Equalization of Underwater Optical Communication Signals with Artificial Neural Networks

Department of Electrical, Electronic and Computer Engineering

An Honors report submitted for the degree of:

BEng(Hon) in Computing and Electronics

Author: Bernard Hollands

Supervisor: Dr Mauro Dragone

13 April 2021

Abstract

In underwater communication the non-linear noise, distortion and interference experienced by optical signals between the transmitter and receiver leads to a high bit error rate over the communication channel. To mitigate this when the signal is received it needs to go through an equalization process to remove these errors, this process is typically done using a form decision feedback equalizer. This report tests three alternatives to this equalizer in the form of trained artificial neural networks in an effort for a faster less computationally complex solution.

This project involved the training and testing of a Multi-Layer Perceptron Neural Network, Long Short Term Memory Network and Deep Echo State Network to this task of equalizing the received optical communication signals on an initial low constellation PAM2 data-set with a high error rate. It will go on to show that from testing on this initial data-set the Deep Echo State Network proved most promising in terms of required pre-processing and performance. This was followed by a further investigation into the the effectiveness of the Deep Echo State Network on more complex signals and data-sets at multiple speeds. This project was able to show that the Deep Echo State Network is adept at the task being able equalize the signals faster in all cases and able to outperform the conventional decision back equalizer performance on half of the data-sets made up of different formats including PAM4, PAM8 and PAM16 running at various bit rates.

Acknowledgments

I would first like to express my thanks and appreciation to my supervisor Dr. Dragone for his advice, recommendations and input over the duration of this project. I would also like to express my gratitude for the help I received from the Underwater Optical Communication research group, their advice and ideas proved invaluable. I would also like to thank them for providing me with the data-sets used throughout this project. Lastly I would like to thank my family for their continued support over the past four years.

Declaration of Authorship

I, Bernard Hollands, H00271461,

State that this work submitted for assessment is my own and expressed in my own words. Any uses made within if of works of other authors in any form (e.g Idea, figure, text, tables) are properly acknowledged at their point of use. A list of the references employed is included.

Date: 13 April 2021

Signed: BERNARD HOLLANDS

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Visible light communication for underwater vehicles and communication buoys that monitor seabed equipment and submerged environments such as pipelines has been becoming increasingly popular form of communication. This is due to their dramatically increased bandwidth over the more traditional acoustic communication enabling the potential possibility for advantages for high bandwidth applications such as live video [1]. Currently optical links can only be used over short distances as they are far more susceptible to non-linear distortion, noise and inter-symbol interference due to environmental factors such as currents, particles, and absorption in the water [1]. This interference requires the signal to be equalized once it is received, this process can be slow and inefficient using conventional methods such as a Decision Feedback Equalizer. The use of trained machine learning and artificial neural network model equalizers have been emerging as an alternative solution as they can provide fast low complexity solutions to this problem saving on computational power and energy [2]. Artificial neural networks being a machine learning model means they can also be trained to specific channels and be re-trained if their are major changes to the environment.

1.1 Aim

The aim of this project was to train an Artificial Neutral Networks (ANN) to equalize Underwater Optical Communication (UWOC) signals faster and more efficiently than a conventional Decision Feedback Equalizer.

1.2 Objectives

The objectives of this project are to implement, train and test the following styles of networks for UWOC signal equalization:

1. Multi-Layer Perception Neural Network (MLP)
2. Long-Short Term Memory Network (LSTM)
3. Deep Echo State Network (DeepESN)

Then based on preliminary tests perform further testing on more data formats and bit rates on the most promising network to find optimal settings and amounts of training data.

1.3 Project Outline

Over the course of the project each style of network was implemented using machine learning libraries and APIs. The MLP and the LSTM were implemented in the keras API [3] for TensorFlow, a machine learning library developed by Google [4]. The Deep Echo State Network was implemented with the DeepESN library [5] developed by the Computational Intelligence and Machine Learning (CIML) Group in the University of Pisa. All of the artificial neural network programs were executed in python 3.8.8 while the Bit Error

Rate Calculation and Decision Feedback Equalizer (DFE) was run in MATLAB R2020B. All networks were trained and tested initially on their capability using a PAM2-600Mbps data-set provided by the Underwater Optical Communication Research Group. The network performance was evaluated through both the Mean-Squared Error (MSE) of the output as well as the Bit Error Rate (BER). The eye diagrams and network outputs were also examined for their quality as a visual indicator. Optimal training settings for each network were found through experimentation with the hyperparameters (learning parameters) through sweeps of values one at a time.

It was found that the DeepESN performed overall the best in terms of its ability of be trained on much fewer amounts of data and its computational efficiency during training. The MLP was also very effective but required a lot more data to perform at the same level as well as the data requiring an extra pre-processing steps such as sliding windows. For this reason as well as the fact that the DeepESN has less overall research for this application it was decided to keep testing the DeepESN on the higher constellation data-sets. The DeepESN was quite effective on the more complex data-sets being able to improve the signal quality and perform the task quicker on all signals cases and able outperform the DFE in terms of BER in half of the data-sets.

Chapter 2

Background Information

2.1 Underwater Optical Communication

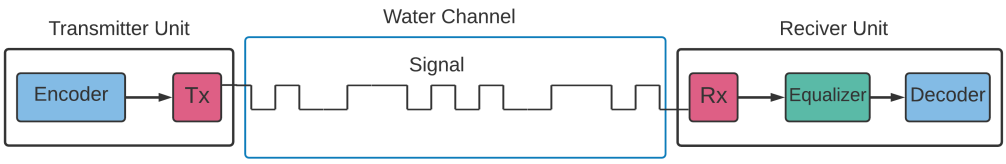


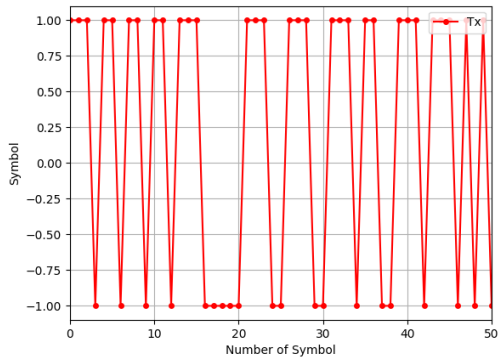
Figure 2.1: Block diagram of UWOC system

Wireless underwater communication is the process of sending unguided wireless signals via an open body of water. There are 3 common methods used to achieve this, acoustic signals, radio frequency and optical communication [1].

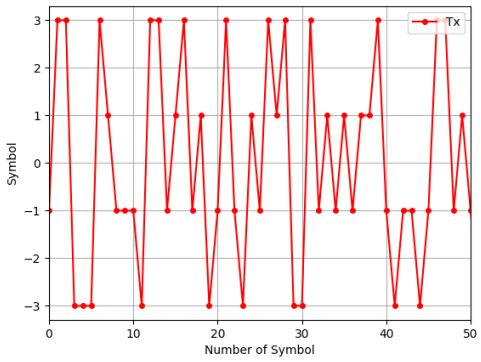
The focus of this project was optical communication. Optical communication signals are typically generated by modulating light emitting diodes (LEDs) and laser diodes [1] in the spectral range of 450nm to 550nm [6] to be picked up by a receiver. Typically this modulation is done via on-off keying (OOK) due to its simplicity [7]. Optical communication is by far the fastest method of underwater wireless data transmission being able in some cases able to operate on the order of Gigabits per second (Gbps) [1]. This is however limited to only 10s of meters with a direct line of sight and the signal is unable to pass from air and water

and visa versa. A major challenge for Underwater Optical Communication is the distortion and inter symbol interference (ISI) that a signal can experience between the transmitter and receiver [1] making the signal hard to decode. This can be caused by the water channels moving with current, absorption and scattering of the light creating non-linear distortion [1].

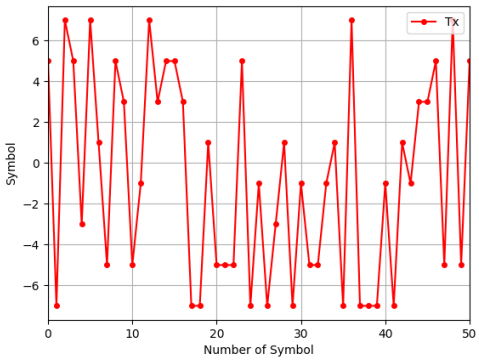
2.2 Description of PAM Signals



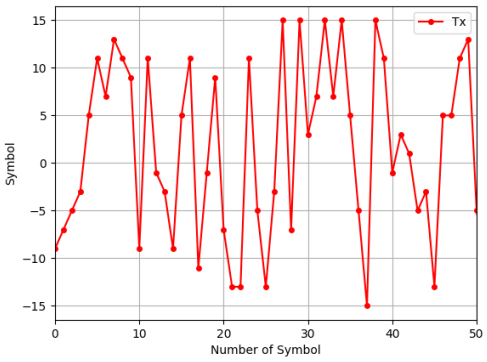
(a) PAM2 Signal



(b) PAM4 Signal



(c) PAM8 Signal



(d) PAM16 Signal

The type of signal used for this project is a Pulse Amplitude Modulation (PAM) signal. For PAM, the signal is modulated between 2 or more positions encoding the data at different levels where each level is represented by a symbol as can be seen in Figure 2.5a, b, c, and d. The number of levels is called the constellation as it describes the number of positions that the signal modulates between. 6 different data-sets are used for this project with a variety constellations and bit-rates, the

signals used are described in Table 2.2.

Each symbol in the signal represents grey code so only a single bit changes between each level E.g PAM4 symbols as shown in Figure 2.5b symbols are -3, -1, 1, 3 and are encoded as 00, 01, 11, 10 respectively. When the signal is received the it is demodulated creating a binary series. Due to the signal distortions these symbols and get misinterpreted during demodulation and create an incorrect received signal. The finding the ratio between the send and received binary series is how the bit error rate is calculated.

Data-set	Constellation	Bit Rate	Distance	Size	Raw BER
PAM2-600Mbps	2	600 Mbps	60m	1004020	0.2626
PAM4-125Mbps	4	125 Mbps	27m	247494	5.569×10^{-3}
PAM8-94Mbps	8	94 Mbps	27m	123744	6.222×10^{-4}
PAM8-188Mbps	8	188 Mbps	27m	247494	5.198×10^{-2}
PAM16-125Mbps	16	125 Mbps	27m	123744	1.971×10^{-2}
PAM16-250Mbps	16	250 Mbps	27m	247494	0.1447

Table 2.1: Descriptions of data-sets

2.3 Equalization

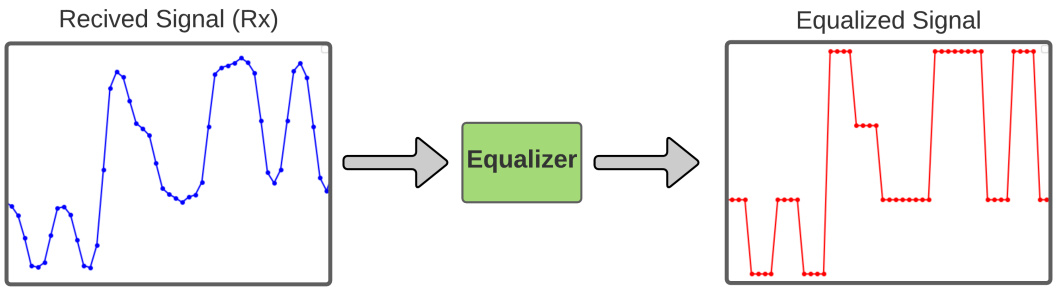


Figure 2.3: Block Diagram of Equalization

Equalization is a very important process in almost any communication system [8] UWOC is not exempt from this, equalization describes the process of removing noise and/or distortion from a communication signal so it can be decoded properly and accurately. Equalization for UWOC is typically achieved by a Decision Feedback Equalizer (DFE) [9] as it needs to remove non-linear

distortions. This is an important process as when a signal arrives if it is not equalized then the decoded signal will contain lots of errors and interference compared to what was sent. Figure 2.4 shows the difference between the transmitted (Tx) and received (Rx) signals on the PAM2-600Mbps data-set used in this project that has a high error rate.

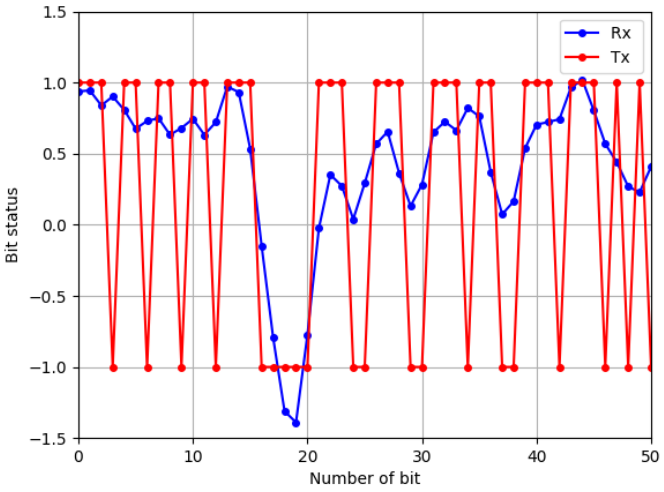


Figure 2.4: Raw PAM2 Rx and Tx signals

2.4 Review of Machine Learning Equalizers

There are many types of machine learning based equalizers described in literature, many are based off of different concepts, architectures and training types and built for different signal formats. This section will briefly cover some of the most popular Machine Learning techniques for equalization of non-linear communication channels and their specific application.

The most common models and networks used are Multi-Layer Perceptron (MLP) [10][11][12], Long Short Term Memory network (LSTM) [13][14] Gaussian Mixture Model (GMM) [15], as well as echo state networks [16][17]. Each method has found success being able to match or beat its counterpart in some form. See Table 2.2 for a summary.

Multi-Layer Perceptron Network

The Multi-Layer Perceptron (MLP) is a versatile feed forward neutral network made up of layers of perceptions, a form of binary classifier. Each layer is totally connected to the next. Training methods can vary depending on the the application however back propagation with a variation of stochastic gradient decent are often used. A more detailed description is available in later sections.

