Deep Networks for Equalization in Communications

by

Laura Brink

A thesis submitted in partial satisfaction of the
requirements for the degree of
Master's of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Anant Sahai, Chair
Professor John Wawrzynek

Fall 2018

The thesis of Laura Brink, titled Deep Networks for Equalization in Communications, is approved:

Chair _____ Date _____

_____ Date _____

_____ Date _____

University of California, Berkeley

Deep Networks for Equalization in Communications

Abstract

Deep Networks for Equalization in Communications

by

Laura Brink

Master's of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Anant Sahai, Chair

Robustness is critical for future communication systems. We apply the techniques from meta-learning and machine learning to the communications domain. Specifically, we explore how equalization techniques can learn how to handle new channel environments without training on them and how neural networks can learn to estimate and correct carrier frequency offset for new rates of rotation without training on them. We show that deep neural networks can learn to learn to estimate channel taps for two tap channels. We also explore how deep recursive neural networks learn to learn to equalize for any given channel. We demonstrate that neural networks can learn to learn to estimate and correct carrier frequency offset for new rates of rotation. We do all of this without using backpropagation to re-train the networks for each new set of environment conditions.

Contents

List of Figures

Chapter 1

Introduction

Signal processing has long relied on well-defined, structured processes and protocols to function. However, in order to move to more robust and adaptive systems, we will need to overhaul these tightly structured processes. We must design new robust and adaptive communications systems.

From an academic perspective, the rise of machine learning tools and processing has allowed us to tackle problems we have not yet been able to, like in image processing. However, we still do not fully understand the reach or the limitations of this technology. In order to study the limitations of machine learning, we must apply them to spaces that we have studied extensively, like communications, and compare them to the well-known baselines.

Most communications systems have three main processes at the receiver; equalization, demodulation, and error-correction. While we will need to design robust forms of all of these processes, we will focus on equalization for the remainder of this paper.

1.1 Motivation

Meta-learning is the idea that algorithms need to be able to 'learn to learn' in order to generalize to different applications. For example, if a robot is trained to pick up coffee mugs, then we want that robot to be able to quickly learn how to pick up water bottles without having to re-train it. Meta-learning was inspired by humans ability to generalize how to learn [11]. Additionally, researchers in neurology have studied how the brain synapses change over time, suggesting that our algorithms will have to change over time to continue learning [1]. We refer the reader to [12] for a survey of meta-learning technologies and to [7] for recent developments in meta-learning algorithms.

In this paper, we want to understand if our neural network based communication processes can learn to learn. As in, can they handle new environments with new data sequeneces without having to re-train for the new variables. The communications application is arguably an easier application of meta-learning than others have been exploring, like image classification [**khodadeh**]. In order to better understand the reach and limitations of meta-learning,

we need to apply it to something simpler, like communications. In this paper, we will explore whether or not we can learn to learn to communicate.

1.2 Background

Inter-symbol Interference and Equalization

Inter-symbol interference occurs when we are transmitting over a channel that has some echos. These echos cause the receiver to hear a garbled signal instead of the original signal from the transmitter. This is called inter-symbol interference because the receiver is hearing a combination of symbols across time.

Let $\vec{x} = [x_0, x_1, \dots x_n]$ be the set of n complex symbols that the transmitter sends over the channel that connects the transmitter to the receiver. Each channel will have different characteristics. Some channels may have echos, others may have delays, often channels will have both. When a channel has echos, this is called a multipath channel because there are multiple paths to reach the receiver. Each path is called a tap. We can characterize a channel by characterizing the taps.

Let $\vec{a} = [a_0, a_1, \dots a_\ell]$ be the set of characteristic for a multipath channel that has l taps. When a sequence of symbols like $\vec{x}$ is transmitted over this channel, the channel taps are convolved over the sequence. Additionally, there is noise in the system denoted by v_i.

$$\tilde{x}_m = \sum_{i=0}^{\ell} a_i x_{m-i} + v_m \tag{1.1}$$

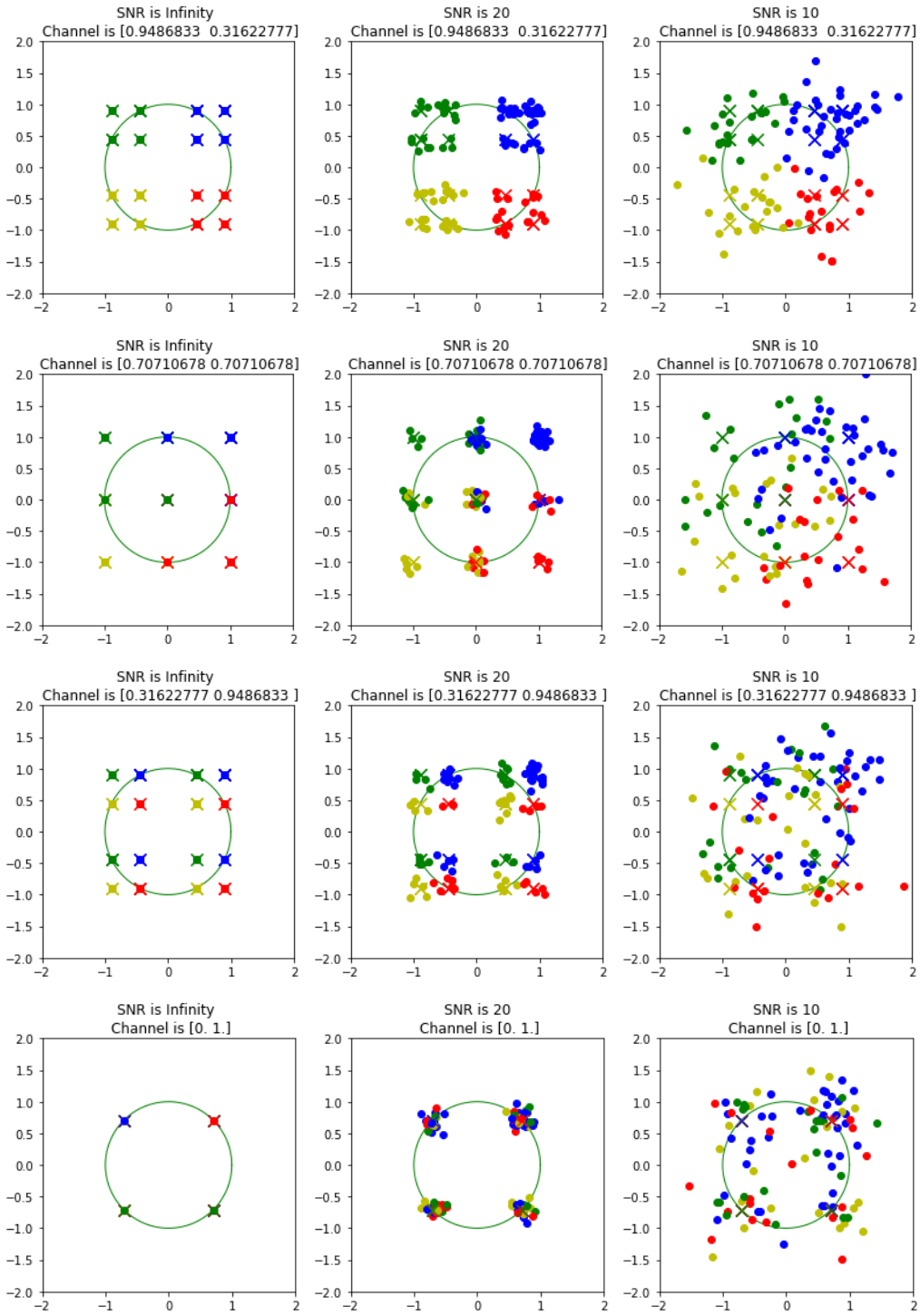
The receiver will hear a signal that is corrupted by inter-symbol interfence and noise; $\vec{\tilde{x}} = [\tilde{x}_0, \tilde{x_1}, \dots \tilde{x}_{n+\ell}]$. Receivers must be able to handle garbled signals in order to transmit data in the real world. The process of removing the inter-symbol interference is called equalization. The goal of equalization is to take in a garbled signal and output a signal with minimal inter-symbol interference.

Figure 1.1 demonstrates the effects of multi-tap channels on a QPSK modulation constellation. We show what the received signal symbols constellations are from a sequence of 100 symbols modulated in QPSK through different two tap channels. We assume that the channel taps are constant during the transmission of the signal. We see that under certain channel conditions, like when the two taps are equal, it is very difficult to distinguish between the four constellations. Engineers have built processes to remove inter-symbol interference. First, let's go into the case when the channel characteristics are known.

