UPPSALA
UNIVERSITET

UPTEC IT 23009
Examensarbete 30 hp
June 11, 2023

Utilizing Neural Networks To Adaptively Demodulate And Decode Signals In An Impulsive Environment

Andreas Andersson

Civilingenjörsprogrammet i informationsteknologi

Master Programme in Computer and Information Engineering

UPPSALA
UNIVERSITET

Abstract

Utilizing Neural Networks To Adaptively Demodulate And Decode Signals In An Impulsive Environment

Andreas Andersson

Electromagnetic disturbance can be detrimental to the performance of a radio communication system, and in today's society where more and more electronic devices are present in our everyday life it is increasingly vital to consider man-made interference. A communication system can take into consideration the noise characteristics and by doing so will excel in such areas, however, this follows that the algorithms utilized in such systems are more computationally complex and are therefore slow. This master thesis aims to explore the possibility of a neural network-based solution that reaches the same accuracy, as existing methods, but more quickly. Numerous different existing model alternatives have been explored and a plethora of different improvement techniques have been outlined. Two models, Hannet and Lannet, have been designed and improved to enable adaptive demodulation both including or excluding decoding at the receiver in an end-to-end communication system.
The evaluation results demonstrate that the proposed models are comparable and in some cases even more accurate than current standardized methods. However, the models are unable to fully learn the decoding algorithms present in the experiments. Thus even though demodulation by itself thrives, performing decoding in conjunction with demodulation is out of reach for these models.

Teknisk-naturvetenskapliga fakulteten
Uppsala universitet, Utgivningsort Uppsala/Visby
Supervisors: Erik Axell, Kristoffer Hägglund
Reviewer: Kamran Forghani
Examiner: Lars-Åke Nordén
UPTEC IT 23009

1 Acknowledgements

I would like to express my deepest gratitude to everyone at the Department of Radio Communications (FOI), especially my supervisors Erik Axell and Kristoffer Hägglund, for giving me the opportunity to work with them and for all their support and assistance throughout the project. I'd also like to give a special thanks to the department head Peter Stenumgaard, for giving me the opportunity to perform this thesis at their facilities. Lastly, I would like to thank my subject reviewer, Kamran Forghani, for helping me every step of the way and for always making time to answer my questions. I am truly thankful for everyone's hard work and dedication in making this project a success.

Contents

2 Introduction

In modern vehicles, several coexisting electronic systems create electromagnetic interference which may impair the performance of wireless communication. A communication system's performance largely depends on the noise characteristics, which in practice usually derives from other electric equipment [31]. Many communication systems merely take into consideration additive white Gaussian noise (AWGN) [37], however, impulse noise characteristics are prevalent in many situations [48, 52] and the AWGN model does not account for every behavior [26]. By altering the demodulation or decoding in a communication system based on the noise characteristics it will hopefully yield performance improvements.

Deep learning (DL) systems are a new way of rethinking the communication problem and have seen a rise in popularity in recent years [46]. Neural networks are particularly interesting when it comes to tailoring a system to a specific environment [37]. Most algorithms for signal processing are proven optimal for tractable mathematical models, which are stationary, linear, and have Gaussian statistics. Since reality has many non-linearities and imperfections a DL-based communication system is a reasonable alternative since it can be optimized for specific hardware and channel configurations, and does not require a mathematically tractable model.

The standard method of communication, where communication can be divided up into multiple independent blocks with their isolated functionality, is not necessarily the clear optimal method for end-to-end communication. Neural networks are shown to be universal function approximators and are thus a reasonable area to explore when it comes to communication methods with their accuracy being comparable to state-of-the-art methods [37, 19].

This project explores using neural networks as an alternative to the traditional method of end-to-end communication. In contrast to previous research that has focused on creating a functioning machine learning system to adapt to AWGN, this project aims to experiment with a different stochastic noise model representing a more realistic environment.

2.1 Purpose and Goals

The purpose of this project is to implement an adaptive demodulator either including or excluding decoding using neural networks, for Hamming and low-density parity-check encoding. In this setting, adaptiveness means that the model should adapt to the symmetrical alpha-stable distribution, see section 3.1.3, with varying α values, or at least not neglect any α values. This objective can further be divided into three different goals:

Data Generation - Researching and implementing a method for synthesizing signal data that has propagated an environment with present man-made interference.