Haigh et al [10] applies this network to equalize a 20 Mbps visible light communication link using on-off keying of a Polymer LED at a frequency of 350kHz over a distance of 0.05m. The network had a single hidden layer where 5, 10, 20, 30 and 40 neurons were trained using a Levenberg-Marquardt back propagation optimiser. The training length was varied from 10^3, 10^4, 2^{14}, 2^{15}. The raw signal had an average BER of 0.2 at up to 6 Mbps. The equalizer was able to achieve a BER of 10^6 with 40 neurons at a bit rate of 19Mbps. However even with a hidden layer size of 5 a BER of 10^6 was still achieved at 8Mbps.

Long-Short Term Memory Network

A Long-short Term Memory Network (LSTM) is a form a recurrent neural network where the outputs are also combined with the inputs enabling the network to "remember" patterns. A further detailed breakdown is available in chapter 5. Wang et al [13] implements the LSTM network via an FPGA for purpose of signal recovery as an alternative to the DFE. The network is trained via back propagation. Wang et al was able to experimentally demonstrate that for a signal of 10Gbps an LSTM based equalized was able to recover the signal and was able to conclude that their LSTM PCB with 20 hidden units and 15 delays had good signal recovery ability in their test scenarios able to outperform the DFE. It is stated that a main limitation of the LSTM equalizer lies in its computational complexity when compared with a Traditional DFE.

Deep Long-Short Term Memory

A Deep LSTM uses the same LSTM units however instead of operating in a single line, the output goes to a layer of neurons, and then again to another layer of LSTM cell. Lu et al [14] proposes this form of network to equalize a PAM8 visual light communication signals at 1.15Gbps as an alternative to a volterra based equalizer due to its lower computation complexity. The volterra equalizer had a complexity of $O(n^2)$ where as the LSTM only had a complexity of $O(n)$. They report the LSTM being able to achieve an acceptable BER of 3.8×10^{-3} at a distance of 0.8m for the PAM8 signal.

Echo-State Network

The Echo-state network (ESN) is a form of reservoir based recurrent neural network, where the inputs echo in a sparsely connected reservoir and only the output layer is trained. Bauduin et al [16] applies this task to the equalization of non-linear satellite communication as an alternative to a volterra equalizer. A style of ESN where the reservoir matrix is represented as a circular matrix and replace the hyperbolic tangent with a polynomial function of order 3 to reduce complexity. When simulating with 16-QAM (Quadrature amplitude modulation) signals it was observed that the ESN would perform better with more neurons in the reservoir testing 100, 300 and 1000. 300 and 1000 neurons performed the best being able to achieve a BER of approximately 2×10^{-5} where the volterra achieved 1×10^{-5}. They were able to show that an ESN with a slightly higher complexity then a volterra equalizer can achieve a similar performance.

Gaussian Mixture Model - Hidden Markov Model

A Gaussian mixture model (GMM) [18] and Hidden Markov Model (HMM) [19] are supervised machine learning models used commonly for pattern recognition

tasks such as speech recognition [20]. Tian et al [15] applies these models to equalization of non-linear PAM4 optical fiber transmission signals as a lower complexity alternative to recurrent neural networks with similar BER performance. For testing the data-sets were split 50% for training and 50% for testing. They were able to achieve at an appropriate receiver power a BER up-to 1×10^{-1} with both a RNN and GMM-HMM. However with appropriate settings the GMM-HMM is at least 73% lower complexity than the RNN. Both the RNN and GMM-HMM were able to in some form outperform the volterra equalizer.

Summary Table

Model	Recurrent	Symbol Rate	Format	Viable
Deep LSTM	Yes	0.9Gbps	PAM8	yes
LSTM	Yes	10Gbps	PAM2	yes
MLP	No	20 Mbps	PAM2	yes
ESN	Yes	33Mbps	16-QAM	yes
GMM	No	56 Gbps	PAM4	yes

Table 2.2: Overview of current Machine Learning Equalizers

2.5 Performance Metrics

Before performing any experiments it was crucial that the right performance and metrics were used to evaluate the performance of each network. The chosen metrics needed to give a consistent and measurable number as well be comparable number between different runs and style of network. The metrics chosen to evaluate the performance was the Mean-Squared Error (MSE) and the Bit Error Rate (BER). The Bit Error Rate is the key metric in communication and was used as the primary metric in this project. Visual indicators were also used such eye diagrams of a signal. Both of these equations together and the eye diagrams give a clear picture of how well a model has learned the problem.

2.5.1 Mean-Squared Error

The Mean Squared Error equation [21] is a common loss function in machine learning that determines how close the outputted points are to their target. The lower the MSE is to zero the closer the output is to a perfect result. The MSE was used to evaluate the output before demodulation. The equation is as follows

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i)^2 \tag{2.1}$$

where n is the total number of data points, Y_i is the observed value and $\hat{Y}_i$ are the predicted values.

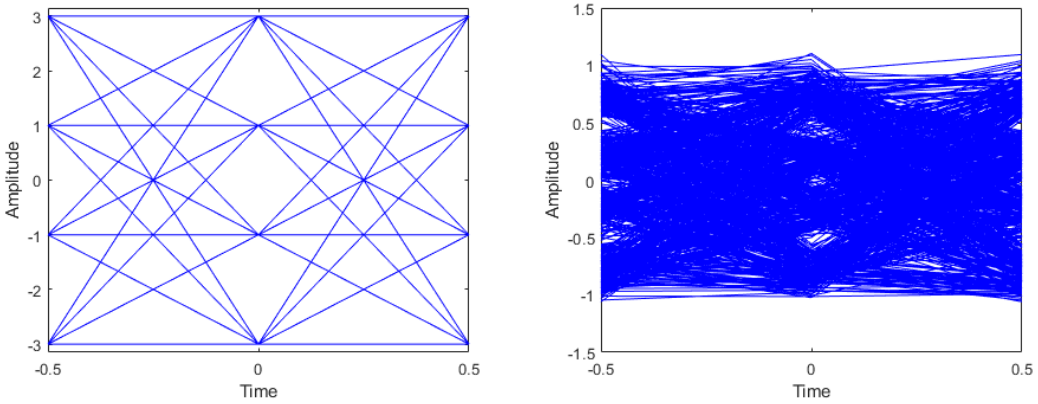
2.5.2 Bit Error Rate

The Bit Error Rate (BER) [22] is a very simple but very important metric in communication systems to determine how many bits have been lost over a channel.

$$BER = \frac{E(T)}{N(T)} \tag{2.2}$$

where E(T) is the number of bit errors received in time T and N(T) is the total number of bits received. The BER is measured on a range from 0 to 1, the closer to 0 the more accurate the signal. The BER was calculated after the signal had been demodulated into a binary series.

2.5.3 Eye Diagrams



(a) Ideal PAM4 Eye diagram

(b) High BER PAM4 Eye Diagram

An eye diagram [23] is a standard visual representation of a communication signal where it is easy see the general quality. Received symbols are mapped to one another showing the transitions between them in the series showing a subsection of or the entire signal in a single image.

The diagram is named after the way to looks as accurate signals will have open eye's down the center with dense transition lines. Signals with higher error rate will have smaller and messy eyes as can be seen in Figure 2.5b. An eye diagram for a PAM signal has a distinct number of levels corresponding the the constellation i.e PAM4 has 4 levels and PAM8 has 8 levels. In this project eye diagrams are used to look at the received signals before equalization and after via both the DFE and neural networks. If the signal have been equalized well then the eye diagram is much improved. All the the signal eye diagrams for this project can be seen in Appendix E.

Chapter 3

Data Pre-processing

Pre-processing in this context describes the processes that the data goes through prior to being used to train and test a neural network. Pre-processing is done to ensure that the networks will interact with the data correctly and train as effectively as possible and has a great effect on the outcome and general performance [24]. Different networks require different processes and there are many processes that can be performed. The key data processing methods chosen for the networks in this project were Normalisation, Train-Test Splitting and Sliding Windows.

3.1 Normalisation

Normalisation [25] is the process of bringing all the data to the same scale without disturbing or distorting the relationship between them. Normalisation is often an important process in machine learning as if the input and target data are in different scales it can drastically reduce the accuracy. Data often also need to be within the range of 0 and 1 to properly interact with the activation function(s) in neural networks such as an LSTM or MLP. For this the project min-max scaling equation [26] was used on the inputs and targets. The equation is as follows

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.1}$$

Where X' is the normalised value, X_{min} is the minimum value in the series, X is the original value and X_{max} is the maximum value in the series.

The scale of the output from the network is also important as to gain a proper MSE between data-sets. If the data is in a different scale to an another e.g higher constellation data-set is used, even if there is approximately the same degree of error the MSE would be larger for the wider scale not giving a clear picture. Once the MSE of the output is calculated then the inverse of this transform is taken to bring it back to the original scale and the Error Rate can be calculated.

3.2 Train-Test Splitting

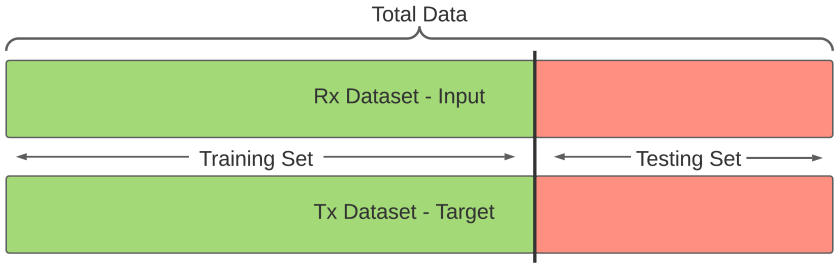


Figure 3.1: Example of train test split

Train-test splitting is a standard process in supervised machine learning where the data-set is split from 2 groups, in this case Transmitted (Tx) and Received (Rx) time series into 2 sets of inputs and targets using the Rx as the input to be equalized and Tx as the target value. One set will be used to train and optimise the network to the problem with the other being used to test and evaluate the performance of the trained network. It was important when performing this process that the testing sets not be embedded in training sets as that would bias the outcome. For the most reliable results the testing inputs and targets must be unseen. If data that the network has been trained on is used for testing then the

network will know how to deal with it and not give an accurate measure of how well the network has been trained to the problem.

The way and ratios of the train test split can have quite an impact on the overall performance in machine leaning applications [27]. As both the amount of training and testing data are adjusted so is the unmodified MSE and BER in the test set.

To test the amount of training data required to train a specific network this style of data splitting was not used. When Train-Test splitting is used the size of the training set is changed according to the size of the testing set and subsequently the unequalized BER and MSE do not stay consistent while adjusting the amount of training data. This would lead to not being able to compare the effectiveness of different amounts of training data between experiments to determine the optimal amount. To solve this the testing set was held at a static size of 100k data points while only the training size changed. Once optimal amounts of training data was determined then the typical train test split was used to utilize all of the available data for testing.

3.3 Sliding Windows

Sliding Windows [28] or windowing is a relatively common processes when preparing time series data to be input into a non-recurrent machine learning model or network. A sliding window can dramatically increase the effectiveness during training and overall performance as demonstrated in later sections. Sliding windows works by defining an input windows size that encapsulates a given number of data points and defines the following point as the target. This enables the network to learn the patterns and numbers that lead to a certain result. This is critical as non-recurrent networks such as the MLP are unable to remember patterns that have passed though them. Figure 3.2 shows a visualisation of sliding sliding windows with a window size of 3 on a time series of 1 to 10 of positive integers.

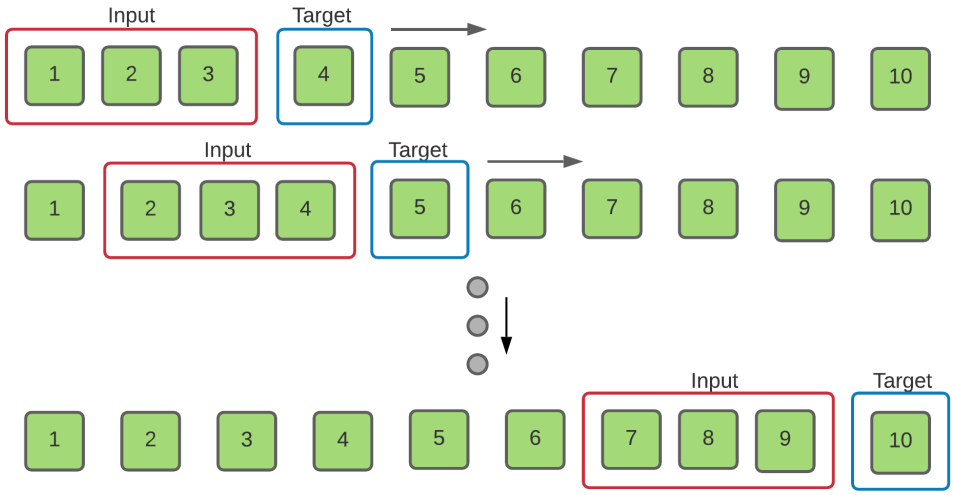


Figure 3.2: Visualisation of Sliding Windows

Chapter 4

Neural Network Descriptions

The Neural Networks selected for this project was the Multi-Layer Perceptron (MLP) neural network, Long short Term Memory network (LSTM) and a Deep Echo State network(DeepESN). The MLP was selected because it has been becoming a more common and proven capable equalizer if trained correctly [29]. The LSTM was selected as an option as it has also previously shown capability in equalization tasks as well as time series prediction [30] due to its recurrent nature and advantages over a standard recurrent network. The DeepESN was chosen as while it has not yet been tested for equalization, standard shallow echo state networks have been shown to perform well in equalization [16]. The DeepESN has also shown impressive results in non-linear time series prediction tasks and the handling of time series data [31][32].

4.1 Multi-Layer Perceptron

Multi-Layer Perceptron neural networks [33] are arguably one of the most common forms of feed forward artificial neural network (ANN) due to their classification ability and versatile nature. As the name suggests the model is made up of multiple layers of perceptions or neurons that are entirely connected

to the next layer and the data if fed through as shown in Figure 4.1. Each perceptron has a set of weights on their input as well as a bias that get modified through training enabling the network to change an input to a specific output.

There are multiple ways to train a MLP network, a majority of and the most common techniques are based on back propagation and stochastic gradient descent [34]. Variations of these techniques were used for this project.