Equalization for a known channel

If the receiver knows the channel characteristics, $\vec{a}$, perfectly, then there are a few different methods that can be used. The zero-forcing equalizer applies the inverse of the channel

Figure 1.1: The effects of a two tap channel on the QPSK constellation.



response to the received signal. It is called zero-forcing because there will be zero inter-symbol interference if there is no noise. There are some limitations of the zero-forcing equalizer. First, the impulse response of the equalizer needs to be infinitely long. Second, if there is a weak signal at a frequency, then the inverse gain is going to be very large. This will amplify any noise in the system. Third, if there are any zeros in frequency response, these cannot be inverted.

Another equalizer, the minimum mean squared error (MMSE) equalizer, handles noise much better than the zero-forcing equalizer and is explained in the next section. While it's important to consider how well a receiver can equalize with a known channel, this is rarely the case. Usually, we do not know the channel characteristics.

Equalization for an unknown channel

When the receiver does not know the channel characteristics, the process of equalization essentially has two jobs; first, identify the channel, second, remove the inter-symbol interference. If the receiver did not identify the channel first, there would be no way to remove the affects of it on the received signal.

In order to do channel estimation, most systems require that packets begin with a known sequence called a preamble. The signal sent will be broken into two parts; $\vec{x} = [\vec{x}_{pre}, \vec{x}_{data}]$. The signal received on the transmitter is $\vec{\tilde{x}} = [\vec{\tilde{x}}_{pre}, \vec{\tilde{x}}_{data}]$. The receiver knows what the orginal preamble sequence was, $\vec{x}_{pre}$, and can use the received preamble sequence, $\vec{\tilde{x}}_{pre}$, to estimate the behavior of the channel. Once the channel is estimated, the receiver then equalizes the data, $\vec{\tilde{x}}_{data}$.

One common method to estimate the channel is to use the least squares optimization framework. Let H be the estimate of the channel. The least squares channel estimator wants to find H that minimizes the squared error between the received signal, $\vec{\tilde{x}}_{pre}$, and what the predicted received signal would be if the the channel H was applied to the original preamble, $\vec{x}_{pre}$.

$$\min_{H} ||\vec{\tilde{x}}_{pre} - H\vec{x}_{pre}||^2 \tag{1.2}$$

The receiver needs to choose the length of H, representing how many taps (or echos) there might be in the channel. The length of H will be set based on the environment that the transmitter and receiver are communicating in.

Once the channel response has been estimated, an equalizer has to use that information to remove the inter-symbol interference from the received signal. The MMSE equalizer minimizes the error between the equalized preamble and the original known preamble by choosing the optimal inverse of the channel response, W.

$$\min_{W} ||\vec{x}_{pre} - \vec{\hat{x}}_{pre}||^2 \tag{1.3}$$

$$\vec{\hat{x}}_{pre} = W(H\vec{x}_{pre} + \vec{v}) \tag{1.4}$$

$$W^* = \vec{x}_{pre}\vec{x}_{pre}^T H^T (H\vec{x}_{pre}\vec{x}_{pre}^T H^T + \sigma_v^2 I)^{-1} \tag{1.5}$$

The MMSE equalizer works well for known and unknown channels and does not amplify noise like the zero-forcing equalizer. This is why the MMSE equalizer paired with a least squares channel estimator are widely used.

Carrier Frequency Offset and Correction

Now, if we were to implement our minimum mean squared error equalizer on a physical receiver, we would find some problems with our equalization process. Our equalizer will equalize the first symbols very well. However, as we equalize end parts of our sequence, we will encounter a physical phenomonen called carrier frequency offset, CFO. Carrier frequency offset occurs when the transmitter and receiver are at slightly different frequencies. It also occurs when the transmitter and receiver are moving, causing a sort of Doppler effect.

When there is a significant CFO present, the symbols will gradually rotate. The received signals will be the original signals rotated at a rate ω.

$$\tilde{x}_i = x_i e^{ij\omega} + v_i \tag{1.6}$$

Figure 1.2 demonstrates the effects of CFO on a QPSK modulation constellation (no multipath channels). We show what the received signal symbols constellations are from a sequence of 100 symbols modulated in QPSK with different CFO rates. We assume that the CFO rate, ω, is constant during the transmission of the signal.

There are a few ways to handle CFO, some are more elegant than others. The first solution is to try to side-step the problem entirely. Since the effects of CFO depend on the length of a packet, one solution is to make packets so short that the symbols only move a little bit. In this case, the effects of CFO can be ignored.

Another solution is using a phase-locked loop, which is a control system that outputs a signal with a phase related to the input signal. A Costas loop is a circuit that implements a phase-locked loop for CFO correction for continuous time signals by adapting the sampling rate [3].

What happens when there is both CFO and inter-symbol interference? The received signal will have the effects of the channel and CFO as well as the noise.

$$\tilde{x}_m = (\sum_{i=0}^{l} a_i x_{m-i}) e^{mj\omega} + v_m \tag{1.7}$$

Figure 1.3 demonstrates the effects of CFO and multi-tap channels on a QPSK modulation constellation. We show what the received signal symbols constellations are from a sequence of 100 symbols modulated in QPSK with different CFO rates and two tap channels. We assume that the CFO rate, ω, and the channel taps are constant during the transmission of the signal. Modern day receivers have to combine CFO correction, channel estimation, and equalization processes in order to handle these received signals.

Figure 1.2: The effects of a carrier frequency offset on the QPSK constellation.