Model - Define and improve a neural network model that utilizes the given synthesized data and predicts the original message in that data.

Evaluation - Compare this project's proposed neural network model to existing state-of-the-art solutions, such as other neural networks, standardized demodulators and decoders, to determine whether the solution using the proposed model can be used in practice in a realistic communications environment.

The goals were completed in chronological order as they are dependent on each goal before them. By achieving these goals the hope is to further expand the knowledge regarding signal processing through machine learning.

2.2 Contribution

From the completion of the goals defined in section 2.1 a number of expansions within the field of machine learning-based signal processing are extracted:

Machine learning receivers - Expanded on existing knowledge regarding the design of machine learning-based implementations of demodulators, decoders, and choice of encoding, to improve receiver message interpretation accuracy, by examining their utilization in a scenario where man-made noise is present.

Adaptive communication - Furthered the research regarding *adaptive* demodulators and decoders, in terms of increasing robustness, specifically in an environment with impulsive noise where the system should adapt accordingly to the characteristics of the noise.

2.3 Limitations

There is one limitation present in this work and that is the range of α, expanded on in section 3.1.3, upon which the model is evaluated. Early on in development, it was found that using low α values created numerical issues when calculating the loss function, as a result, the optimizers were never able to lower the loss functions. This evolved into the decision of only evaluating the models over the range of $1.0 \leq \alpha \leq 2.0$. A real world example is seen in the city of Aalborg [15]; measurements from five different locations show that heavy-tailed interference is present in IoT communications. The α-stable distributions fitted to the findings yielded an α range of $0.9 \leq \alpha \leq 1.9$. By confining the value of α to the interval [1,2], the edge-cases of the parameter range can be achieved using closed-form expressions such as Cauchy and Gaussian distributions. A few studies[25], [32] have found that a mixture of Cauchy and Gaussian distributions best express the general distribution for α in this range.

2.4 Related Work

Prior to this project, many other researchers have investigated similar topics such as demodulation and decoding through neural networks and a variety of different methods, such as noise distribution estimation in combination with a standard demodulator, to facilitate these features. Some of these, which are listed below, were the main inspiration that led to, the research made in this project as well as how it was conducted.

Adaptive Demodulation in Symmetric Alpha-Stable Impulse Noise Channels written by *Kristoffer Hägglund* and *Erik Axell* [27]. As the authors of this article are supervisors for this project, understandably, they have delved into this research previous to this project. In this article, an adaptive demodulator is proposed for an environment where non-Gaussian interference more commonly occurs. The modulator produces, from the received symbols, a vector of log-likelihood ratios (LLR) corresponding to all the bits in the transmitted signal. The LLRs are based on an estimation of the interference, from the channel, that the receiver experiences. The interference is estimated by the demodulator through four different methods where it calculates an approximation of the parameters used in a SaS-model to model the noise as close to the received signal interference as possible. The demodulator is compared to other existing demodulators via the Monte Carlo method and is seen to outperform these, in some more impulsive scenarios it outperforms upwards of 20dB. A drawback with the method proposed in [27] is the time-intensive mathematical calculations being done, which is one of the main reasons why a different approach is being explored in the work presented in this report. This article highlights the need for research in scenarios where the noise is modelled

with another noise distribution.

On Deep Learning-Based Channel Decoding written by *T. Gruber*, *S. Cammerer*, *J. Hoydis* and *S. Brink* [23]. In this article, the authors explore the possibility of neural networks being able to perform decoding of different types of codes. They found that neural networks were able to generalize for structured codes, which looks promising for the possibility of neural network-based decoders being used in the future. Currently, decoding suffers from a lack of parallelization and high complexity which results in an unavoidable high latency. On the contrary neural network designs do not suffer from the same deficits and are therefore a reasonable alternative given that they can reach the same accuracy. The main takeaway from this article is that neural networks are able to decode codewords, in some manner. The authors also mention a topic which is further discussed in section 3.2.7, where an issue with neural networks is that they must classify each input sequence as a specific output. This causes issues, in regards to network complexity when the amount of bits in the codeword increases as this also increases the amount of output by a power of two. The proposed neural network consists of several dense layers stacked after one another.