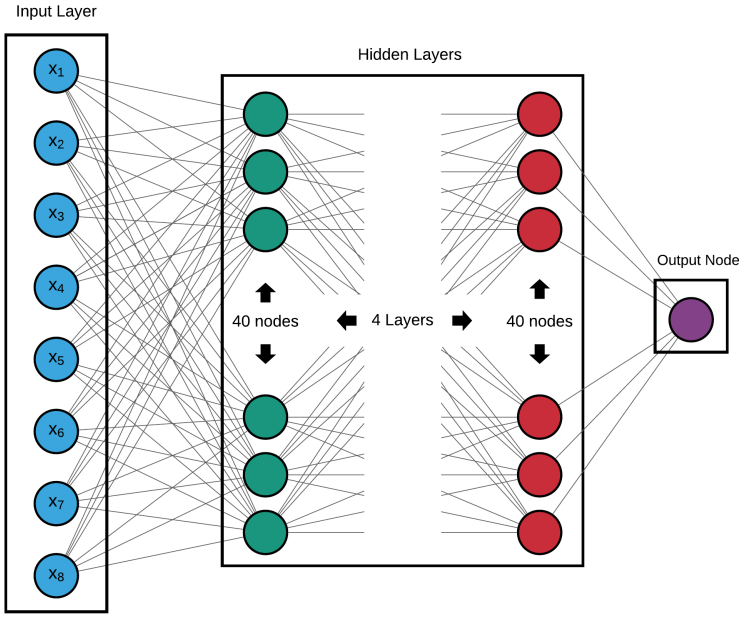


Figure 4.1: Multi-Layer Perception Network

4.1.1 Model Structure

Perception

The perceptron [35] is what the MLP is constructed of. The perceptron itself is a linear binary classifier and is capable of learning however combing them can greatly their overall capability even being able to handle non-linear data. Their structure can be seen in Figure 4.2. By putting the inputs from data-set into the perceptron through weighted connections as described by Equation 4.8 followed by the activation function the output is produced and sent as an input to the

perceptions in the next layer until it reaches the output node. The weights of the inputs and the bias are all initially randomly distributed between 0 and 1 and are updated though training. The equation is

$$y = \sum_{i=1}^{n} w_i x_i + b \tag{4.1}$$

Where w_i is the weight of the connection, x_i is the value from the previous perceptron and b is the bias. The value is then multiplied through the activation function.

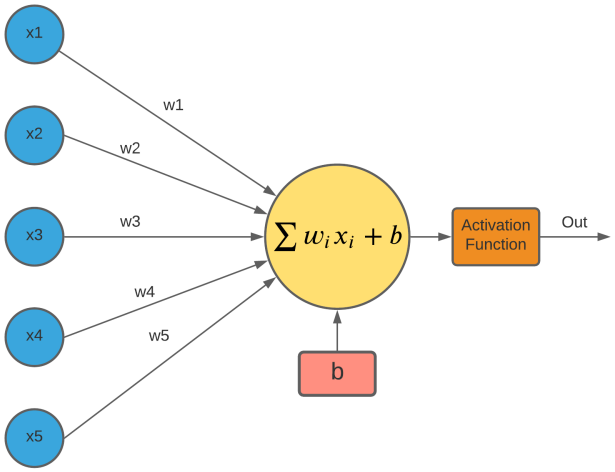


Figure 4.2: Example Perceptron [36]

Activation Functions

As mentioned after the input values have gone through the perceptron it must be then passed through an activation function. This determines at what threshold the neuron actives and how much influence that it has [37]. Typically activation functions output a value between 0 and 1 or between -1 and 1 with the lower the outcome meaning less influence. Common activation functions used include a sigmoid function and hyperbolic tangent although there are many types of activation functions for many applications and situations [38]. The activation function of the output perceptron is particularly important as it

defines the range of the output. The best function for the situation of often found though experimentation on the data-sets and network.

The SeLU activation [39] proved to be the most effective for both data sets in the project giving consistently better result than any other and is described as.

$$selu(x) = \lambda \begin{Bmatrix} x & x > 0 \\ \alpha e^x - \alpha & x \leq 0 \end{Bmatrix} \tag{4.2}$$

Where λ and α constants are 1.051 and 1.673 respectively

Training Algorithm

Much like the activation functions there are many types of training algorithms. As mentioned most training algorithms operate on back propagation with gradient decent. During training there is a forward pass where the network produces a result, this result is compared with the perfect target and an amount of error is calculated (often MSE). This error is then propagated back through the network and all the weights and bias are updated according to how much they caused the error. It is this process that benefits from the normalization of the input data to ensure all inputs are treated the same and uniform error can be calculated. All of the training algorithms tested for this project operated on this principle and can be read about in detail here [34]. The most effective training method found for this project was Adaptive Moment Estimation (ADAM) [40] witch combines the advantages of RMSprop [34] and Stochastic Gradient Descent [34].

4.2 Long-Short Term Memory Network

A long short term memory network [42] is a common form of supervised recurrent neural network that was developed to over come the problem with remembering

long term dependencies in traditional Recurrent Neural networks [41].

The LSTM is made up of units/cells that take in each data point in the time series that feed into one another giving it recurrent properties as shown in Figure 4.3 [41]. The fact it is recurrent means that it can "remember" what has passed through and is able it identify patterns during training. The fact that it is recurrent also means that sliding windows is not required when inputting the data unlike the MLP network. However normalization is still crucial for proper training [43].

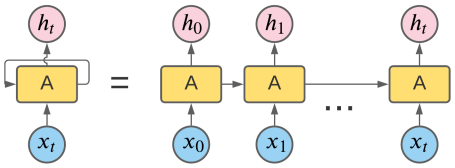


Figure 4.3: Unrolled LSTM [41]

The core idea behind an LSTM is the cell state C_t, a vector that runs through all the units with only small interactions and the gates that govern that behavior that comes from the previous unit. This allows information to pass through relatively unchanged. The only way to add or remove from the cell state is the through neural network gates. They are made up of a sigmoid layer indicating how much information to let through. There are 3 gates in an LSTM, The forget Gate, Input Gate and Output Gate.

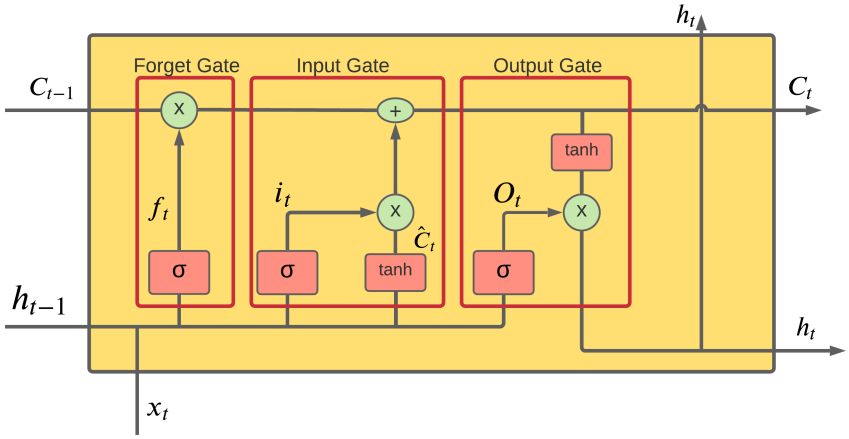


Figure 4.4: Example LSTM Unit [41]

Forget Gate

The forget gate is responsible for keeping or removing information from the previous unit and operates between 0 and 1. A 0 is used to keep nothing and 1 to keep everything and is described by the following equation.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{4.3}$$

Input Gate

The input gate updates the cell state of the unit, the previous hidden state h_{t-1} and the input x_t pass though the sigmoid to get i_t, to help regulate the network they are then are also passed though a hyperbolic tangent to get $\hat{C}_t$ and multiplied together and added to the forget gate and is performed as follows.

$$i_t = \sigma(W_i x_t + U_{ri} h_{t-1} + b_i) \tag{4.4}$$

$$\hat{C}_t = tanh(W_c x_t + U_{rt} h_{t-1} + b_C) \tag{4.5}$$

$$C_t = f_t \otimes C_{t-1} + i_t \otimes \hat{C}_t \tag{4.6}$$

Output Gate

The output gate determines the next hidden state h_t. It is calculated by passing the previous hidden state h_{t-1} into a sigmoid function with the current input to get O_t witch is multiplied with the current cell state after going through a hyperbolic tangent.

$$o_t = \sigma(W_o x_t + U_{ro} h_{t-1} + b_o) \tag{4.7}$$

$$h_t = O_t \otimes tanh(C_t) \tag{4.8}$$

W are the neural network weights, U_r are the recurrent weights and b is the bias that are all trainable. Training much like the MLP is performed using back propagation and gradient descent variations [44], with ADAM [40] specifically being used for this project.

4.3 Deep Echo State Network

This section will introduce the idea of the Shallow Echo state Network that is used as the basis for Deep Echo State Network as well as the deep echo state network itself.

4.3.1 Shallow Echo-State Network

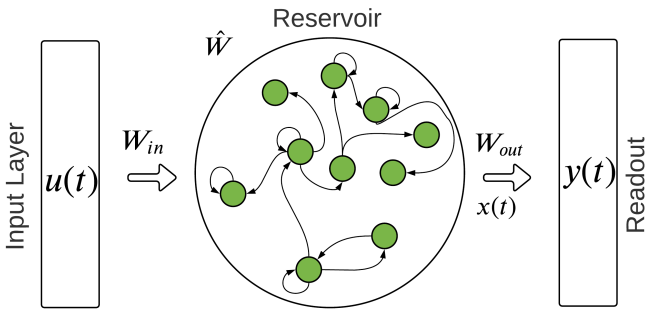


Figure 4.5: Shallow Echo-State Network Architecture [45]

The Shallow Echo-State Network (ESN) [46] is a form of supervised recurrent neural network that uses a sparsely connected reservoir as its hidden layer with only the output layer bring trained exploiting the temporal state representation generated by the reservoir [32]. The Shallow ESN that is used to build the Deep Echo State Network is based on the leaky-integrated Echo state network (LI-ESN). This is a version of an standard echo state network that has leaky

units integrated into the reservoir layer introduced by Jager, Lukosevicius and Popvici [47]

The connections and weights in this reservoir are fixed and cannot be changed once initialised, with the reservoir weights being randomly distributed between -1 and 1 and then scaled according to the spectral radius [32] [45]. Only the output layer of the network is trained through ridge regression using Singular value Decomposition (SVD) [48]. When establishing the reservoir other than the number of units, there are 3 key hyperparameters that have the greatest influence on the outcome and need to be optimised for the best performance these are Input Scaling (the distribution scale of the input matrix weights), Spectral Radius and with the introduction of leaky units the leaking rate. These parameters are typically found through trial and error as they are very task specific [45]. After the number of units one of the most key parameters for good performance is the spectral radius of the reservoir.

Spectral Radius

The spectral radius (ρ) refers to the maximum eigenvalue allowed in the reservoir matrix. It is responsible for "scaling the width of the distribution of it non-zero elements" [45]. When the reservoir is generated it's spectral radius is found and scaled to the specified value.

The size of the spectral radius is key as for an ESN as it must maintain its echo state property (ESP) [49][50][51]. The ESP is where the values of each neuron must depend on the past history of the inputs. A spectral radius that is too large can violate this rule and ruin the performance of the network. Typically to ensure the echo state the spectral radius must be less than 1 [45].

4.3.2 Deep Echo State Network

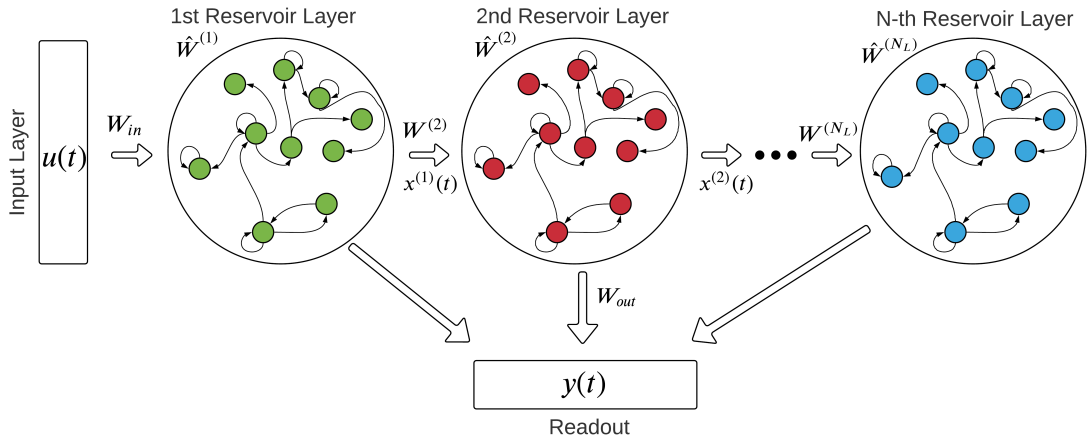


Figure 4.6: DeepESN Architecture [32]

A Deep Echo State Network (DeepESN) aims to bring deep learning to ESNs. The DeepESN network used for this project was introduced by Gallicchio, Micheli and Pedrelli [32]. The DeepESN is constructed of many reservoirs that feed their output into the next before feeding into the readout layer. This depth can lead to better a fit than a single ESN on its own. The number of layers (N_L) and units per layer (N_R) can be defined. In this project setup each layer contained the same number of units. The basic principle is that at every time-step t a value from the input matrix $u(t)$ is put into the first layer and each layer after receives the output from the last. The updating of the first layer is described by:

$$x^{(1)}(t) = (1 - a^{(1)})x^{(1)}(t-1) + a^{(1)}tanh(W_{in}u(t) + \hat{W}^{(1)}x^{(1)}(t-1)) \tag{4.9}$$

for every layer $i > 1$

$$x^{(i)}(t) = (1 - a^{(i)})x^{(i)}(t-1) + a^{(i)}tanh(W^i x^{i-1}(t) + \hat{W}^{(i)}x^{(i)}(t-1)) \tag{4.10}$$

where $x^i(t)$ is the state of a layer at time t, a is the leaky parameter in range

[0,1], W_{in} is the input weight matrix of layer size $N_R * N_R$ and $\hat{W}^{(i)}$ is the weight matrix of size $N_R * N_R$ of recurrent layer i.