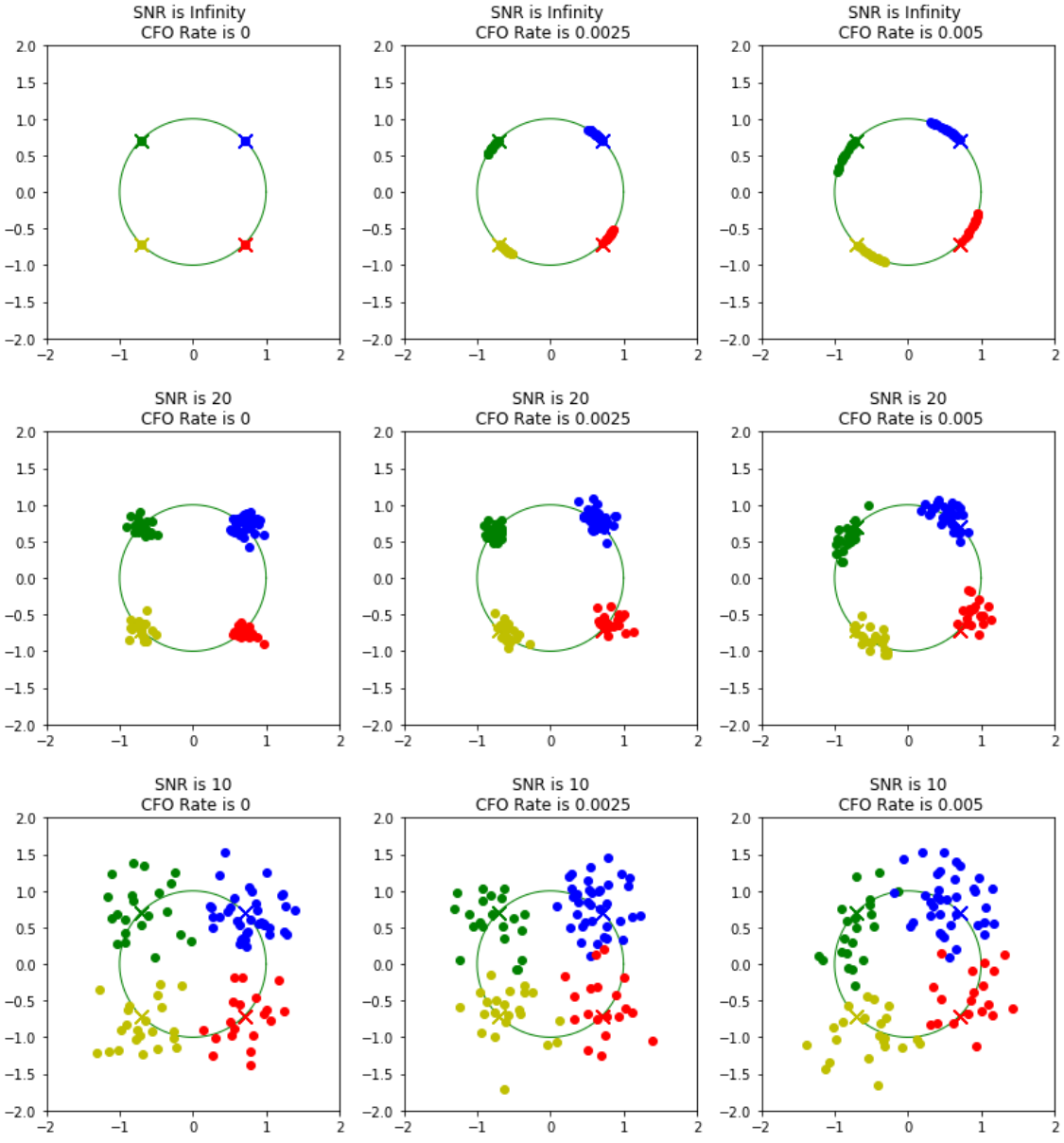
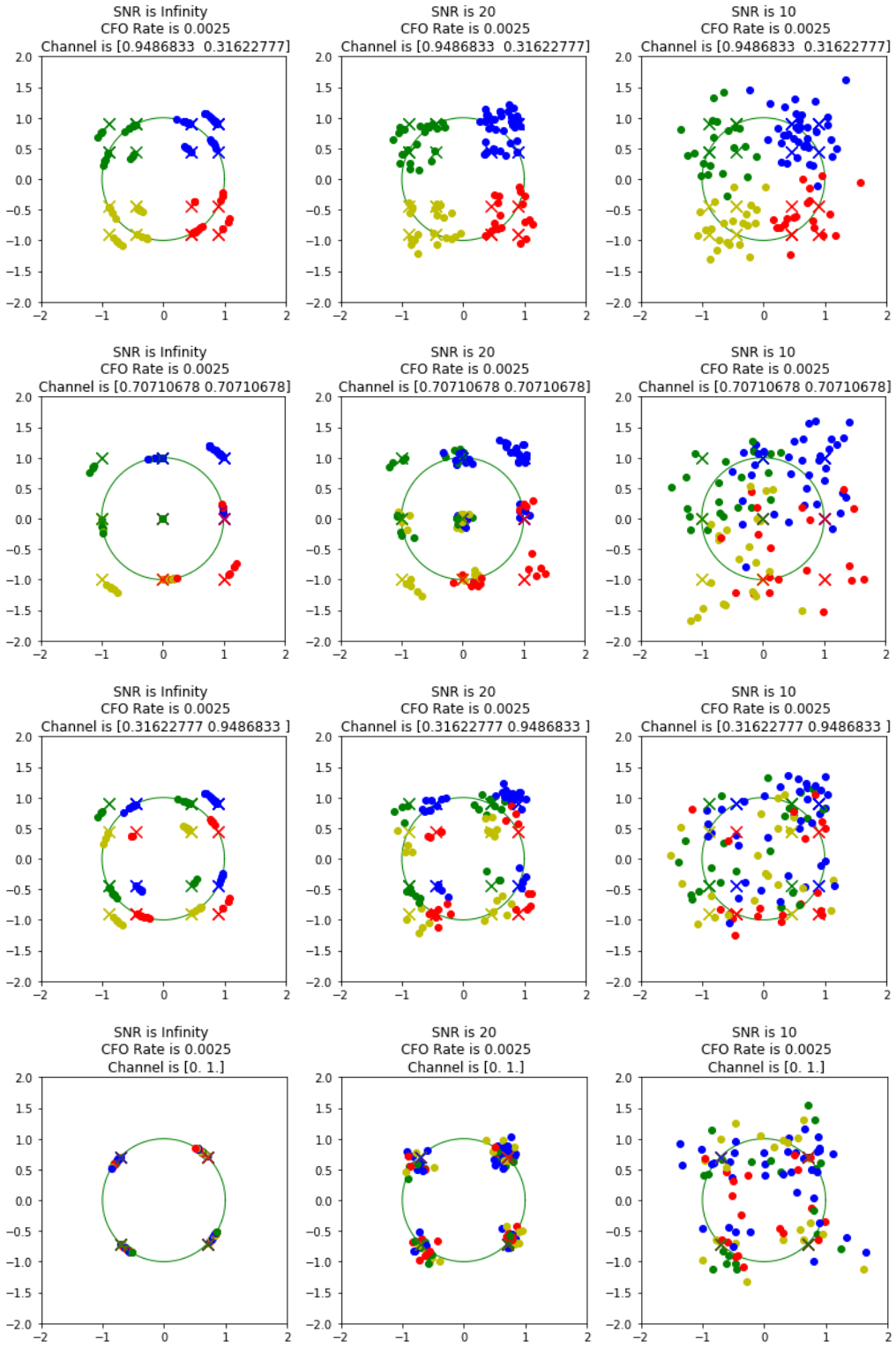


Figure 1.3: The effects of a two tap channel and a carrier frequency offset on the QPSK constellation.



1.3 Related Works

Recently, many researchers have become more interested in applying deep learning techniques to communications systems. We refer the reader to [2][4][20][8][17] for surveys and motivations of these kind of works. Most of the works that we have found typically only deal with additive white gaussian noise (AWGN) channels or Rayleigh fading channels.

In [5], Dörner et. al. implemented an end to end transmitter and reciever with neural networks that allowed gradients to flow all the way back from the receiver during training. However, they restricted themselves to only sending a certain set of messages, only seeing AWGN channels, and they did not address CFO.

Other groups have been focusing on decoding with recurrent neural networks (RNNs) but also only deal with AWGN channels [9][10]. Others have been working with generative adversarial networks (GANs) to train an end-to-end communication system [21]. However, they also only consider AWGN channels or Rayleigh fading channels.

For those that do consider more complex channels, most re-train their models for each new channel seen. Ye et. al. consider OFDM systems where their feedforward neural networks estimate the channel state information then train offline for that specific channel [22]. [19] considers nonlinear channels but re-trains their network for each new channel. Note, this work does not go into detail about the architecture used or how the networks are trained.

Goldsmith and Farsad train a detector for optical and molecululular channels [6]. They assume a Poisson model for the channel and attempt to predict the probability mass function. However, they assume that they retrain for each Poisson parameter and they do not address CFO.

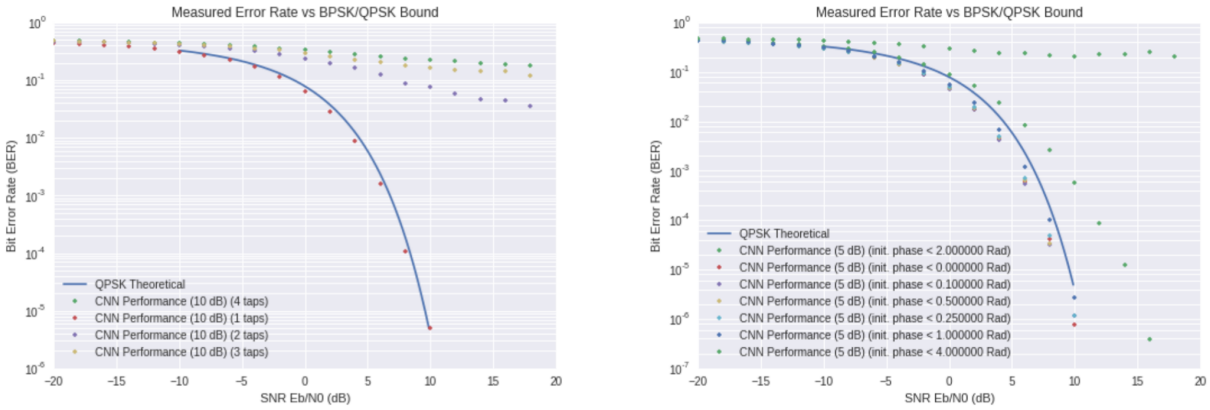
Timothy O'Shea's group has been doing some excellent work in this area. O'Shea's first work in this area jointly optimized a transmitter and receiver for a given channel model (AWGN and Rayleigh fading) but the work does not go into detail about the perforamnce for various channels [17].

In a subsequent paper, O'Shea et. al. explore how multipath channels and random inital phase affects the system performance [13]. Figure 1.4 shows how their performance drastically decreased with multipath channels and for non-zero phase offset.

In [15], they added an attention model to perform synchronization for time, frequnecy, phase and sample timing offset. However, their work showed that the synchronization was still quite noisy and did not perform much better than without the attention model [13]. Additionally, this work did not consider channel fading.

