>(None, 256)</td>
<td>1,024</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 256)</td>
<td>65,792</td>
</tr>
<tr>
<td>BatchNormalization</td>
<td>(None, 256)</td>
<td>1,024</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 8)</td>
<td>2,056</td>
</tr>
</tbody>
</table>

Table 3. Generator Network Architecture

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Shape</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input + Noise</td>
<td>(None, 16 + 16)</td>
<td>0</td>
</tr>
<tr>
<td>Reshape</td>
<td>(None, 32)</td>
<td>0</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 128)</td>
<td>4,224</td>
</tr>
<tr>
<td>BatchNormalization</td>
<td>(None, 128)</td>
<td>512</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 128)</td>
<td>16,512</td>
</tr>
<tr>
<td>BatchNormalization</td>
<td>(None, 128)</td>
<td>512</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 128)</td>
<td>16,512</td>
</tr>
<tr>
<td>BatchNormalization</td>
<td>(None, 128)</td>
<td>512</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 8)</td>
<td>1,032</td>
</tr>
<tr>
<td>Reshape</td>
<td>(None, 4, 2)</td>
<td>0</td>
</tr>
<tr>
<td>TxNormalizer</td>
<td>(None, 4, 2)</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4. Discriminator Network Architecture

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Shape</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>(None, 8, 2)</td>
<td>0</td>
</tr>
<tr>
<td>Reshape</td>
<td>(None, 16)</td>
<td>0</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 32)</td>
<td>544</td>
</tr>
<tr>
<td>BatchNormalization</td>
<td>(None, 32)</td>
<td>128</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 32)</td>
<td>1,056</td>
</tr>
<tr>
<td>BatchNormalization</td>
<td>(None, 32)</td>
<td>128</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 32)</td>
<td>1,056</td>
</tr>
<tr>
<td>BatchNormalization</td>
<td>(None, 32)</td>
<td>128</td>
</tr>
<tr>
<td>Dense</td>
<td>(None, 1)</td>
<td>33</td>
</tr>
</tbody>
</table>
4.3 Training

Mirroring the original work [1, 7], the system was trained using the Adam optimizer. Each of the four neural networks was given its own separate optimizer. For the transmitter and receiver, a learning rate of 0.001 was used and for the generator and discriminator a learning rate of 0.0001 was used. Loss functions for each network are described below. It was found that the test performance seemed the best when the networks were trained at a constant and low SNR value as opposed to training at a higher SNR or a range of SNR values. Subsequently, all results that are communicated in this work used networks trained at an $E_b/N_0 = 3$ dB.

For input codewords of $N = 8$ bits, there are $2^8 = 256$ possible combinations. Subsequently we chose a training batch size larger than this so each codeword would get represented on average a few times in each batch. We chose a batch size of $M = 2^{11}$ to satisfy this.

We wrote our own custom training loop using the TensorFlow GradientTape feature. Each epoch of training, entirely new binary data $s$ and pilot symbols $y_p$ were generated using Sionna as the source of the channel outputs (but not for back-propagating because that would violate the spirit of this method).

Because our channel is a flat fading channel, the main perturbation that it applies is a phase shift and amplitude scaling. Empirically it was found that the loss wouldn’t start to decrease until about 1000 epochs, after which it dropped rapidly. Given that estimation of the phase shifts to a high accuracy is required for coherent signal demodulation, the receiver would need to learn how to “unwarp” the phase shift each possible phase shift. If a 1 degree accuracy were needed (hypothetically), that would mean it would need to learn how to undo 360 different phase shifts. Under this view, 1000 epochs would correspond to seeing each phase shift about three times, which intuitively makes sense, as it would probably take a few observations of each example phase shift before it “caught on”. It was found that $2^{12}$ training epochs were sufficient for all of the networks to train sufficiently well before the error “bottomed out” near what appeared to be a global minimum.

The training process is visualized in Figure 1. Details for how each of the networks was trained is given below.

4.3.1 Receiver

In this step, the instantiation of the channel can be thought of as a random draw of $h$ from the set of all possible channels in the sample space $\mathcal{H}$. It is assumed that this random draw is representative of the channels state during a training batch. A batch of binary data $s$ is encoded into a batch of codewords $x$. A deterministic pilot signal $x_p$ is also generated which is used to probe the state of the channel to assist the receiver in decoding. In continuous-time notation, the pilot signal is assumed to be a very short pulse approximated by the Dirac delta function:

$$x_p(t) = \delta(t)$$  \hspace{1cm} (6)

The received pilot signal $y_p$ is therefore (assuming the channel is locally-LTI):

$$y_p(t) \approx \int_{-\infty}^{\infty} h(\tau) \delta(t - \tau) \, d\tau = h(t)$$  \hspace{1cm} (7)

Thus the pilot allows for estimation of the CSI at the receiver. This is necessary for performing coherent detection. Similarly, channel output $y$ represents the response of the channel to input $x$. During training, it
is assumed that the receiver knows the input sequence of bits $s$, so the receiver can be trained directly during this step using the binary cross-entropy loss between the true bits $s$ and the estimated bits $\hat{s}$. During this step the weights of the transmitter (TX) and GAN are frozen.

Because the data passes through the TX through the real channel to the RX during this step, this step is an appropriate place to compute a validation bit error rate (BER) to ensure that the trained network works in a realistic scenario.

### 4.3.2 Transmitter

The transmitter is trained using the GAN to back-propagate derivatives across a simulated channel to train the transmitter using the binary cross-entropy loss. In this step the RX and GAN weights are frozen. In our implementation we generated a new $y_p$ value and batch of bits $s$ for this training step.