The readout layer $y(t)$ is described as:

$$y(t) = W_{out}x(t) \tag{4.11}$$

where w_{out} is the output weight matrix and $x(t)$ is the global state of the model described as $x(t) = (x^{(1)}(t), ..., x^{N_L}(t))$. Much like the Shallow ESN the output weight matrix(W_{out}) is also the only set of weights that is trained. This is done through using the standard reservoir computing method [48] using ridge regression implemented using SVD.

The weight values for the recurrent matrices $\hat{W}^{(i)}$ are randomly assigned on the uniform distribution of -1 and 1 and re-scaled to maintain the echo state property where the spectral radius must be less than 1. Full details of DeepESN model used are available in [32].

4.3.3 Computational Complexity

DeepESN has the advantage of having a training lower computation efficiency and times than than most other deep learning models. By reducing the number of non-zero connections by removing the input connections to layers above the first as well as removing the connections from higher to lower layers [31] in combination with having fixed weights in the reservoir and only training the output layer. This leads to an training computational efficiency of $O(N_R^2 M_L^2 N_T)$ where N_T is the number of time steps, M_L is the number of layers and N_R is the number of units [32]. Once the network is training it is simply a series of matrix multiplications.

Chapter 5

Preliminary Results

In first stages of the project the goal was to determine which of the chosen networks the most promising network for the task of equalization. This was done by testing each of the networks using the PAM2-600Mbps data-set. This data-set was used because this is what was available as well as the fact that it had a high BER of 0.2626 with no equalization applied. This would give a good indication of a networks ability to very handle noisy data with a lot of ISI.

The best results from each of the networks from this initial testing stage can be seen in Table 5.1 and Figure 5.1. Overall the MLP performed the best, followed by the DeepESN and then the LSTM.

Metric	**DeepESN**	**MLP**	**LSTM**
BER	4.655×10^{-2}	1.939×10^{-2}	2.613×10^{-1}
Test MSE	0.175	0.047	0.253
Training Time	23s	45s	32s

Table 5.1: Metrics from PAM2-600Mbps Testing on all Networks

An interesting observation about the MLP was that the that Mean Square Error was much lower than either of the other networks tested even though the DeepESN result has a relatively close BER. A possible reason for this is that when the MLP was outputting the correct pattern it is very close to perfect, however it would sometimes get the pattern completely wrong.

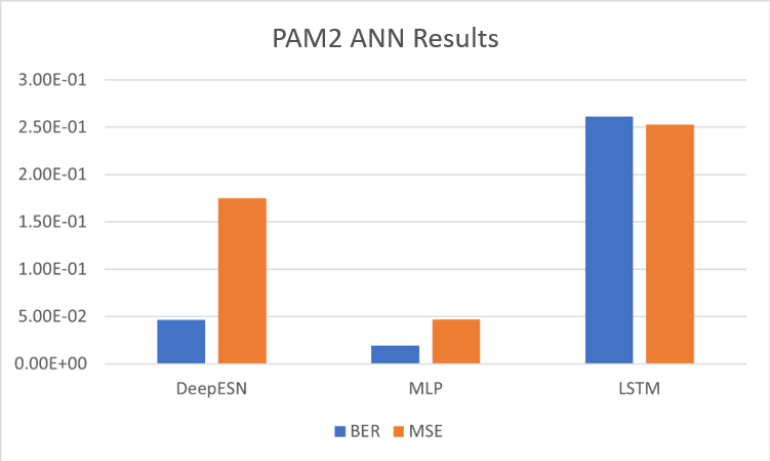


Figure 5.1: PAM2-600Mbps network Results

5.1 Multi-layer Perceptron

The Multi-layer perceptron when trained properly with the application of sliding windows was found to be quite effective on this data-set. The best configuration found can be seen in Table 5.4 as well as the output signal in Figure 5.2

Hyperparameter	**Value**
Activation Function	SeLU
Input Nodes	8
Hidden Layers size	40
of Hidden Layers	4
Epochs	10
Training Optimiser	ADAM
Window Size	8
Training Dataset size	100k
Testing Dataset size	100k

Table 5.2: MLP Configuration

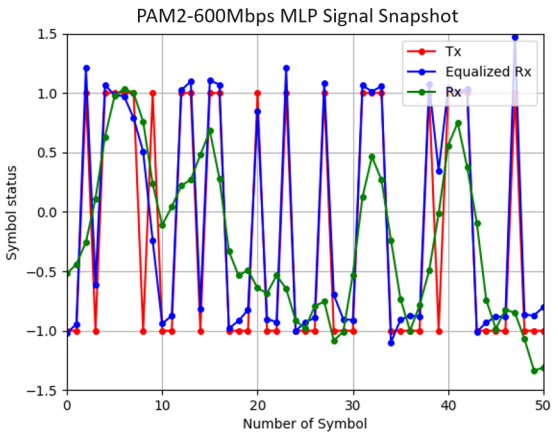


Figure 5.2: MLP Signal Snapshot

The MLP proved to have some interesting characteristics when performing the equalization, it was able to make the eye diagrams in some cases very dense indicating that when it was correct it was very close to perfect, however the eye would never be fully clear. This indicated that when the network was wrong, it was very wrong and it would misidentify symbols. This can be seen in Figure 5.2, in the first 10 symbols the 'Equalized Rx' misses several modulations.

This is not surprising because there are no recurrent nodes that would help with pattern recognition or prediction. Meaning that it was likely struggle when a pattern it has not seen before it put through to be equalized. The MLP also needed the most amount of data to train to be effective, needing $100k$ symbols to produce a good result as well as needing the sliding window.

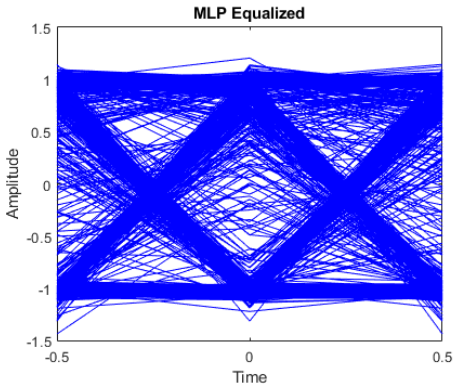
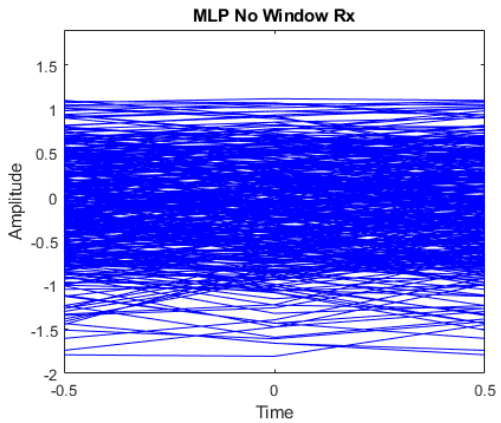


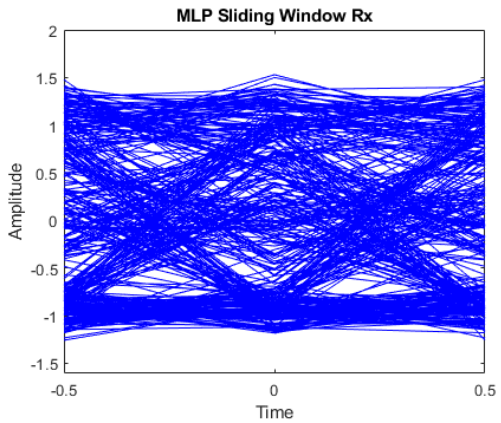
Figure 5.3: MLP Eye Diagram

5.1.1 Impact of Sliding Windows

When initially testing the capabilities of the MLP network a sliding window was not implemented, this as a consequence meant that the network only had a single input node and was unable to determine patterns. The overall performance was very poor, in some instances degrading the input signal even futher. Once a sliding window was implemented there was an instant performance increase without adjusting any other hyperparameters with the BER dropping from .312 to 8.648×10^{-2}. This can be seen in the eye diagrams out the output in Figure 5.4a and 5.4b where the signal structure is starting to form. It was found that a window size of 8 offered the best performance.



(a) Eye diagram without Sliding Window



(b) Eye diagram with Sliding Windows

5.2 Long Short Term Memory

The LSTM provided the most surprising result as it has been proven to be an effective method for equalization in [13] as well as being well documented as an excellent tool for time series prediction. However for this data-set the LSTM was not able to train to the problem very well. Overall the LSTM was only able to match the BER of the unequalized signal as can be seen in figure 5.5. This still showed that the LSTM was learning as otherwise its output would be close to random and degrade the BER.

Hyperparameter	Value
Activation Function	tanh
Recurrent Activation	sigmoid
LSTM Units	5
Epochs	15
Training Optimiser	ADAM
Training Dataset size	50k
Testing Dataset size	100k

Table 5.3: LSTM Configuration

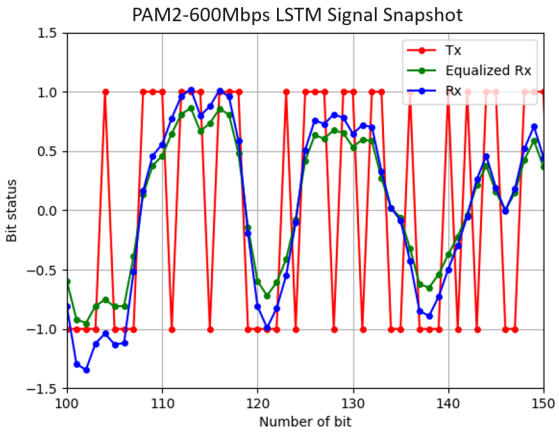


Figure 5.5: LSTM Signal Snapshot

5.3 Deep Echo State Network

The DeepESN proved to be able to produce good results on this data-set being able to achieve a BER of 4.655×10^{-2}. The training times were in general very fast when compared to MLP and LSTM only getting significantly longer when adding more layers. The DeepESN required little to no data pre-processing with only the normalization being applied. The BER of this network was close to that of the MLP with the difference only being on average 0.3.

Hyperparameter	Value
Units	50
Layer	5
Spectral Radius	0.6
Input Scaling	0.03
Leaking Rate	0.9
Training Dataset size	20k
Testing Dataset size	100k

Table 5.4: DeepESN Settings

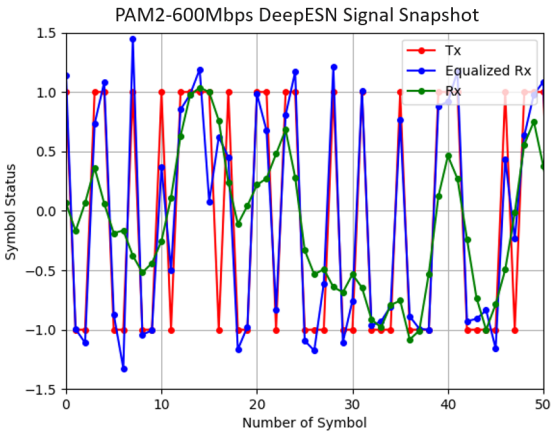


Figure 5.7: DeepESN Signal Snapshot

5.4 Initial Testing Conclusion

After preliminary testing none of the networks on this data-set was capable of matching the Decision Feedback Equalizer that had a BER of 1.334×10^{-3}. For further testing on higher constellation data-sets it was decided to continue with the DeepESN. The DeepESN was chosen because it had good performance when compared with the other networks being tested as well as the fact that is has been far less tested for this application and has shown to be adept in time series prediction and manipulation tasks [52].

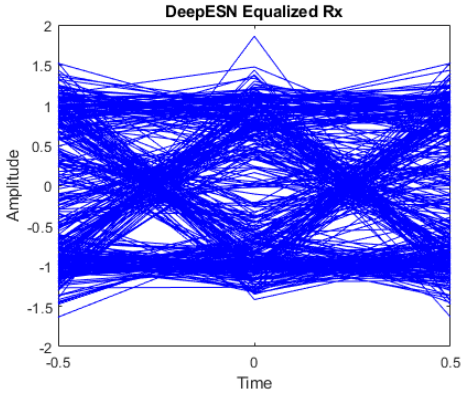


Figure 5.6: DeepESN PAM2-600Mbps Eye Diagram

While the MLP was able to produce a smaller BER and MSE on this data in testing. To achieve this the MLP required more data pre-processing needing to create the sliding window as well as normalization. The MLP also needed 5 times more training data to achieve its best result as well as taking substantially longer to train than the DeepESN.

Chapter 6

Further DeepESN Results

Dataset	Raw BER	DeepESN BER	DFE BER
PAM2-600Mbps	0.2609	4.655×10^{-2}	2.296×10^{-3}
PAM4-125Mbps	5.668×10^{-3}	6.591×10^{-5}	2.037×10^{-4}
PAM8-94Mbps*	5.848×10^{-4}	2.795×10^{-4}	1.169×10^{-4}
PAM8-188Mbps	5.171×10^{-2}	4.517×10^{-3}	5.134×10^{-3}
PAM16-125Mbps*	1.927×10^{-2}	1.303×10^{-2}	4.791×10^{-3}
PAM16-250Mbps	1.446×10^{-1}	4.262×10^{-2}	4.842×10^{-2}

Table 6.1: Performance of DeepESN on Each Data-set

For the further testing of the DeepESN the higher constellation data-sets were introduced. Overall the results for the DeepESN over all the tested data-sets proved very promising when compared with the standard DFE error rates and times on the same test data. The trained DeepESN was able to outperform the DFE on half of the data-sets and able to improve the signal in all cases while on average producing the result twice as fast. Table 6.1 shows the performance of the trained network on each signal format and speed tested. For all the results in Table 6.1

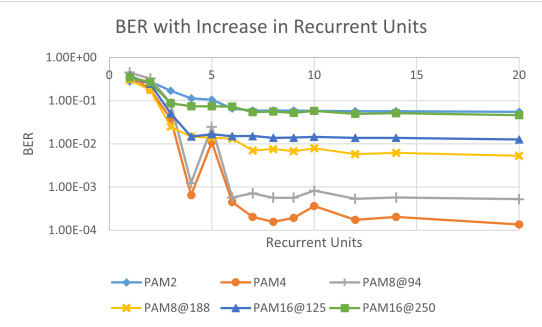
Data-set	DFE	DeepESN
PAM2-600Mbps	2.15s	0.59s
PAM4-125Mbps	1.37s	0.075s
PAM8-94Mbps	1.124s	0.19s
PAM8-188Mbps	2.45s	0.87s
PAM16-125Mbps	1.1s	0.51s
PAM16-250Mbps	2.152	1.15

Table 6.2: Time to Equalize test set in seconds

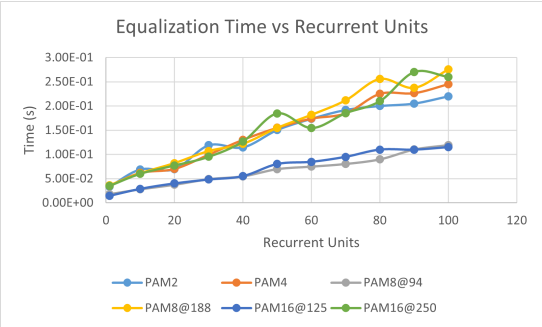
the network was trained on 20k symbols and tested on either 220k or 103k bits depending on the total size of the the data-set. Data-sets trained on 103k are marked with an asterisk. The full details of the network configurations that achieved these result as well as the DFE settings can be seen in Appendix A.