O'Shea et. al. has also attempted to use GANs to approxiate the channel response model for AWGN channels with and without phase noise[14]. The GANs were unable to find accurate probability density functions to represent the channel response.

In [18], they use convolutional neural networks (CNNs) to estimate, but not correct, carrier frequency offset and timing offset. However, they only consider AWGN channels and Rayleigh fading channels. In [16], they explore how unsupervised learning can train autoencoders for multiple antenna communications. They do re-train for each new channel and use a Rayleigh fading channel model.



(a) Impact of Delay Spread

(b) Impact of Random Initial Phase

Figure 1.4: The impacts of multipath channels and initial phase offsets on bit error performance [13].

Chapter 2

Deep Networks for Equalization[1]

The equalization process removes inter-symbol interference caused by the channel from a sequence of symbols. We will explore how neural networks can both estimate the channel characteristics and remove the inter-symbol interference. For the remainder of this paper, we will assume that all channels only have real parts.

2.1 Channel Estimation

The process of channel estimation tries to identify the coefficients of the channel taps as best as possible. We want to estimate $\vec{a}$ that minimizes the difference between the estimated channel taps, $\hat{\vec{a}}$, and the true channel taps, $\vec{a}$. We can choose $\hat{\vec{a}}$ to minimize this difference.

In Figure 2.1, we compare the performance of neural network channel estimator, a K-Nearest-Neighbors (KNN) channel estimator (where $K = 15$), and the least squares channel estimator for two tap real channels. We train the neural network and KNN on $40k$ data sequences with random channels. Each data sequence has a preamble length of 100 bits modulated to 50 QPSK symbols. Each channel tap is a uniform random variable, $[-1, 1]$. We normalize the channel power such that $||\vec{a}||^2 = 1$.

The neural network channel estimator takes the received preamble, $\vec{\tilde{x}}_{pre}$, and the known preamble, $\vec{x}_{pre}$, as inputs. The output of the neural network is the estimate of the channel taps, $\hat{\vec{a}}$. We assume that the neural network knows there are only two channel taps and thus limit the size of the output to two. The neural network has an architecture that consists of the three dense layers. The first and second layers has 300 nodes with sigmoid activation functions. The final layer has two nodes with the tanh activation function. The network has decaying learning rate that starts at 0.01.

All three channel estimators are tested on $10k$ data sequences with random channels. The neural network and the KNN channel estimators are not re-trained for each new channel. The least squares estimator consistently outperforms the neural network and KNN channel

[1]The contents of this chapter were produced in collaboration with Nipun Ramakrishnan.

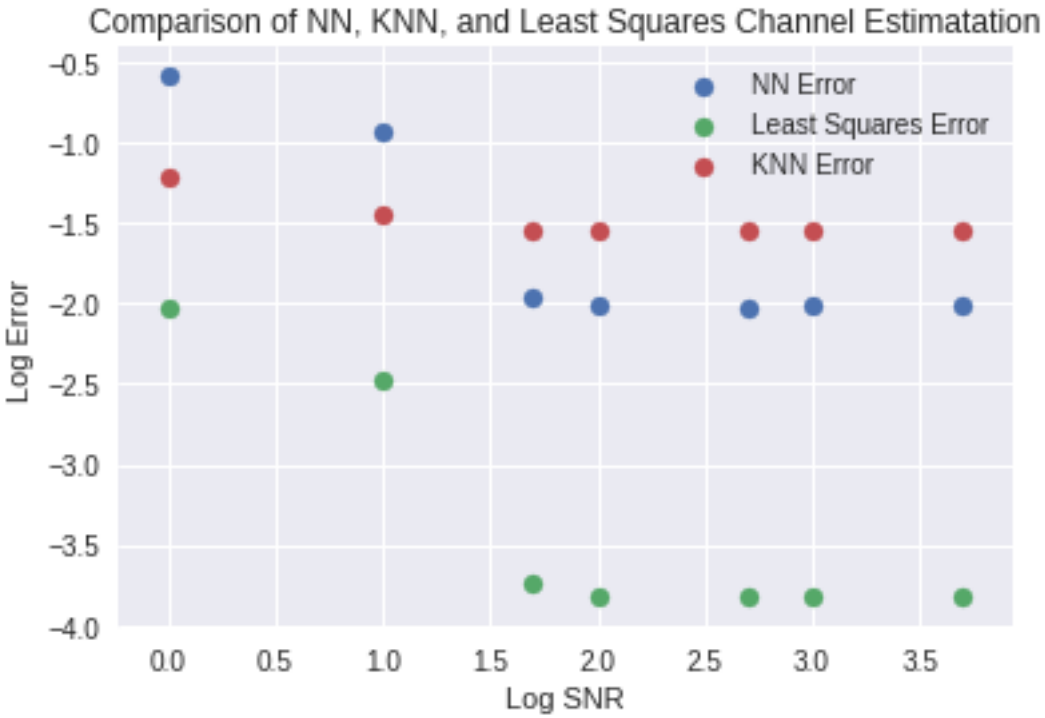


Figure 2.1: Comparison of a neural network, a K-Nearest-Neighbors, and a least squares channel estimator for two tap real channels.

estimators, especially for high SNRs. The neural network channel estimator performs better than the KNN channel estimator for high SNRs but worse for low SNRs.

2.2 Channel Equalization

The process of equalization tries to remove inter-symbol interference from a sequence of symbols. In a two tap channel, the equalization process is trying to remove the effect from the second channel tap. Given the received symbol, $\tilde{x}_m$, the channel taps, a_0, a_1, and the previous symbol, x_{m-1}, we can solve for the best estimate of the orignal symbol.

$$\tilde{x}_m = a_0 x_m + a_1 x_{m-1} + v_m \tag{2.1}$$

$$\hat{x}_m = \frac{\tilde{x}_m - a_1 x_{m-1}}{a_0} \tag{2.2}$$

If we want neural networks to perform equalization, they might try to solve for $\hat{x}_m$ in this way. If that is the case, then the neural networks will need to know how to take and input and divide by it and how to multiply two inputs.

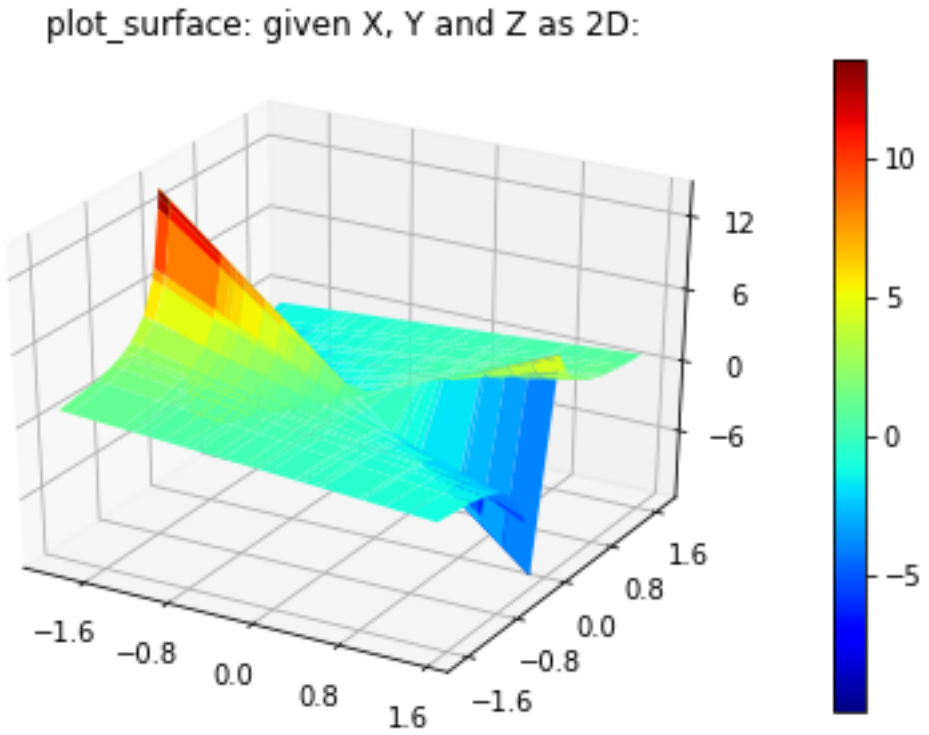


Figure 2.2: Topographical surface representation of the division function; $z = \frac{x}{y}$.