### 4.3.3 Generator

The generator was trained by generating a new $y_p$ and $s$ and concatenating this with input noise $z \in \mathbb{R}^{4K \times 2}$ as inputs to the generator. Element of $z$ are drawn from $\mathcal{U}(0, 1)$, where $\mathcal{U}(0, 1)$ is the uniform distribution over $[0, 1)$. The generator output $y_{\text{gen}}$ is concatenated with $y_p$ and passed to the discriminator. The ground truth for training the discriminator output at this step is all ones (i.e. train the generator to fake out the discriminator). Binary cross-entropy loss is used at this step. The weights for the TX, RX, and discriminator are frozen in this step.

### 4.3.4 Discriminator

The discriminator is trained on two batches of data during this step, one real (drawn from the channel) and one fake (generated by the generator), both of batch size $M$. The loss function used at this step is the summation of the binary cross entropy loss from the fake batch and the loss from the real batch. The ground truth for the fake batch is all zeros, and the ground truth for the real batch is all ones. During this step, the TX, RX, and generator weights are frozen.

### 4.4 Modifications to Improve Performance

As we will see in later sections, we made a few modifications from the scheme described in [1] and [7].

The first major modification to the training process that we implemented was training the TX and RX jointly instead of just training the TX during the “Transmitter” training step described in Section 4.3.2. There is no reason why joint training of the TX/RX together can’t occur at this step, given that the GAN allows for derivatives to be back-propagated to train both networks simultaneously. This improved performance significantly.

Secondly, we added an additional loss term to during the generator training step. It was found that the generator had trouble learning a representative channel distribution $p(y|x)$; this manifested itself in the BER reaching a floor that it couldn’t seem to surpass with additional training epochs. Because the
generator is taking as inputs $x$ and $y_p$, it is essentially learning the convolution operation (for complex-valued signals), with the input noise $z$ potentially regularizing the training a little bit. Because of this, it would be very desirable for the generator output to approximately match $y_p$, which is what the “true” channel output should be given $x$ and $h = y_p$. This was incentivized by adding an extra term to the generator loss, namely the mean squared error (MSE) between $y_p$ and $y_{\text{gen}}$. Both terms were given equal weight (i.e. just added to form the total loss). This was also found to significantly improve performance.

4.5 Testing

After training had completed, we performed a test of the BER as a function of $E_b/N_0$, with values ranging from 0 dB to 10 dB. During the test phase, entirely new binary data $s$, channels $h$, and noise was generated for each batch. The total number of test samples per $E_b/N_0$ value was $2^{18}$ (corresponding to $2^{21}$ bits), broken up into batches of $2^{15}$, mostly for memory reasons. This test size gave us the ability to detect errors down to the level of $2^{-21} \approx 4.8 \times 10^{-7}$. Using larger test sizes would be overkill, as our system error rate was much higher than this, and using a smaller test size would lead to higher variance and unreliable results.

5 EXPERIMENTS AND RESULTS

In this section we will describe the test performance using the aforementioned metrics described earlier, specifically the BER as a function of $E_b/N_0$, for a variety of scenarios. For comparison, we will compare each to the BER for QPSK under AWGN, which can be computed analytically as:

$$P_b = Q\left(\sqrt{\frac{2E_b}{N_0}}\right)$$

5.1 GAN Channel Model

The GAN was trained as described and used for training the transmitter as described in Section 4.3.2. The performance from the basic implementation of the GAN was lackluster, as shown in Figure 2. The GAN did not adequately learn the channel model. To improve learning in the GAN, an additional mean-squared error (MSE) loss was added between the generator output and actual channel output, as described in Section 4.4. This forces the conditional GAN to learn the convolution operation which occurs in the channel. The results marginally improved, as shown in Figure 3.
Figure 2. Train TX w/ GAN, but no MSE term, no RX training w/ GAN
Figure 3. Train TX w/ GAN, with MSE term, no RX training on GAN
The transmitter and receiver were also not adequately learning with the GAN implementation. In [1], it was proposed that the receiver weights should be frozen during training so that the learning occurs only in the transmitter. To improve training, the receiver weights were unfrozen and the GAN implementation was repeated with and without the additional MSE loss for the GAN, as described in Section 4.4.

Without the MSE loss, there was again some marginal gain but still sub-optimal performance, as shown in Figure 4. When the two modifications were combined, the performance improved drastically. The GAN was able to adequately learn the channel model, which was then successfully learned by the transmitter and receiver so the overall performance approached the optimal value, which is QPSK in AWGN.

The results for when the two modifications were applied together are shown in Figure 5. These results are very close to the theoretical max performance, which we will further see in the next section.
Figure 4. Train TX w/ GAN, no MSE term, with RX training on GAN
Figure 5. Train TX w/ GAN, with MSE term, with RX training on GAN
5.2 Training on Sionna Channel Model Directly

As a thought experiment, we also evaluated the performance of the system if we trained it all at once using the differentiable Sionna channel models and forgoing use of the GAN completely. This allows for the TX and RX to be trained on a single GradientTape. This can be reasonably interpreted as the “best” that our system could hope to perform. The test result from this experiment is shown in Figure 6. As it can be seen, the performance is even closer to ideal than the best GAN case, but not by much, implying that GAN implementation was effective.
Figure 6. Train on channel with no GAN
6 CONCLUSION

A deep learning implementation of the transmitter and receiver in a wireless communications network model was successfully able to adapt to a differentiable channel model for efficient data transmission.

In absence of a differentiable channel model, the use of a GAN to create a differentiable channel model from real channel noise was demonstrated. In theory, any channel could be approximated with a GAN so that such a channel could then be adapted in a deep learning approach.

References


[9] “Study on channel model for frequencies from 0.5 to 100 GHz, release 16.1,” Tech. Rep. 3GPP TR 38.901, ESTI, 2018.