DemodNet: Learning Soft Demodulation from Hard Information Using Convolution Neural Network written by *S. Zheng*, *X. Zhou*, *S. Chen*, *P. Qi*, *C. Lou* and *X. Yang* [56]. In contrast to what was attempted in [23], where the traditional decoding was exchanged for a neural network solution, here the demodulation is constructed through a neural network. In the same way that the received symbols were transformed into LLR values in [26], so does the demodulator in this article however as Log probability ratios (LPRs). The difference between work done in [26, 23] is that in [23] proposes a solution based on neural networks and focuses on Gaussian noise rather than SaS. Under the disturbance of additive white Gaussian noise, the model performs on par with traditional methods, in terms of accuracy, however when the disturbance is of additive generalized Gaussian noise the system surpasses traditional methods. The model architecture consists of a deconvolution layer followed by several convolution layers, excluding batch normalization and activation layers. Given the results presented in this article, it seems highly likely that using neural networks to demodulate a signal is entirely possible.

MSK Demodulator and Impulsive Noise Depression Based on Convolutional Neural Network with Gated Layers written by *Q. Tan* and *L. Zhao* [49]. In this paper, the authors found that most demodulators only consider Gaussian-distributed noise. This demodulator is aimed towards more impulsive noise and has therefore been trained on Symmetrical α-stable (SaS) noise much like that of the work done in [26], rather than an implementation aimed towards Gaussian noise seen in [56], therefore the network schematic differs from the model described in [56]. In the model, two convolution layers are stacked on each other followed by a GatedNet and finally a global average

pooling layer feeds the information to the output. The proposed architecture outperforms other neural network architectures, consisting of convolutional layers followed by dense layers, by approximately 2dB. This aspect, namely processing signals which have been disrupted by *SaS* noise using neural networks, is a similarity between this project and this article [49]. However, it is noteworthy to mention that the performance of this method, in regards to accuracy, is not as great as that of the prior ones mentioned [26, 23, 56].

3 Technical Background

First and foremost it is important to gather an understanding of the problem scenario itself and how previous solutions have aimed to tackle it. Even though there is a focus on neural networks in this project, it is important to research the areas of communication and noise distributions to better formulate a solution. There are many ways to build a neural network, therefore, it is important to establish a good base for how to begin creating one for this scenario, and what consequences different choices in design ultimately have. The research performed in this section will stand as a basis for the methodology conducted.

3.1 Communication Techniques

Modern communication through electromagnetic waves is usually modelled into blocks, see *Fig.* 1, of different procedures which have been perfected over the years. These functionalities facilitate end-to-end communication, making it possible for a transmitter and receiver to send information over a channel [40, p. 2]. The design of these block diagrams differs depending on the design of the system, however, two different block pairs seem almost mandatory for end-to-end communication to function properly. These are Modulation/Demodulation and Encoding/Decoding for the transmitter and receiver respectively.

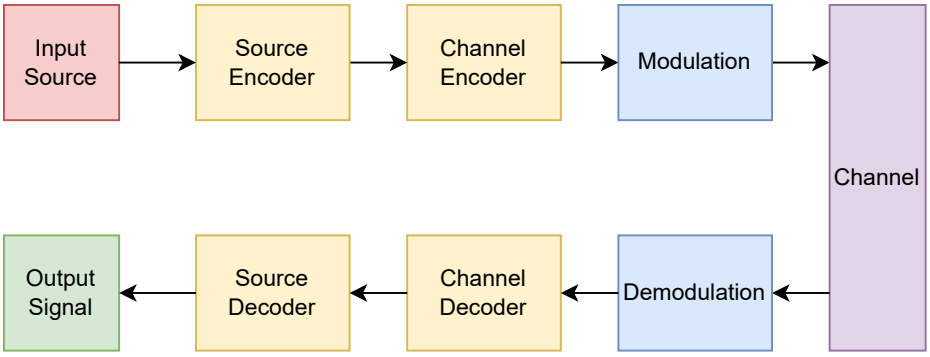


Figure 1 Typical radio communication block diagram. The purpose of this project's proposed model is to replace the channel decoder and/or the demodulation blocks.

3.1.1 Demodulation

The purpose of modulating a signal is to send data across a channel, i.e. associating the signal to a frequency(s), in a cost-effective way. When the modulated signal has reached its destination it is the duty of the receiver to convert the signal back into its previous representation [40, p. 3], before modulation.