Learning an inverse

We explore whether a neural network can learn to do division. We did not do a thorough architecture search or train for very long as we wanted an idea of the behavior of the neural network and will do this in future work. Figure 2.2 shows the topological representation of the division function, $z = \frac{x}{y}$. A neural network that is trained to do division will need to approximate the surface of this function. When y is close to zero, z becomes very large and goes to infinity. These large peaks will make it difficult for a neural network to do division.

Figure 2.3 shows the performance of a neural network learning division. The neural network's input is x and y and the output is $\hat{z}$. Both inputs are uniform random variables; x is drawn uniformly $[-1, 1]$ and y is drawn uniformly $[\beta, 1]$. The loss function is the mean squared error between the estimated division and the true division, $\hat{z} - \frac{x}{y}$. The architecture consists of two dense layers with 50 nodes each and sigmoid activiation function. This feeds into a linear layer that outputs the scalar estimate of $\hat{z}$. The network is re-trained with $10k$ data points for different values of β. As β gets close to zero, the error increases dramatically. This is expected because the closer β is to zero, the stronger the effects are of the infinite peaks of the division function.

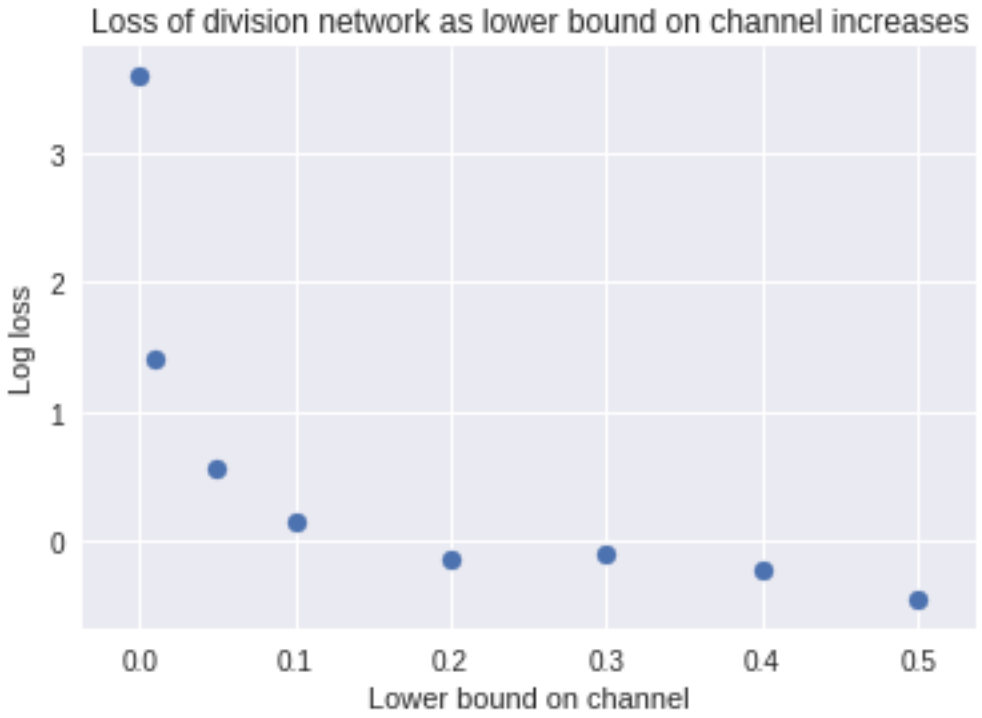


Figure 2.3: Neural network divsion with respect to the lower bound of y.

Learning to multiply two inputs

We explore whether a neural network can learn to do multiplication. We did not do a thorough architecture search or train for very long as we wanted an idea of the behavior of the neural network and will do this in future work. Figure 2.4 shows the topological representation of the multiplication function, $z = xy$. A neural network that is trained to do multiplication will need to approximate the surface of this function. When x and y are close to zero, the function is stable. However, as x and y grow, the function becomes unstable and grows to infinity.

Figure 2.5 shows the performance of a neural network learning multiplication. The neural network's input is x and y and the output is $\hat{z}$. Both inputs are uniform random variables; x is drawn uniformly $[-1, 1]$ and y is drawn uniformly $[0, \gamma]$. The loss function is the mean squared error between the estimated multiplication and the true multiplication, $\hat{z} - xy$. The architecture consists of two dense layers with 50 nodes each and sigmoid activiation function. This feeds into a linear layer that outputs the scalar estimate of $\hat{z}$. The network is re-trained with $50k$ data points for different values of γ. As γ increases, the error increases dramatically. This is expected because the larger γ is, the stronger the effects are of the infinite limits of the multiplication function.

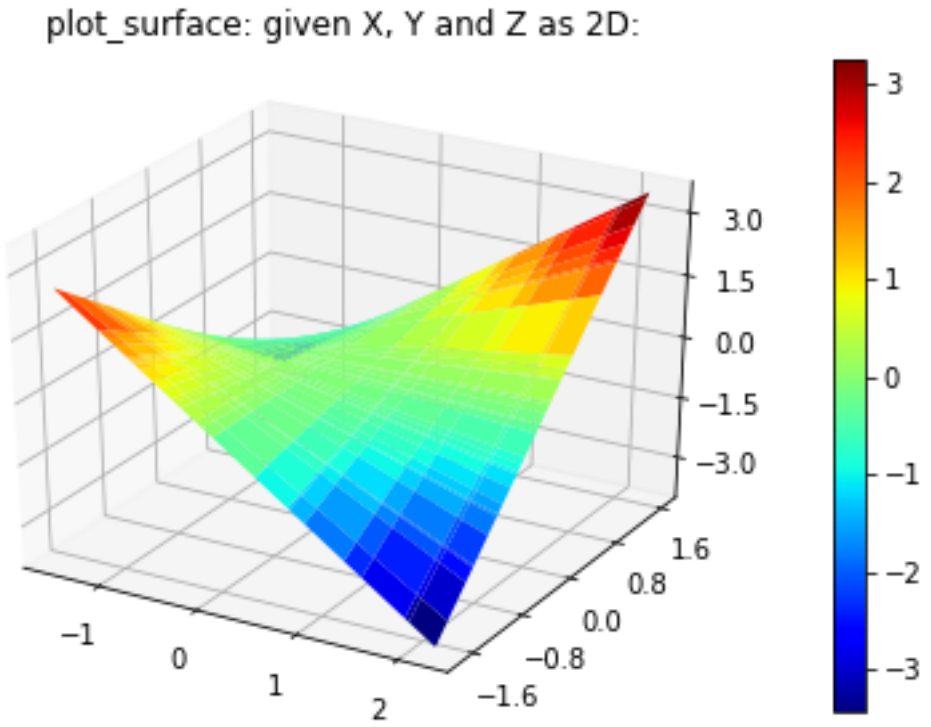


Figure 2.4: Topographical surface representation of the multiplication function; $z = xy$.

RNN for Channel Equalization

We designed an RNN to perform channel equalization. The inputs of the RNN are the true channel taps. We assume a two tap channel so a_0, a_1 are input to the RNN. Another input is the received data sequence, $\vec{\tilde{x}}_{data}$. The output of the RNN is an estimate of the original data sequence, $\vec{\hat{x}}_{data}$. The loss function that the RNN is trained on is the mean squared error between the original and estimated data sequence, $||\vec{x}_{data} - \vec{\hat{x}}_{data}||^2$.

The neural network architecture for our equalizer uses a special type of RNN called a bidriectional long-short term memory (LSTM) network. A bidirectional RNN connects the forward and backwards nodes of the RNN, allowing the output to depend on both future and past states. An RNN that has LSTM units allows the nodes to store memory for a short period of time. Our network also has time-distributed dense layers which are used after LSTMs to flatten the output by applying a fully connected dense layer at each time step.