6.1 Effects of Hyperparameters

Recurrent Units



(a) BER vs Number of Recurrent Units



(b) Equalization Time vs Recurrent Units

The recurrent units describe the number of units inside of the reservoirs, this makes them a critical and one of the most influential hyperparameters. Figure 6.1a shows the BER performance when only increasing the number of units on a single layer. A dramatic improvement on as little as 8 units on all the data-sets was seen. However after this point improvements were still made they were just not as dramatic as within the first few units.

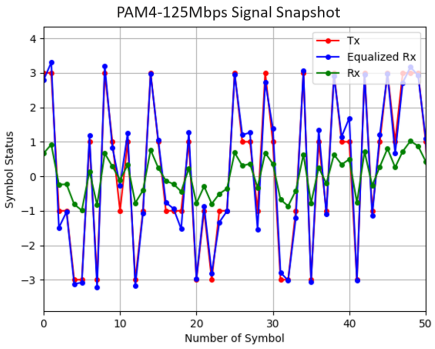
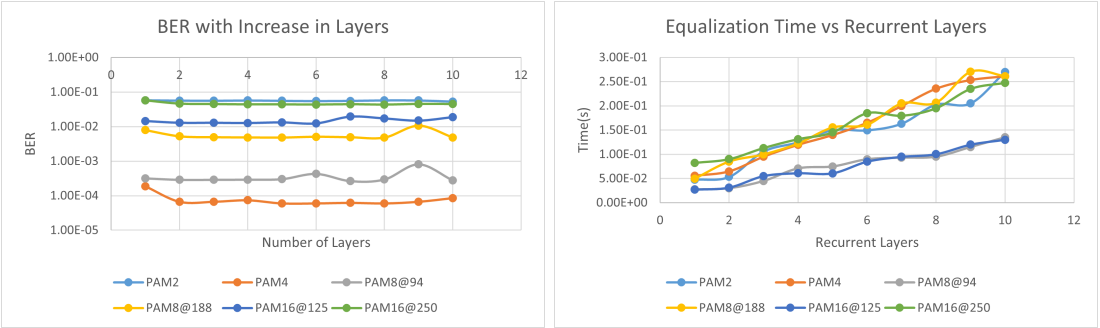


Figure 6.2: PAM4-125Mbps Output

It was observed that on the more complex signals such as PAM16 or signals that had a high unequalized error rate benefited more from using more units before reaching a plateau. In most cases the majority of the equalization could be achieved by just increasing the number of units on a single layer.

On all data-sets when increasing the number of units there was a linear increase in the time taken to equalize the signal on the test set. This was expected as when the number of units increases this increases the size of the reservoir matrix meaning more operations must take place. This also means the more data being equalized the longer it will take. In Figure 6.1b it is clear that the test sets of size 103k take less time than those of 240k. However the time taken was still faster than the standard DFE.

Recurrent Layers



(a) BER vs Number of Recurrent Layer (b) Training Time vs Recurrent Layers

The number of layers is the next most influential hyperparameter, defining how many reservoirs are in the network making it deep. During testing of this hyperparameter the number of recurrent units was set to 10, this would show the benefits of increasing the the number of layers on an already capable single layer network.

It was found that the greatest increase in performance was when increasing from 1 layer to 2. Increasing further there were minor improvements and deviations, however no improvements were large enough to give a significant change, this is largely reflected in Figure 6.3a. However it was found that when increasing the number of units and layers together a better performance could be found.

A downside to increasing the number of layers was the increase in training time. Testing showed a linear relationship between the number of layers and the time taken to train the network. While this was not unexpected this should be taken

into consideration. Much like the recurrent units this linear relationship also applies to the equalization times as shown in Figure 6.3b. This was due to the fact that the increase in layers is essentially a multiplier for the number of recurrent units as it creates more reservoir matrices leading to more operations being computed as the total number of units goes up.

Spectral Radius

When testing the effects of the spectral radius the network behaved as expected. As described in the model description once the radius was raised above 1 the network was unable to maintain its ESP and performance degraded, even to the point of the output being worse than the input. However while the radius was less than 1, it was seen that on average if the value was lower than 0.4 then an optimal solution was not be found, giving a lopsided bathtub graph as seen in the PAM4-125Mbps signal equalization in Figure 6.5b. Adjusting the spectral radius had no effect on training or testing times. On average a Spectral radius of 0.9 gave a good result on all data-sets. Full tables with test results can be seen in appendix B.

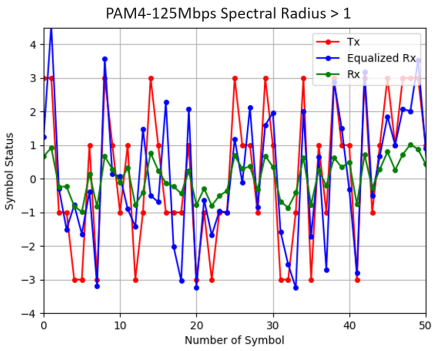
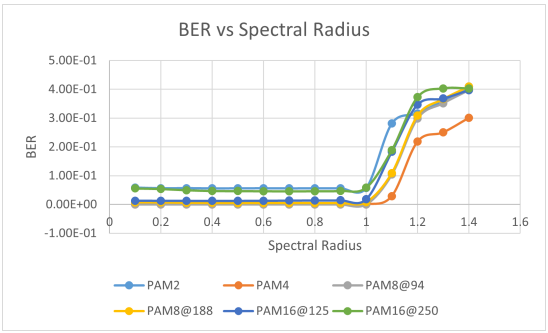
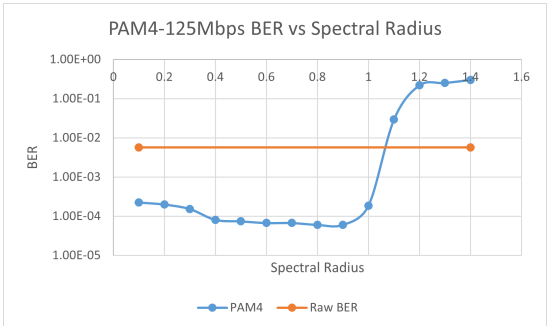


Figure 6.4: PAM4-125Mbps output with Spectral Radius above 1



(a) BER vs Reservoir Spectral Radius

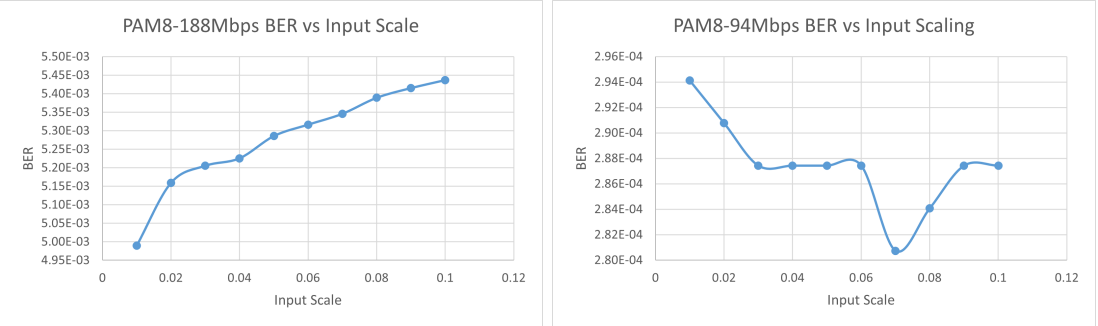


(b) PAM4-125Mbps BER vs Spectral Results

Input Scaling

When testing the effects of the input scale there was only small differences made to the effectiveness of the DeepESN, this was expected as the Input Scale is typically used for fine tuning the result. Changes were more pronounced on data-sets that had a lower unequalized BERs such as the PAM4-125Mbps signal. There were two noticeable patterns in the results. On the data-sets with a faster bit-rate such as the PAM16-250Mbps, PAM8-188Mbps and PAM2-600Mbps there was a trend that as the Input Scale became larger their BER increased.

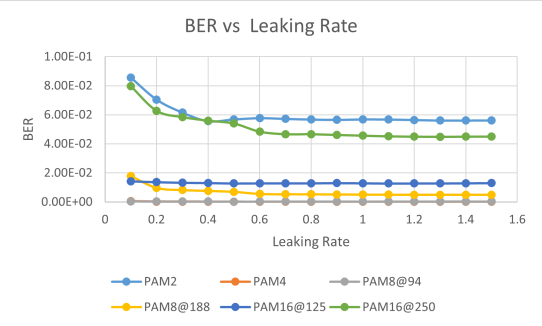
This is shown in Figure 6.7b with PAM8-188Mbps as an example. The other trend was a V style graph on the data-sets with a lower bit-rate where on the lower scales the BER started high and reduced. Then after a threshold the BER started to grow again. Figure 6.7b shows PAM16-125Mbps as an example. The rest of the results can be seen in Appendix B.



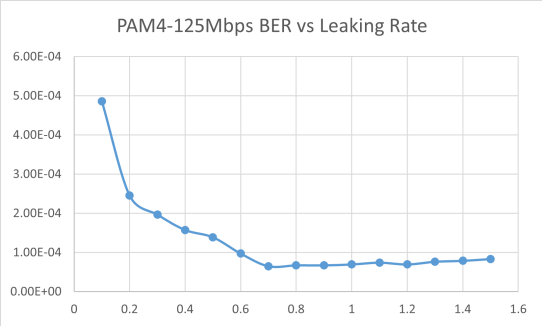
(a) BER vs Input Scale on PAM8-188Mbps (b) BER vs Input Scale on PAM8-94Mbps

Leaking Rate

During testing the effects of the Leaking Rate was fairly predictable across all data-sets, much like the Spectral radius and Input Scale the Leaking Rate only had a smaller effect on the outcome when compared to the number of units or layers in the network. The general trend while increasing the Leaking Rate was for the BER to start high and gradually lower until the rate was approximately 0.6 to 0.8, after which the BER would plateau.



(a) BER vs Leaking Rate



(b) PAM4 vs Leaking Rate

6.2 Best General Configuration for Training

Although each data-set had specific network configuration to get the lowest Error Rates. The settings used for the PAM4-125Mbps signal appeared as a good general configuration for training all of the data-sets. It was found that using this configuration the network was able to still improve the signal and give acceptable results on all data-sets while also training relatively quickly. With this general configuration the majority equalization can be achieved. The differences between the results can be seen in Table 6.4. This general configuration reflects what was discovered in the testing of hyperparameters as with only 10 units and 2 layers the DeepESN was still able to perform well.

Parameter	Value
Recurrent Units	10
Recurrent Layers	2
Spectral Radius	0.9
Input Scale	0.02
Leaking Rate	0.9
Training Symbols	20000

Table 6.3: General DeepESN Training Configuration

Dataset	Optimal	General	Difference
PAM2-600Mbps	4.655×10^{-2}	5.652×10^{-2}	9.97×10^{-3}
PAM4-125Mbps	6.019×10^{-5}	6.019×10^{-5}	0
PAM8-94Mbps*	2.795×10^{-4}	2.907×10^{-4}	1.12×10^{-5}
PAM8-188Mbps	4.517×10^{-3}	5.148×10^{-3}	6.31×10^{-4}
PAM16-125Mbps*	1.303×10^{-2}	1.428×10^{-2}	1.35×10^{-3}
PAM16-250Mbps	4.262×10^{-2}	4.682×10^{-2}	4.2×10^{-3}

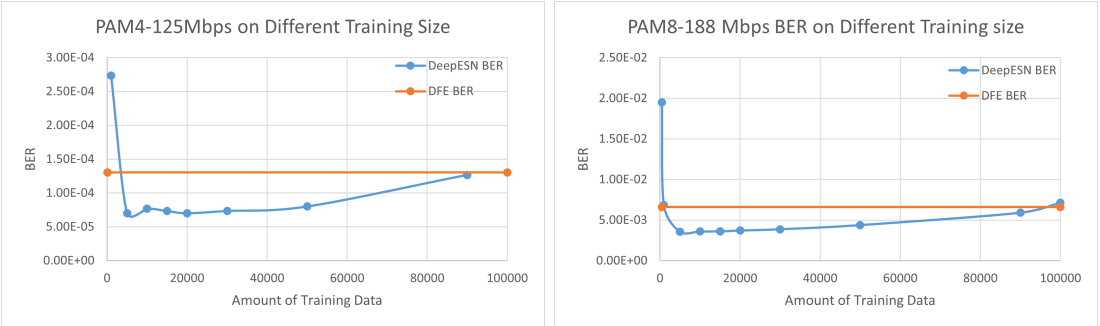
Table 6.4: General Configuration BER Performance of DeepESN

6.3 Amount of Training Data Required

The amount of data required to get an accurate result is quite crucial if this network were to be implemented on an embedded system and placed in a channel. If too little data is provided then the DeepESN will not be able to learn the patterns and do a poor job. If too much data is used then computing time is wasted and the network has the potential to get too familiar with the specific data in the training set and perform worst on the testing set as it is unpredictable [14].

For testing this feature the training set was increased from 100 symbols to 90k symbols incrementally while the testing set was was left unchanged at a size of 100k symbols.

It was found that in all of the data-sets the BER and MSE would reduce very rapidly when training on 100 to 1k symbols, then until 20k the performance reached a plateau after which the performance would start the degrade and the BER would rise. This was seen very clearly in the PAM4-125 Mbps and PAM8-188 Mbps data-sets where in Figure 6.8a and 6.8b shows they both surpassed the performance of the DFE Equalizer (in orange) but with only an increase in the amount of training data the error rate also increased. This behavior was seen in all of the data-sets however it was not as apparent in every one e.g PAM16-250Mbps BER only degraded by approximately 0.008.



(a) Ideal PAM4 Eye diagram

(b) High BER PAM4 Eye Diagram

Minor mitigation for this behavior was found by increasing the size of the network, however only approximately 10% could be recovered at the cost of training time.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

In conclusion the application of artificial neural networks to the problem of Equalizing underwater optical communication was quite successful in this project. The DeepESN showed great promise being able to improve the signal quality significantly in all cases and outperform the DFE on half of the data-sets it was trained and tested on. The Multi-Layer Perceptron network also showed great promise on the PAM2-600Mbps signal with major drawbacks in training time, data pre-processing and required training data, with only the LSTM failing to significantly increase the signal quality.

In further testing the DeepESN also proved once trained, that data could be passed through it very quickly giving a very rapid equalization with times on average being less than half of the DFE. Further Testing also revealed that DeepESN also did not need to be very large to be effective on any of the data-sets showing good results on networks as small as 10 units and 2 layers, which also benefited the time to equalize the signal. While more accurate results could be achieved with larger networks and more optimised networks for each signal, that came at the price of training time with diminishing returns.

A key takeaway from the testing of the DeepESN was the effects on the amount of training data required and its effect on the test results. Unlike networks such as the MLP where the more data it is given during training the more accurate it becomes. The DeepESN showed behaviour where especially in data-sets that had a low initial BER, when too much data was used for training, often greater than 20k symbols the network would perform worse then on less data.

While the MSE gave a good indication of the overall accuracy of an outputted signal, it was found that when the networks started to become more accurate and the closer the BER came to zero the MSE became less meaningful as a lower MSE would not always indicate a lower BER.

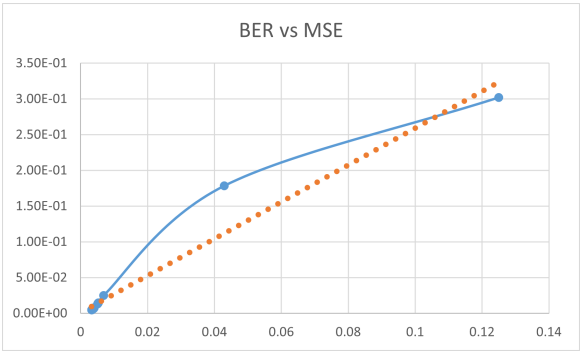


Figure 7.1: BER vs MSE

This was the case across all data-sets however it was more pronounced on the higher constellations such as PAM16. As you can see in Figure 7.1 when the MSE and BER are graphed together during the lower accuracy they match well, however the closer to zero they become its curves downwards and becomes less meaningful with the orange dotted line representing what the line would look like if the MSE and BER were perfectly correlated. Moving forward the use of another metric such was Root Mean Squared Error (RMSE) [53] or Mean Absolute Percentage Error (MAPE) [54] could be explored.

7.2 Future Work

Future work for this project has lots of potential, such as developing a purpose built Deep Echo state network program implementation to move away from using

Libraries, developing an embedded version of this style of equalizer that could be placed in a channel and as well as potentially integrate the decoder into the DeepESN.

7.2.1 Software

There is the potential to develop application specific software for the DeepESN implementation to move away from or adapt the use of the general purpose DeepESN Library. This would provide more flexibility while testing as well as improve data handling by the software as the different types of signals be all that would be passed though. This would also have the potential to help with an embedded implementation which would also needed to written from scratch.

7.2.2 Integrated Equalizer and Decoder

As future work there there is the option of attempt to train the network to not only equalize the signal but also decode the information inside being able to directly produce what was being sent. This would have many advantages the most obvious being the fact that it would remove a process from the communication pipeline potentially again giving the possibility to speedup the connection as well as giving advantages to using less power on remote devices that run on batteries.

7.2.3 Embedded Implementation

The next step to this project could be implementing this system on a physical embedded system such as an FGPA with CPU that could be placed in a water channel. This would enable physical testing of the capabilities of the trained DeepESN in a real environment with live data. The accuracy and speed could be thoroughly tested to determine if the advantages translate to a real world scenario.

References

[1] Zhaoquan Zeng et al. "A survey of underwater optical wireless communications". In: *IEEE communications surveys & tutorials* 19.1 (2016), pp. 204–238.

[2] Nan Chi et al. "Challenges and prospects of machine learning in visible light communication". In: *Journal of Communications and Information Networks* 5.3 (2020), pp. 302–309.

[3] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[4] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[5] Luca Pedrelli. *DeepESN*. `https://github.com/lucapedrelli/DeepESN`. 2019.

[6] Hemani Kaushal and Georges Kaddoum. "Underwater optical wireless communication". In: *IEEE access* 4 (2016), pp. 1518–1547.

[7] Mohammad-Ali Khalighi et al. "Underwater wireless optical communication; recent advances and remaining challenges". In: *2014 16th International Conference on Transparent Optical Networks (ICTON)*. IEEE. 2014, pp. 1–4.

[8] George Kechriotis, Evangelos Zervas, and Elias S Manolakos. "Using recurrent neural networks for adaptive communication channel

equalization". In: *IEEE transactions on Neural Networks* 5.2 (1994), pp. 267–278.

[9] Behzad Razavi. "The decision-feedback equalizer [a circuit for all seasons]". In: *IEEE Solid-State Circuits Magazine* 9.4 (2017), pp. 13–132.

[10] Paul Anthony Haigh et al. "A 20-Mb/s VLC link with a polymer LED and a multilayer perceptron equalizer". In: *IEEE Photonics Technology Letters* 26.19 (2014), pp. 1975–1978.

[11] Taehwan Kim and Tülay Adali. "Fully complex multi-layer perceptron network for nonlinear signal processing". In: *Journal of VLSI signal processing systems for signal, image and video technology* 32.1 (2002), pp. 29–43.

[12] Yiheng Zhao, Peng Zou, and Nan Chi. "3.2 Gbps underwater visible light communication system utilizing dual-branch multi-layer perceptron based post-equalizer". In: *Optics Communications* 460 (2020), p. 125197.

[13] Zihao Wang et al. "Long Short-Term Memory Neuron Equalizer". In: *arXiv preprint arXiv:2010.14009* (2020).

[14] Xingyu Lu et al. "Memory-controlled deep LSTM neural network post-equalizer used in high-speed PAM VLC system". In: *Optics express* 27.5 (2019), pp. 7822–7833.

[15] Fukui Tian, Qingyi Zhou, and Chuanchuan Yang. "Gaussian mixture model-hidden Markov model based nonlinear equalizer for optical fiber transmission". In: *Optics express* 28.7 (2020), pp. 9728–9737.

[16] Marc Bauduin et al. "Equalization of the non-linear satellite communication channel with an echo state network". In: *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*. IEEE. 2015, pp. 1–5.

[17] Hai-Peng Ren et al. "Performance improvement of chaotic baseband wireless communication using echo state network". In: *IEEE Transactions on Communications* 68.10 (2020), pp. 6525–6536.

[18] Douglas Reynolds. "Gaussian Mixture Models". In: *Encyclopedia of Biometrics*. Ed. by Stan Z. Li and Anil Jain. Boston, MA: Springer US, 2009, pp. 659–663. ISBN: 978-0-387-73003-5. DOI: 10.1007/978-0-387-73003-5_196. URL: https://doi.org/10.1007/978-0-387-73003-5_196.

[19] Lawrence Rabiner and Biinghwang Juang. "An introduction to hidden Markov models". In: *ieee assp magazine* 3.1 (1986), pp. 4–16.

[20] Elena Rodrıguez et al. "Speech/speaker recognition using a HMM/GMM hybrid model". In: *International Conference on Audio-and Video-Based Biometric Person Authentication*. Springer. 1997, pp. 227–234.

[21] Chayan Kathuria. https://towardsdatascience.com/https-medium-com-chayankathuria-regression-why-mean-square-error-a8cad2a1c96f. 2019.

[22] Gary Breed. "Bit error rate: Fundamental concepts and measurement issues". In: *High Frequency Electronics* 2.1 (2003), pp. 46–47.

[23] Gary Breed. "Analyzing signals using the eye diagram". In: *High Frequency Electronics* 4.11 (2005), pp. 50–53.

[24] Sotiris B Kotsiantis, Dimitris Kanellopoulos, and Panagiotis E Pintelas. "Data preprocessing for supervised leaning". In: *International Journal of Computer Science* 1.2 (2006), pp. 111–117.

[25] Aniruddha Bhandari. *Feature Scaling for Machine Learning: Understanding the Difference Between Normalization vs. Standardization*. https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/. 2020.

[27] Anita Rácz, Dávid Bajusz, and Károly Héberger. "Effect of Dataset Size and Train/Test Split Ratios in QSAR/QSPR Multiclass Classification". In: *Molecules* 26.4 (2021), p. 1111.

[28] Thomas G Dietterich. "Machine learning for sequential data: A review". In: *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*. Springer. 2002, pp. 15–30.

[29] Kavita Burse, Ram Narayan Yadav, and SC Shrivastava. "Channel equalization using neural networks: A review". In: *IEEE transactions on systems, man, and cybernetics, Part C (Applications and Reviews)* 40.3 (2010), pp. 352–357.

[31] Huanling Hu, Lin Wang, and Sheng-Xiang Lv. "Forecasting energy consumption and wind power generation using deep echo state network". In: *Renewable Energy* 154 (2020), pp. 598–613.

[32] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. "Design of deep echo state networks". In: *Neural Networks* 108 (2018), pp. 33–47.

[33] VA Golovko. "Deep learning: an overview and main paradigms". In: *Optical memory and neural networks* 26.1 (2017), pp. 1–17.

[34] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[35] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[37] Jason Brownlee. *Crash Course On Multi-Layer Perceptron Neural Networks*. `https://machinelearningmastery.com/neural-networks-crash-course/`. 2020.

[38] Sagar Sharma. "Activation functions in neural networks". In: *towards data science* 6 (2017).

[39] Günter Klambauer et al. "Self-normalizing neural networks". In: *arXiv preprint arXiv:1706.02515* (2017).

[40] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[41] Christopher Olah. *Understanding LSTM Networks*. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. 2015.

[42] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[43] Jason Brownlee. *How to use Data Scaling Improve Deep Learning Model Stability and Performance*. https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/. 2019.

[44] Van Huyen. *Back propagation in Long Short Term Memory (LSTM)*. https://medium.com/@dovanhuyen2018/back-propagation-in-long-short-term-memory-lstm-a13ad8ae7a57. 2018.

[45] Mantas Lukoševičius. "A practical guide to applying echo state networks". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 659–686.

[46] Herbert Jaeger. "Echo state network". In: *scholarpedia* 2.9 (2007), p. 2330.

[47] Herbert Jaeger et al. "Optimization and applications of echo state networks with leaky-integrator neurons". In: *Neural networks* 20.3 (2007), pp. 335–352.

[48] Mantas Lukoševičius and Herbert Jaeger. "Reservoir computing approaches to recurrent neural network training". In: *Computer Science Review* 3.3 (2009), pp. 127–149.

[49] Claudio Gallicchio and Alessio Micheli. "Architectural and markovian factors of echo state networks". In: *Neural Networks* 24.5 (2011), pp. 440–456.

[50] Herbert Jaeger and Harald Haas. "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication". In: *science* 304.5667 (2004), pp. 78–80.

[51] Izzet B Yildiz, Herbert Jaeger, and Stefan J Kiebel. "Re-visiting the echo state property". In: *Neural networks* 35 (2012), pp. 1–9.

[52] Claudio Gallicchio and Alessio Micheli. "Deep echo state network (deepesn): A brief survey". In: *arXiv preprint arXiv:1712.04323* (2017).

[53] Susan Holmes. *RMS Error*. `https://statweb.stanford.edu/~susan/courses/s60/split/node60.html`. 2000.

[54] "MAPE (mean absolute percentage error)MEAN ABSOLUTE PERCENTAGE ERROR (MAPE)". In: *Encyclopedia of Production and Manufacturing Management*. Ed. by P. M. Swamidass. Boston, MA: Springer US, 2000, pp. 462–462. ISBN: 978-1-4020-0612-8. DOI: 10.1007/1-4020-0612-8_580. URL: https://doi.org/10.1007/1-4020-0612-8_580.

Bibliography

[55] Pau Vilimelis Aceituno, Gang Yan, and Yang-Yu Liu. "Tailoring Echo State Networks for Optimal Learning". In: *Iscience* 23.9 (2020), p. 101440.

[56] Chieh-Fang Teng, Han-Mo Ou, and An-Yeu Wu. "Neural network-based equalizer by utilizing coding gain in advance". In: *arXiv preprint arXiv:1907.04980* (2019).

[57] Haiquan Zhao and Jiashu Zhang. "Adaptively combined FIR and functional link artificial neural network equalizer for nonlinear communication channel". In: *IEEE Transactions on Neural Networks* 20.4 (2009), pp. 665–674.

[58] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. "Deep reservoir computing: A critical experimental analysis". In: *Neurocomputing* 268 (2017), pp. 87–99.