The input to our RNN is the channel taps concatenated with a sequence of 10 of the symbols. The inputs are fed into four layers of bidirectional LSTM units. Each layer has a state size of 90 units for each forward and backward state. The output of the four bidirectional LSTM layers is 180 units for each of the 10 sequences. This is then fed into two time-distributed dense layers with 180 nodes and 100 nodes respectively and they each have Relu activation functions. These time-distributed dense layers bring the output size down

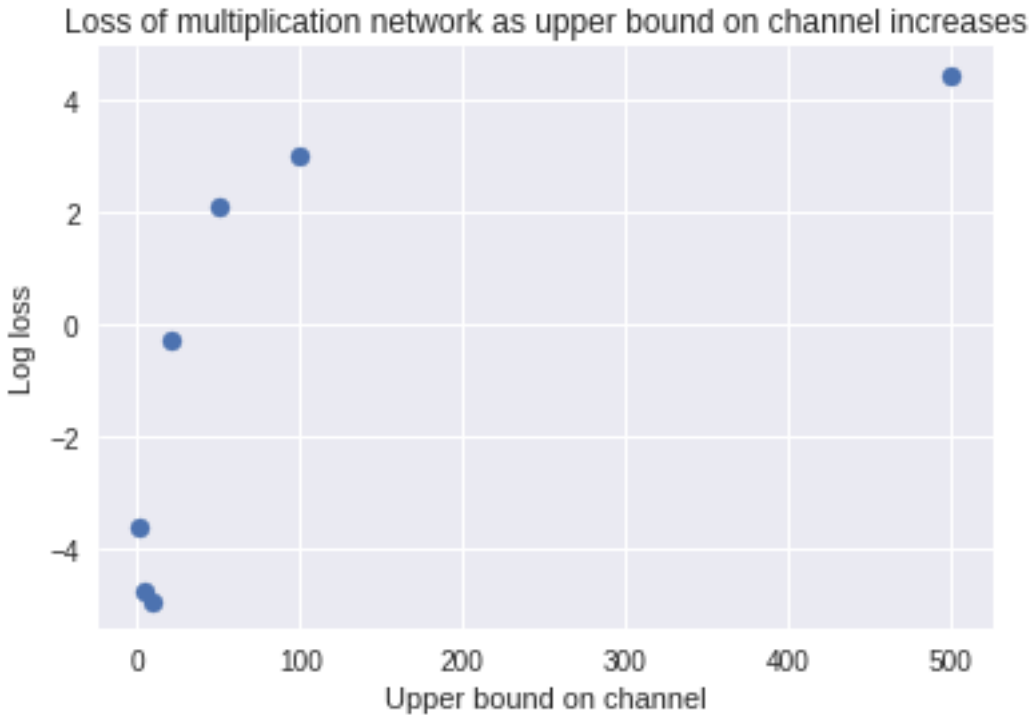


Figure 2.5: Neural network divsion with respect to the upper bound of y.

from 180 to 100 to 100. The output is then fed into one final time-distributed dense layer with 100 nodes and linear activation. The outputs of the RNN are 10 equalized symbols.

Figure 2.6 compares the mean squared error loss on test data for the RNN architecture defined above and the MMSE equalizer defined in the introduction. The RNN is trained on $40k$ data sequences, with a decaying learning rate starting at 0.01. The RNN and MMSE are tested on the same $10k$ data sequences. New data is generated for each SNR and the RNN is re-trained for each SNR. Each data sequence consits of 60 bits modulated to 30 complex symbols. We assume that the RNN and MMSE both have access to the true channel chracteristics.

The RNN equalizer performs comparably to the MMSE equalizer for low SNRs and performs better than MMSE equalizer for medium SNRs. However, the RNN equalizer performs worse than the MMSE equalizer for high SNRs. We take a closer look at what kind of channels the RNN equalizer is getting wrong.

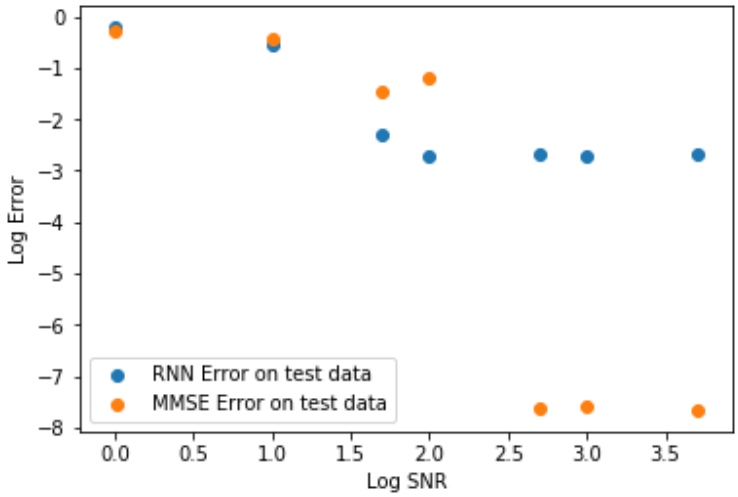


Figure 2.6: Comparison of RNN and MMSE mean squared error after equalization.

Tap 1	Tap 2	Counts of bad estimates	Mean Squared Error
0.707	0.707	14	0.7162
−0.707	0.707	1	0.5236
0.707	−0.707	16	0.8172
−0.707	−0.707	12	0.6492

Figure 2.7 shows which channels the RNN equalizer and classic demodulator got any bits wrong. There were a total of $50k$ random channels in the test set with SNR= 100. Of that set, only 43 channels resulted in incorrect bits after the RNN equalizer and classic demodulator. From the figure, it is clear that these difficult channels are clustered into four regions. All of the four regions are when the first and second tap of the channel are equal in magnitude. The mean squared error between the equalized data and the true data among just the bad channels was 0.7306. The mean squared error among the good channels was 0.000264.

We expect these channels to be difficult. Refer to Figure 1.1 to see how the constellations fall right on top of each other. Additionally, if we think about these channels in terms of the infinite impulse response, these channels could result in highly variable inverses. Because the two channels are equal, then the infinite impulse response is constantly cancelling out and everything will have to balance out exactly. This representation means there will be a lot of variation of the inverse of those channels.

In this chapter, we designed a deep neural network to estimate two tap channels. We explored the difficulties in learning to divide and learning to multiply with neural networks. We showed that a deep recursive neural network, with bidirectional LSTM layers, can learn to learn to equalize for random channels. We also examined when these equalizers fail.

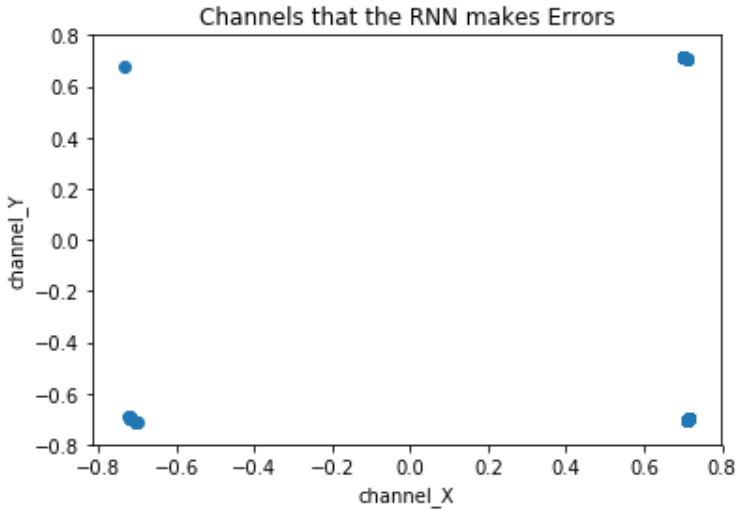


Figure 2.7: What two tap channels does the equalizer get wrong?

Chapter 3

Deep Networks for Carrier Frequency Offset[1]

The carrier frequency offset (CFO) correction process removes the symbol rotation caused by the non-ideal conditions of the transmitter and receiver environment. We will explore how neural networks can both estimate the CFO and correct for rate of rotation without using backpropagation for each new rate.

3.1 Recurrent Neural Network Follows a Circle

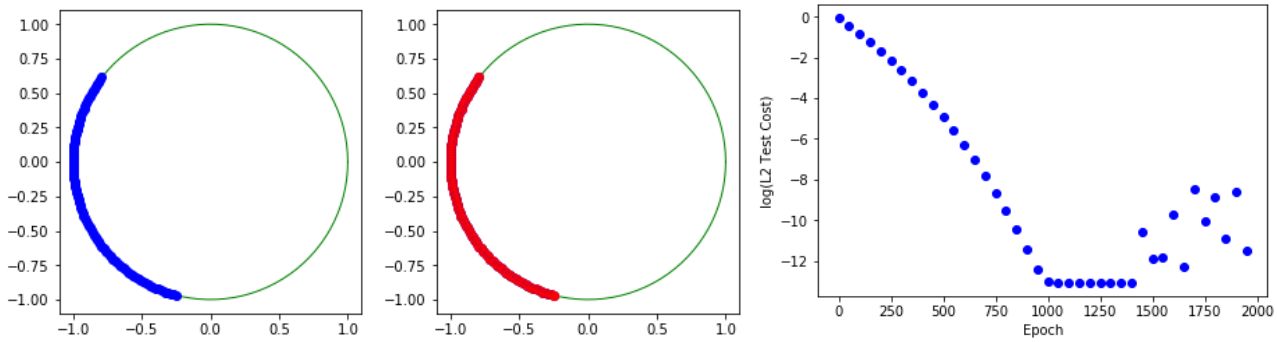
A carrier frequency offset affects the received symbols by slowly rotating them around the origin. If we want a neural network to be able to correct this, we essentially need it to act as a clock, following around a circle at a certain rate. We first explore what kind of architectures are necessary for a recurrent neural network (RNN) to learn how to follow a circle. We show that a simple, small linear RNN can learn to act as the rotation matrix for a single rate of rotation around a circle. We also show that in order to have an RNN follow a circle for different rates of rotation, we need a large non-linear layer.