Appendix A

Equalizer Configurations

A.1 Decision Feedback Equalizer

Parameter	Value
Forward Taps	16
Tap Delay	8
Feedback Taps	8
Training Symbols	4000

Table A.1: Decision feedback Equalizer Configuration

A.2 DeepESN for 2PAM at 600Mbps

Parameter	Value
Recurrent Units	50
Recurrent Layers	5
Spectral Radius	0.7
Input Scale	0.03
Leaking Rate	0.4
Training Symbols	20000

Table A.2: DeepESN Configuration for 2PAM Data-set

A.3 DeepESN for 4PAM at 125Mbps

Parameter	Value
Recurrent Units	10
Recurrent Layers	2
Spectral Radius	0.9
Input Scale	0.02
Leaking Rate	0.9
Training Symbols	20000

Table A.3: DeepESN Configuration for 4PAM Data-set

A.4 DeepESN for 8PAM at 94Mbps

Parameter	Value
Recurrent Units	40
Recurrent Layers	4
Spectral Radius	0.5
Input Scale	0.07
Leaking Rate	0.5
Training Symbols	20000

Table A.4: DeepESN Configuration for 8PAM at 94Mbps Data-set

A.5 DeepESN for 8PAM at 188Mbps

Parameter	Value
Recurrent Units	70
Recurrent Layers	5
Spectral Radius	0.7
Input Scale	0.01
Leaking Rate	1.2
Training Symbols	20000

Table A.5: DeepESN Configuration for 8PAM at 188Mbps Data-set

A.6 DeepESN for 16PAM at 125Mbps

Parameter	Value
Recurrent Units	100
Recurrent Layers	5
Spectral Radius	0.5
Input Scale	0.02
Leaking Rate	0.5
Training Symbols	20000

Table A.6: DeepESN Configuration for 16PAM at 125Mbps Data-set

A.7 DeepESN for 16PAM at 250Mbps

Parameter	Value
Recurrent Units	100
Recurrent Layers	5
Spectral Radius	0.7
Input Scale	0.01
Leaking Rate	1.3
Training Symbols	20000

Table A.7: DeepESN Configuration for 16PAM at 250Mbps Data-set

Appendix B

DeepESN Testing Results

Changes with BER with respect to changing hyperparameters

B.1 Recurrent Units

Units	PAM2	PAM4	PAM8@94	PAM8@188	PAM16@125	PAM16@250
1	2.79E-01	3.97E-01	4.47E-01	3.02E-01	3.74E-01	3.46E-01
2	2.79E-01	1.82E-01	3.26E-01	1.78E-01	2.46E-01	2.69E-01
3	1.70E-01	3.81E-02	8.64E-02	2.50E-02	5.07E-02	8.66E-02
4	1.14E-01	6.49E-04	1.23E-03	1.46E-02	1.49E-02	7.45E-02
5	1.06E-01	1.10E-02	2.49E-02	1.39E-02	1.66E-02	7.40E-02
6	6.58E-02	4.50E-04	5.71E-04	1.32E-02	1.52E-02	7.30E-02
7	5.94E-02	2.05E-04	7.22E-04	6.96E-03	1.52E-02	5.41E-02
8	5.95E-02	1.56E-04	5.66E-04	7.59E-03	1.37E-02	5.63E-02
9	5.92E-02	1.94E-04	5.66E-04	6.72E-03	1.39E-02	5.26E-02
10	5.79E-02	3.66E-04	8.30E-04	7.94E-03	1.46E-02	5.81E-02
12	5.72E-02	1.75E-04	5.33E-04	5.80E-03	1.37E-02	4.97E-02
14	5.68E-02	2.05E-04	5.71E-04	6.24E-03	1.37E-02	5.10E-02
20	5.55E-02	1.37E-04	5.23E-04	5.25E-03	1.27E-02	4.67E-02
30	5.62E-02	1.08E-04	5.17E-04	5.25E-03	1.26E-02	4.62E-02
40	5.37E-02	1.10E-04	5.44E-04	4.99E-03	1.24E-02	4.41E-02
50	5.49E-02	9.16E-05	4.74E-04	4.79E-03	1.24E-02	4.37E-02
100	4.97E-02	9.16E-05	4.7E-04	4.71E-03	1.24E-02	4.33E-02

B.2 Recurrent Layers

Layers	PAM2	PAM4	PAM8@94	PAM8@188	PAM16@125	PAM16@250
1	5.77E-02	1.88E-04	3.18E-04	7.94E-03	1.46E-02	5.81E-02
2	5.65E-02	6.71E-05	2.87E-04	5.21E-03	1.30E-02	4.61E-02
3	5.60E-02	6.71E-05	2.87E-04	4.89E-03	1.30E-02	4.48E-02
4	5.70E-02	7.41E-05	2.91E-04	4.82E-03	1.28E-02	4.38E-02
5	5.58E-02	6.02E-05	3.01E-04	4.77E-03	1.33E-02	4.38E-02
6	5.49E-02	6.02E-05	4.28E-04	5.05E-03	1.25E-02	4.32E-02
7	5.55E-02	6.25E-05	2.64E-04	4.88E-03	1.95E-02	4.42E-02
8	5.74E-02	6.02E-05	2.94E-04	4.77E-03	1.73E-02	4.31E-02
9	5.69E-02	6.71E-05	8.09E-04	1.07E-02	1.49E-02	4.51E-02
10	5.32E-02	8.57E-05	2.77E-04	4.84E-03	1.88E-02	4.49E-02

B.3 Spectral Radius

Radius	PAM2	PAM4	PAM8@94	PAM8@188	PAM16@125	PAM16@250
0.1	5.87E-02	2.25E-04	3.01E-04	7.47E-03	1.32E-02	5.56E-02
0.2	5.64E-02	1.99E-04	2.57E-04	7.26E-03	1.30E-02	5.38E-02
0.3	5.67E-02	1.53E-04	2.71E-04	5.90E-03	1.28E-02	4.96E-02
0.4	5.58E-02	8.10E-05	2.74E-04	5.33E-03	1.28E-02	4.69E-02
0.5	5.61E-02	7.41E-05	2.67E-04	5.27E-03	1.27E-02	4.65E-02
0.6	5.63E-02	6.71E-05	2.87E-04	5.21E-03	1.30E-02	4.61E-02
0.7	5.59E-02	6.71E-05	3.01E-04	5.07E-03	1.37E-02	4.57E-02
0.8	5.61E-02	6.02E-05	3.21E-04	5.07E-03	1.42E-02	4.61E-02
0.9	5.60E-02	6.02E-05	2.97E-04	5.20E-03	1.44E-02	4.72E-02
1	5.69E-02	1.85E-04	5.65E-04	8.38E-03	1.82E-02	5.84E-02
1.1	2.81E-01	2.91E-02	1.04E-01	1.09E-01	1.83E-01	1.89E-01
1.2	3.16E-01	2.18E-01	2.98E-01	3.09E-01	3.47E-01	3.73E-01
1.3	3.60E-01	2.51E-01	3.52E-01	3.66E-01	3.68E-01	4.03E-01
1.4	4.09E-01	3.01E-01	3.97E-01	4.10E-01	3.97E-01	4.03E-01

B.4 Leaking Rate

Rate	PAM2	PAM4	PAM8@94	PAM8@188	PAM16@125	PAM16@250
0.1	8.55E-02	4.86E-04	3.28E-04	1.78E-02	1.43E-02	7.96E-02
0.2	7.04E-02	2.45E-04	2.78E-04	9.69E-03	1.37E-02	6.27E-02
0.3	6.15E-02	1.97E-04	2.71E-04	8.24E-03	1.33E-02	5.84E-02
0.4	5.56E-02	1.57E-04	2.71E-04	7.64E-03	1.31E-02	5.59E-02
0.5	5.68E-02	1.39E-04	2.45E-04	7.00E-03	1.28E-02	5.40E-02
0.6	5.77E-02	9.72E-05	2.75E-04	5.57E-03	1.29E-02	4.84E-02
0.7	5.72E-02	6.48E-05	2.78E-04	5.34E-03	1.28E-02	4.67E-02
0.8	5.67E-02	6.71E-05	2.75E-04	5.31E-03	1.29E-02	4.66E-02
0.9	5.65E-02	6.71E-05	2.88E-04	5.21E-03	1.30E-02	4.61E-02
1	5.68E-02	6.94E-05	2.61E-04	5.11E-03	1.29E-02	4.56E-02
1.1	5.67E-02	7.41E-05	2.81E-04	5.04E-03	1.27E-02	4.52E-02
1.2	5.64E-02	6.94E-05	2.81E-04	4.98E-03	1.28E-02	4.49E-02
1.3	5.60E-02	7.64E-05	2.88E-04	4.98E-03	1.28E-02	4.48E-02
1.4	5.61E-02	7.87E-05	2.88E-04	4.98E-03	1.29E-02	4.49E-02
1.5	5.61E-02	8.33E-05	2.81E-04	5.01E-03	1.30E-02	4.50E-02

Scale	PAM2	PAM4	PAM8@94	PAM8@188	PAM16@125	PAM16@250
0.004	5.66E-02	6.48E-05	3.14E-04	4.86E-03	1.37E-02	4.43E-02
0.005	5.65E-02	6.71E-05	3.04E-04	4.87E-03	1.36E-02	4.44E-02
0.006	5.65E-02	6.71E-05	2.91E-04	4.91E-03	1.36E-02	4.44E-02
0.007	5.66E-02	6.94E-05	2.91E-04	4.90E-03	1.35E-02	4.45E-02
0.008	5.66E-02	6.71E-05	2.97E-04	4.93E-03	1.33E-02	4.46E-02
0.009	5.66E-02	6.71E-05	2.91E-04	4.95E-03	1.33E-02	4.46E-02
0.01	5.66E-02	6.25E-05	2.94E-04	4.99E-03	1.32E-02	4.47E-02
0.02	5.65E-02	6.25E-05	2.91E-04	5.16E-03	1.29E-02	4.57E-02
0.03	5.65E-02	6.71E-05	2.87E-04	5.21E-03	1.30E-02	4.61E-02
0.04	5.66E-02	6.94E-05	2.87E-04	5.23E-03	1.30E-02	4.63E-02
0.05	5.66E-02	7.41E-05	2.87E-04	5.29E-03	1.31E-02	4.66E-02
0.06	5.66E-02	7.87E-05	2.87E-04	5.32E-03	1.31E-02	4.68E-02
0.07	5.66E-02	8.10E-05	2.81E-04	5.35E-03	1.31E-02	4.69E-02
0.08	5.68E-02	8.33E-05	2.84E-04	5.39E-03	1.31E-02	4.71E-02
0.09	5.69E-02	8.33E-05	2.87E-04	5.42E-03	1.31E-02	4.71E-02
0.1	5.69E-02	8.33E-05	2.87E-04	5.44E-03	1.31E-02	4.74E-02

B.5 Input Scaling

B.6 Amount of Training Data

Train Size	PAM2	PAM4	PAM8@94	PAM8@188	PAM16@125	PAM16@250
100	5.00E-01	7.64E-02	5.01E-01	9.69E-02	5.01E-01	5.01E-01
500	6.02E-02	1.42E-03	3.50E-03	1.95E-02	3.22E-02	3.66E-02
1000	5.88E-02	2.73E-04	4.86E-04	6.84E-03	1.76E-02	3.65E-02
5000	5.60E-02	7.00E-05	2.46E-04	3.58E-03	1.28E-02	3.81E-02
10000	5.56E-02	7.67E-05	2.25E-04	3.59E-03	1.24E-02	3.91E-02
15000	5.58E-02	7.33E-05	2.32E-04	3.60E-03	1.25E-02	3.96E-02
20000	5.58E-02	7.00E-05	1.81E-04	3.71E-03	1.26E-02	4.05E-02
30000	5.58E-02	7.33E-05	2.17E-04	3.87E-03	1.26E-02	4.21E-02
50000	5.58E-02	8.00E-05	2.32E-04	4.38E-03	1.33E-02	4.58E-02
90000	5.57E-02	1.27E-04		5.90E-03		5.41E-02

B.7 Equalization Time: Units

Time to Equalize in seconds with increasing recurrent units

Units	PAM2	PAM4	PAM8@94	PAM8@188	PAM16@125	PAM16@250
1	3.37E-02	3.51E-02	1.80E-02	3.60E-02	1.45E-02	3.51E-02
10	6.90E-02	6.19E-02	2.80E-02	6.09E-02	2.90E-02	6.03E-02
20	6.97E-02	7.02E-02	3.74E-02	8.19E-02	4.01E-02	7.74E-02
30	1.19E-01	9.98E-02	4.89E-02	1.07E-01	4.82E-02	9.52E-02
40	1.15E-01	1.30E-01	5.48E-02	1.22E-01	5.52E-02	1.27E-01
50	1.50E-01	1.55E-01	6.97E-02	1.55E-01	8.02E-02	1.85E-01
60	1.74E-01	1.74E-01	7.49E-02	1.82E-01	8.47E-02	1.55E-01
70	1.92E-01	1.85E-01	8.04E-02	2.12E-01	9.50E-02	1.85E-01
80	2.00E-01	2.25E-01	8.98E-02	2.56E-01	1.10E-01	2.10E-01
90	2.05E-01	2.27E-01	1.10E-01	2.37E-01	1.10E-01	2.70E-01
100	2.20E-01	2.45E-01	1.19E-01	2.76E-01	1.15E-01	2.60E-01

B.8 Equalization Time: Layers

Time to Equalize in seconds with increasing recurrent Layers

Layers	PAM2	PAM4	PAM8@94	PAM8@188	PAM16@125	PAM16@250
1	4.76E-02	5.58E-02	2.76E-02	4.96E-02	2.70E-02	8.24E-02
2	5.34E-02	6.48E-02	3.01E-02	8.51E-02	3.13E-02	9.01E-02
3	1.06E-01	9.52E-02	4.50E-02	9.98E-02	5.51E-02	1.13E-01
4	1.25E-01	1.20E-01	7.09E-02	1.22E-01	6.09E-02	1.31E-01
5	1.50E-01	1.40E-01	7.48E-02	1.55E-01	6.08E-02	1.45E-01
6	1.50E-01	1.65E-01	8.97E-02	1.61E-01	8.49E-02	1.85E-01
7	1.63E-01	2.00E-01	9.30E-02	2.05E-01	9.49E-02	1.80E-01
8	2.03E-01	2.36E-01	9.50E-02	2.07E-01	1.00E-01	1.95E-01
9	2.05E-01	2.54E-01	1.15E-01	2.70E-01	1.20E-01	2.35E-01
10	2.70E-01	2.61E-01	1.35E-01	2.61E-01	1.30E-01	2.47E-01

Appendix C

MLP Preliminary Results

epochs	BER
1	1.74E-01
2	1.28E-01
3	1.29E-01
4	1.04E-01
5	0.102893
10	1.06E-01
15	9.96E-02
20	9.60E-02
25	9.79E-02
30	9.75E-02
35	9.69E-02
40	9.07E-02
45	8.96E-02
50	1.00E-01
55	8.86E-02
60	8.60E-02
65	9.60E-02
70	8.65E-02
75	9.87E-02