Single Rate

Figure 3.1 shows a recurrent neural network (RNN) that follows a circle for a single rate of rotation. The network's input is the starting point, $x[0]$, and it must predict the next 100 points, $x[1] \dots x[99]$. Point $x[i]$ is rotated by $e^{ij\omega}$ where ω is the rate of rotation. The RNN architecture is one linear layer that takes the current point as input, $x[i]$, and outputs the next point, $x[i+1]$. We are forcing the state of the RNN to be the next point; state $= x[i+1]$. We use a linear layer here because the network essentially has to learn how to become the rotation matrix which is linear with respect to the input.

[1]The contents of this chapter were produced in collaboration with Nikhil Shinde.

Figure 3.1: Linear neural network: follow a circle for a constant CFO rate.



$$R = \begin{bmatrix} \cos(\omega) & -\sin(\omega) \\ \sin(\omega) & \cos(\omega) \end{bmatrix} \tag{3.1}$$

The RNN is trained for a constant rate of rotation, ω, applied to sequences of 100 points. The network trained for $2k$ epochs, each with a batch size of $1k$ data sequences and with a learning rate of 0.006. The initial starting point, $x[0]$ is a uniform random variable drawn from on the unit circle. The network is tested on $1k$ new data sequences but with the same ω. Figure 3.1 shows the results of an RNN trained and tested for $\omega = 0.02$. The network achieves a loss on the order of 10^{-13}. However, the minimum loss is not very sticky. The network ends up jumping away from this minimum while training. A more thorough search over learning rate decay is needed to prevent this jumping from happening.

Different Rates

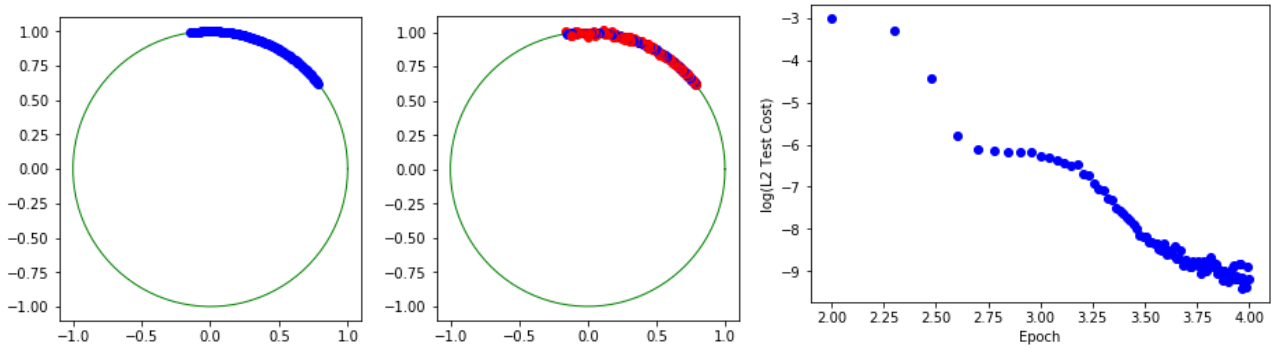
Figure 3.2 shows a recurrent neural network (RNN) that follows a circle for a given rate of rotation. The network's input is the starting point, x_0 and the rate of rotation, ω. The RNN must predict the next 100 points, $x_1 \dots x_{99}$. Point x_i is rotated by $e^{ij\omega}$.

We cannot use just linear layers in this case because the RNN must learn to find $\cos(\omega)$ and $\sin(\omega)$ for different ω. Essentially, this RNN needs to approximate the $\sin, \cos$ functions an then apply them to the data points.

The RNN architecture consists of one non-linear layer, with 100 nodes and Relu activation functions, and a linear layer at the output with 2 nodes. The input to the RNN is the estimated current point, $\hat{x}_i$, and the rate of rotation, ω. The RNN outputs the estimate of the next point, $\hat{x}_{i+1}$. We are forcing the state of the RNN to be the estimate of the next point concatenated with the rate of rotation; state $= [\hat{x}_{i+1}, \omega]$ so that state then becomes the input to the next run of the RNN.

The RNN is trained for different rates of rotation, ω, applied to sequences of 100 points. The rate of rotation is a uniform random variable drawn from $0 - \frac{1}{50}$. We use mean squared

Figure 3.2: Nonlinear neural network: follow a circle for a different CFO rates.



error of the true data sequence and the estimated data sequence for the loss function; $||\vec{x}-\vec{\hat{x}}||^2$. The network is trained for $10k$ epochs, each with a batch size of $1k$ data sequences and with a learning rate of 0.01. The initial starting point, x_0 is a uniform random variable drawn from on the unit circle. The network is tested on $100k$ new data sequences with random ω. Figure 3.2 shows the results of an RNN trained and tested for $10k$ epochs. The network achieves a loss on the order of 10^{-5}.

3.2 Deep Network Carrier Frequency Offset Estimation and Correction

We explore estimating carrier frequency offset and correction with neural networks. One of the challenges with CFO is that it cannot be directly measured in the real world. Therefore, all of the loss functions will not depend on ω. Again, we want a neural network that does not need to update its weights every time it encounters a new rate of rotation.

The inputs of the neural network are the known preamble, $\vec{x}_{pre}$, and the received preamble, $\vec{\hat{x}}_{pre}$. The neural network outputs the estimated preamble symbols, $\vec{\hat{x}}_{pre}$. The loss function is the mean squared error between the original preamble and the estimated preamble, $||\vec{x}_{pre} - \vec{\hat{x}}_{pre}||^2$. The neural network architecture consists of two non-linear layers, each with 10 nodes and sigmoid activations. This then feeds into a linear layer that outputs a scalar, treated as the estimate for the rate of rotation, $\hat{\omega}$. The received preamble is then rotated using a special layer with the function $e^{-ij\hat{\omega}}$.

Figure 3.3 compares the performance of the neural network correcting CFO versus no CFO correction. The training and testing data are solely for a one tap channel and have varying values of ω. The preamble length is kept constant at 100 symbols in QPSK. The neural network is trained for $15k$ epochs with batch size of 100 sequences and with a learning rate of 0.001. The neural network and no CFO correction are tested on $10k$ sequences with different rates of rotation, ω. The rate of rotation, ω, is a uniform random variable, $[0, \frac{1}{100}]$. The training and testing process is repeated for each SNR. Note, we re-train the network

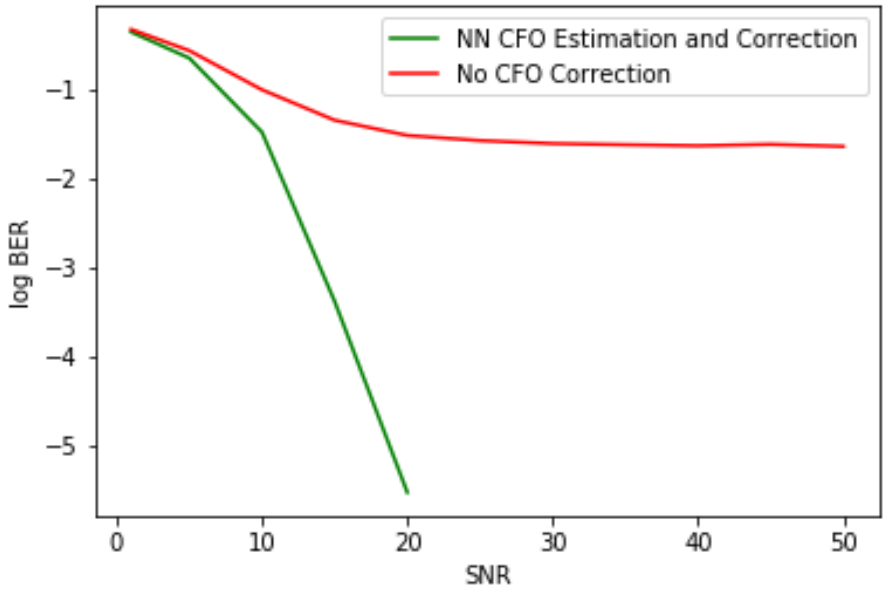


Figure 3.3: The log BER of received signals passed through a neural network CFO estimator with a rotation matrix CFO correction and classic demodulator. The log BER of the same signals passed through just the classic demodulator.