Hidden Size	BER
10	4.00E-02
20	4.63E-02
30	3.17E-02
40	5.40E-02
50	4.43E-02
60	2.43E-02
70	4.60E-02
80	2.90E-02
90	3.30E-02
100	2.97E-02
110	2.80E-02
120	3.90E-02
130	2.40E-02
140	5.20E-02
150	4.20E-02
160	2.43E-02
170	2.27E-02
180	2.93E-02
190	5.02E-01

layers	BER
1	1.36E-01
2	7.57E-02
3	7.10E-02
4	3.70E-02
5	4.33E-02
6	4.54E-02
7	2.53E-02
8	2.30E-02
9	8.70E-02
10	5.84E-02
11	2.83E-02
12	9.14E-02
13	8.64E-02
14	1.05E-01
15	1.12E-01
16	1.08E-01
17	9.94E-02
18	2.37E-01
19	5.02E-01

# of bits (k)	BER
10	1.18E-01
20	9.15E-02
30	9.61E-02
40	5.62E-02
50	4.31E-02
60	8.62E-02
70	9.51E-02
80	2.92E-02
90	6.08E-02

Function	BER
tanh	9.00E-02
sigmoid	4.98E-01
relu	5.02E-01
elu	1.76E-01
selu	2.63E-02
softmax	4.98E-01
exponential	5.02E-01
softplus	1.25E-01
softsign	8.27E-02

Optimiser	BER
SGD	5.02E-01
RMSprop	7.33E-02
Adam	2.80E-02
Adadelta	1.69E-01
Adagrad	3.20E-01
Adamax	7.67E-02
Nadam	9.77E-02
Ftrl	4.09E-01

Size of windows	BER
10	5.57E-02
20	5.64E-02
30	1.43E-01
40	9.12E-02
50	1.02E-01
60	1.20E-01
70	1.08E-01

Appendix D

LSTM Preliminary Results

Units	BER
1	2.610E-01
2	2.615E-01
3	2.609E-01
4	2.610E-01
5	2.608E-01
6	2.613E-01
7	2.608E-01
8	2.613E-01
9	2.612E-01
10	2.614E-01
20	2.615E-01
30	2.610E-01
40	2.612E-01

Epochs	BER
1	2.610E-01
2	2.620E-01
3	2.613E-01
4	2.609E-01
5	2.613E-01
6	2.611E-01
7	2.612E-01
8	2.609E-01
9	2.608E-01
10	2.614E-01
20	2.615E-01
30	2.612E-01
40	2.612E-01

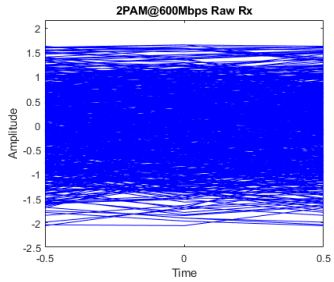
Training	BER
100	5.005E-01
500	5.005E-01
1000	5.005E-01
5000	3.216E-01
10000	2.630E-01
15000	2.736E-01
20000	2.627E-01
30000	2.607E-01
40000	2.609E-01
50000	2.608E-01
60000	2.610E-01
70000	2.612E-01
80000	2.606E-01
90000	2.607E-01

Optimiser	BER
SGD	2.624E-01
RMSprop	2.613E-01
Adam	2.608E-01
Adadelta	4.998E-01
Adagrad	4.998E-01
Adamax	2.619E-01
Nadam	2.631E-01
Ftrl	4.998E-01

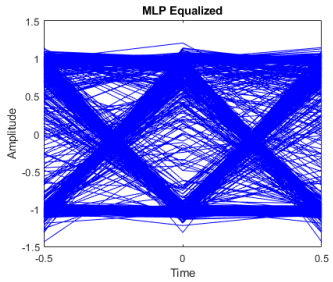
Appendix E

Eye Diagrams

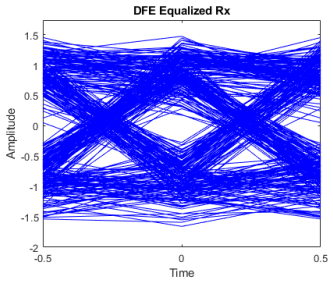
E.1 Multi-Layer Perceptron



(a) Raw

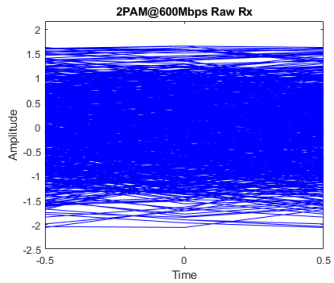


(b) MLP

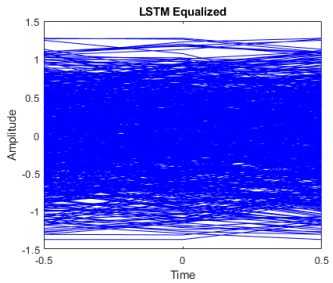


(c) DFE

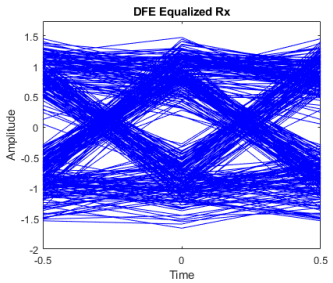
E.2 Long-Short Term Memory



(a) Raw



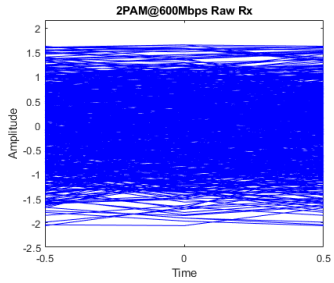
(b) LSTM



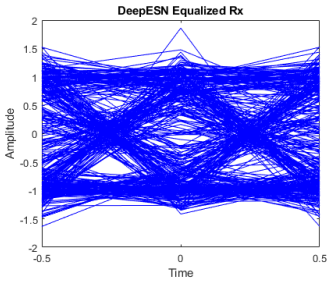
(c) DFE

E.3 Deep Echo-State Network

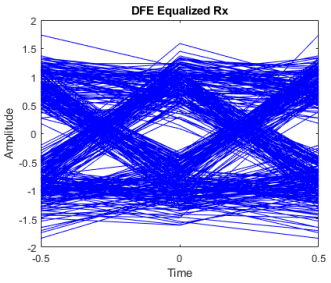
2PAM - 600Mbps



(a) Raw

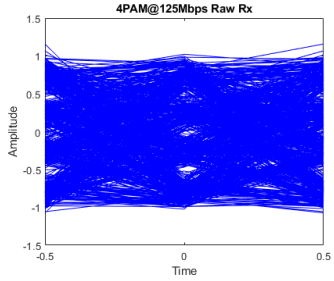


(b) DeepESN

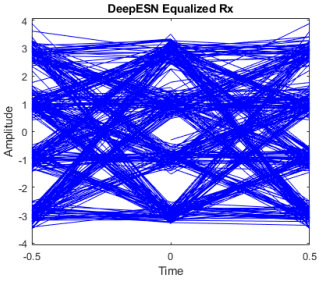


(c) DFE

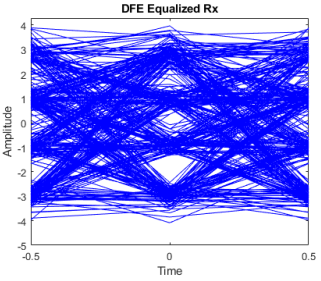
4PAM - 125Mbps



(a) Raw

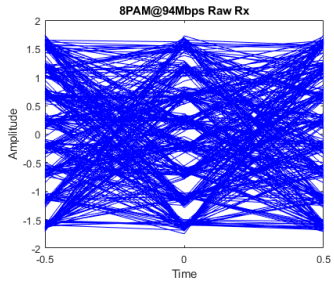


(b) DeepESN

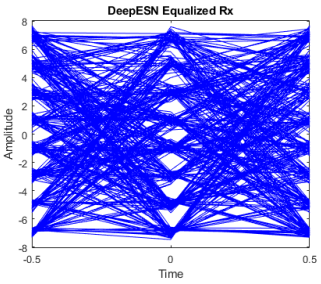


(c) DFE

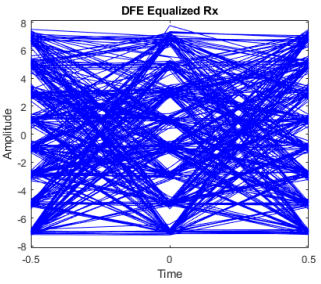
8PAM - 94Mbps



(a) Raw

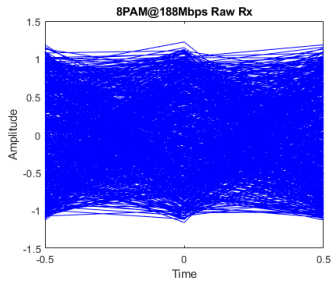


(b) DeepESN

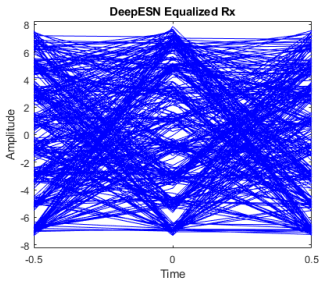


(c) DFE

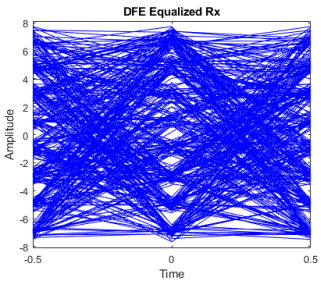
8PAM - 188Mbps



(a) Raw

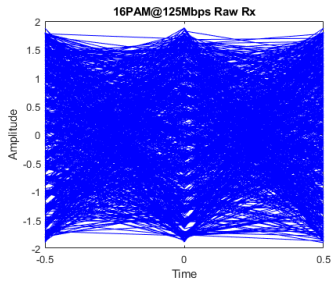


(b) DeepESN

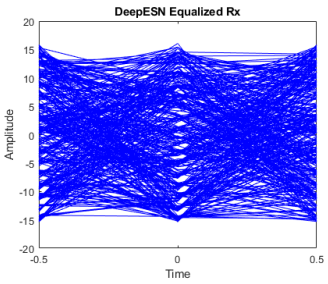


(c) DFE

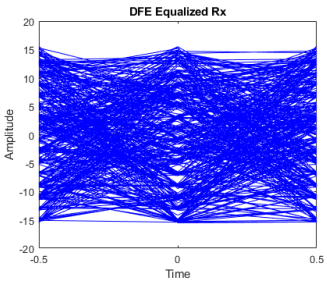
16PAM - 125Mbps



(a) Raw

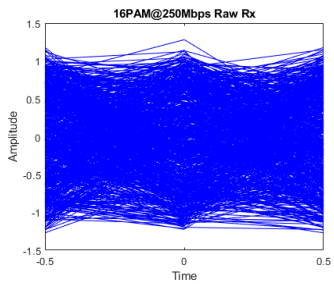


(b) DeepESN

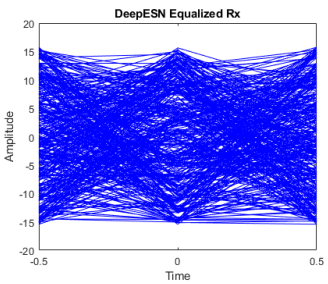


(c) DFE

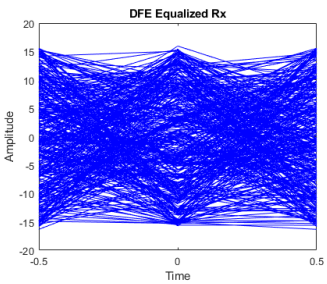
16PAM - 250Mbps



(a) Raw



(b) DeepESN



(c) DFE

Appendix F

Supervisor Meeting Minutes

All meetings took place on Microsoft Teams

F.0.1 Meeting 1 - 29/09/2020

Agenda

- Speak with Mauro for first time
- Discuss possible projects for the next year

Minutes

- Mauro suggests Machine Learning project for equalization
- Mauro explains the project as a whole and the problems that need solving with optical communication

F.0.2 Meeting 2 - 15/10/2020

Agenda

- Discuss with Mauro possible machine leaning Models

- Update on project progress
- Discuss Project Milestones

Minutes

- Milestones decided on
  - implement a few different types of ANN
  - determine most viable
  - investigate further

F.0.3 Meeting 3 - 02/11/2020

Agenda

- Project Update - approx a week behind
- Discuss possibility of using RNN as the equalizer
- Tell Mauro Plan of doing both a DeepESN and Tensor Flow implementation
- Ask about presentation - anything in slides not correct?

Minutes

- Mauro suggests looking at MATLAB code before implementing ML
- Discuss using examples to learn how to use DeepESN
- Mauro explains find best reservoir size for DeepESN can be done through trail and error
- Suggests trying 3 different approaches and continuing on with the most promising

F.0.4 Meeting 4 - 26/01/2021

Agenda

- Discuss computer Memory issue - not enough
- Talk about re-training networks for better performance
- Discuss potential way forward if current networks keep failing

Minutes

- Mauro offers to buy more RAM for PC
- Talk about possibility of finding relationship between BER and MSE
- Am invited to show most recent results at UWOC Meeting
- Mauro Suggests including more data in the models

F.0.5 Meeting 5 - 02/02/2021

Agenda

- Ask Mauro if he knows how to save DeepESN network to use later
- Ask if possibility of using matlab script to calculate BER and not in python

Minutes

- Mauro suggests to implement windowing on MLP before focusing on DeepESN
- Mauro explains the benefits of using windowing learning bits vs bytes

F.0.6 Meeting 6 - 23/02/2021

Agenda

- Show most recent results
- Tell Mauro Memory is on the way

Minutes

- Mauro says what he would like in the report
- Mauro explains a bit of the future work potentially with the DeepESNs

F.0.7 Meeting 7 - 22/03/2021

Agenda

- Update Mauro on results - DeepESN finally working
  - able to beat on 4PAM
  - unable on other data-sets

Minutes

- Mauro shown eye diagrams and time series pictures
- Mauro says he would like to know how much training data is required for working DeepESN
- Mauro suggests potentially citing the masters group