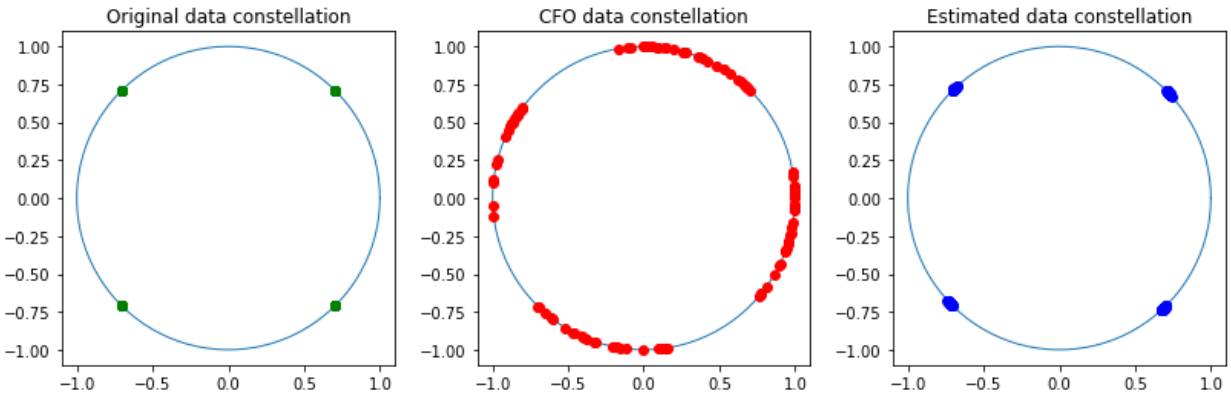
for each SNR. We compare the bit error rate of the neural network and classic demodulator versus the bit error rate of just the classic demodulator without any CFO correction.

For SNR above 25, the neural network does not have any incorrect bits in the $10k$ data sequences each with 100 symbols, or 50 bits. The neural network consistently performs better than just the classic demodulator for all SNR. Although, the performance is similar for SNR= 1.

Figure 3.4 shows one example of a data sequence passed through the neural network described above. The neural network is trained for SNR= 0. The three plots show that the neural network corrects for CFO quite well.

In this chapter, we demonstrated that recursive neural networks can act like clocks; they can follow a circle for a given rate of rotation. They can learn to learn to follow a circle for a given rate. We have also demonstrated that a neural network can learn to learn how to estimate CFO and correct for it. All of the simulations in this chapter were for single tap channels. We attempted to learn how to estimate CFO and correct for it when there were random two tap channels. However, the results were not promising and will need to be explored more in future work.

Figure 3.4: Nonlinear neural network estimating and correcting CFO for different ω. In this particular test example, $\omega = 0.00969545$ and SNR$= \infty$.



Chapter 4

Conclusion

In this paper, we reviewed the current literature on applying machine learning techniques to communcations systems; specifically equalization and carrier frequency offset. We also discussed the value of robustness and adaptability in future communications systems requiring an investigation of meta-learning. We demonstrated that neural networks can learn to learn to estimate channel characteristics for two tap channels. We explored the using deep recursive neural networks to learn to learn to equalize. We also used recursive neural networks to learn to learn to be a clock by rotating in a circle at different rates. Lastly, we used deep neural networks to estimate the rate of carrier frequency offset and correct it.

All of this without using backpropagation when faced with a new environment. Communications is a perfect application to explore the possibilities and limitations of the current meta-learning and machine learning frameworks.

4.1 Future Work

In the future, we would like to expand our channel estimation networks to handle any number of channel taps and channel taps that also have imaginary parts. We would also like to explore how we might be able to learn to estimate the channel even without a shared preamble.

We hope to explore and expand our research on equalization networks. One first step will be investigating how adding logarithmic layers into the networks affects the performance. We also want to expand our CFO estimation and correction to work with multi-tap channels and eventually combine all of the aspects in this paper to create one deep equalizer network.

In general, this paper was more of an exploration into how neural networks can learn to learn to communicate for equalization and CFO. We would like to do a more thorough architecture search and longer training runs. We also hope to to verify the ideas presented in this paper by training on real radio data instead of the simulated data used in this paper.

Bibliography

[1] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. "Learning a synaptic learning rule". In: *Technical Report 751, Départment d'Informatique et de Recherche Opérationelle, Université de Montréal, Canada* (1990).

[2] Corina Botoca, Georgeta Budura, and Miclau Nicolae. "Nonlinear Channel Equalization Using Complex Neural Networks". In: *Seria Electronică Şi Telecommuncaţii, Transactions on Electronics and Communications* (2004), pp. 227–231.

[3] John P. Costas. "Synchronous Communications". In: *Proceedings of the IRE* 44.12 (1956), pp. 1713–1718.

[4] Theo Diamandis. "Survey on Deep Learning Techniques for Wireless Communications". In: *Departement of Electrical Engineering, Stanford University* (2017).

[5] Sebastian Dörner et al. "Deep Learning-Based Communication Over the Air". In: *arXiv preprint arxiv:1707.03384* (2017).

[6] Nariman Farsad and Andrea Goldsmith. "Neural Network Detection of Data Sequences in Communication Systems". In: *IEEE Transactions on Signal Processing* PP (2018).

[7] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *International Conference on Machine Learning* (2017).

[8] Hengtao He et al. "Model-Driven Deep Learning for Physical Layer Communications". In: *arXiv preprint arxiv:1809.06059* (2018).

[9] Hyeji Kim et al. "Communication Algorithms via Deep Learning". In: *International Conference on Learning Representations* (2018).

[10] Hyeji Kim et al. "Deepcode: Feedback Codes via Deep Learning". In: 2018.

[11] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science*. Vol. 350. 6266. 2015, pp. 1332–1338.

[12] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. "Metalearning: a survey of trends and technologies". In: *Artificial Intelligence Review*. Springer, 2015, pp. 117–130.

[13] Timothy J. O'Shea, Kiran Karra, and T. Charles Clancy. "Learning to Communicate: Channel Auto-encoders, Domain Specific Regularizers, and Attention". In: *IEEE International Symposium on Signal Processing and Information Technology*. 2016, pp. 223–228.

[14] Timothy J. O'Shea, Tamoghna Roy, and Nathan West. "Approximating the Void: Learning Stochastic Channel Models from Observation with Variational Generative Adversarial Networks". In: *arXiv preprint arxiv:1805.06350* (2018).

[15] Timothy J. O'Shea et al. "Radio Transformer Networks: Attention Models for Learning to Synchronize in Wireless Systems". In: *arXiv preprint arxiv:1605.00716* (2016).

[16] Timothy O'Shea, Tugba Erpek, and T. Charles Clancy. "Deep Learning-Based MIMO Communications". In: *arXiv preprint arXiv:1707.07980* (2017).

[17] Timothy O'Shea and Jakob Hoydis. "An Introduction to Deep Learning for the Physical Layer". In: *IEEE Transactions on Cognitive Communications and Networking*. Vol. 3. 4. 2017, pp. 563–575.

[18] Timothy O'Shea, Kiran Karra, and T. Charles Clancy. "Learning Approximate Neural Estimators for Wireless Channel State Information". In: *arXiv preprint arXiv:1707.06260* (2017).

[19] K. T. Raghavendra and Amiya K. Tripathy. "An Efficient Channel Equalizer Using Artificial Neural Networks". In: *Neural Network World* 16 (2006), pp. 357–368.

[20] Tianqi Wang et al. "Deep Learning for Wireless Physical Layer: Opportunities and Challenges". In: *China Communications* 14.11 (2017), pp. 92–111.

[21] Hao Ye, Geoffrey Ye Li, and Biing-Hwang Juang. "Channel Agnostic End-to-End Learning based Communication Systems with Conditional GAN". In: *IEEE Global Communications Conference*. 2018.

[22] Hao Ye, Geoffrey Ye Li, and Biing-Hwang Juang. "Power of Deep Learning for Channel Estimation and Signal Detection in OFDM Systems". In: *IEEE Wireless Communications Letters*. 2018, pp. 114–117.