Since electromagnetic devices share the same frequency space it is important to divide this space for the devices to be able to communicate without disturbing each other [13, p. 187]. This has given rise to researching new innovative methods of representing multiple bits with a single frequency. One of the earliest modulation methods invented was Amplitude modulation (AM), it functions by the transmitter increasing or decreasing the amplitude of the frequency in the signal depending on if it is sending a low or high bit. This method as well as Phase modulation (PM), where the phase is altered instead of amplitude, is the basis for the modulation type which will be explored in this work.

By expanding on the method of representing different bits with different ”states” of the frequency it is possible to instead represent different combinations of bits. By using neural networks the hope is to be able to implement adaptive demodulation of one of these techniques, namely quadrature phase shift keying (QPSK). QPSK, which shares its constellations with four quadrature amplitude modulation [13, p. 217-219], utilizes phase modulation to represent more bits see *Fig.* 2. QPSK has four different states, usually referred to as symbols, which means it can represent two bits at any given symbol. When there is no noise present in the signal it is theoretically possible to keep increasing the number of states. However, using more state representations will also make the system more vulnerable to noise.

In certain scenarios, it is desired to have a probability related to the symbol, regarding what bits it represented before channel propagation. When the demodulator outputs such values it is called soft demodulation.

3.1.2 Decoding

The second communication blocks to consider are the encoding and decoding pairs. When a message propagates through a channel there is a chance that data, due to noise, is corrupted to some extent. Even though the demodulator may handle some minor deviations in the original signal and the received signal, some errors may be too large to be corrected. To prohibit this from deteriorating the communication, the signal is first encoded before it is sent, by adding a few redundant bits. These redundant bits can be used later on the receiving end to rebuild any corruption in the message [40, p. 2].

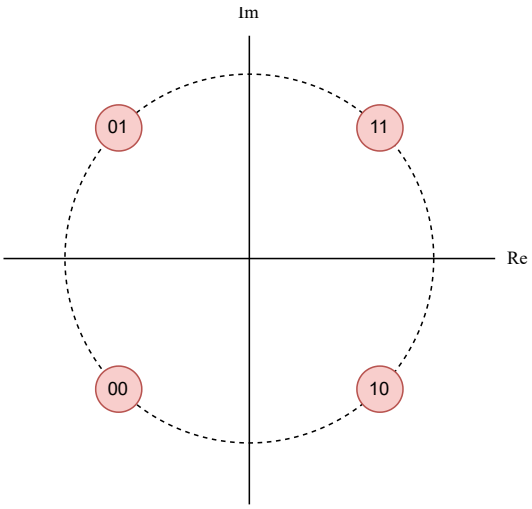


Figure 2 An example of quadrature phase shift keying constellation locations, and their corresponding bit sequence.

An important metric to consider when encoding a message is the code rate, defined as $R = k/n$, where k is the message length and n is the codeword length produced from encoding the message, as lowering the code rate may increase robustness but will inevitably decrease throughput. There are many ways to accomplish this, in this work the Hamming 7.4 and low-density parity-check (LDPC) encoding methods will be the focus [34, p. 34, 634].

3.1.3 Noise

As mentioned earlier, the signal is corrupted while propagating through a channel. Such alterations of the signal are due to the presence of noise, be it intentional disruptive attacks or unintentional interference. To be able to properly test an end-to-end communication method without deploying it one can model the noise, adhering to some distribution, and alter the transmitted signal accordingly to simulate interference. The most common way to simulate noise is to use the AWGN model [35, p. 1], however, this distribution does not imitate reality in every scenario. Other distributions such as the symmetric α-stable distribution are much more fitted for certain environments where for example man-made noise is present.

Gaussian

The Gaussian model has traditionally been prevalent when modelling noise in signal processing. By using additive white Gaussian noise or additive generalized Gaussian noise it is possible to greatly decrease the complexity of the design, and analysis, of the system. The reason behind adding this to a signal when analyzing a system performance is to examine how well the system functions an environment closer to reality [35, p. 1]. The equation for calculating the probability density function, which can be seen in *Fig.* 3, is shown in Eq (1) where μ is the mean and σ^2 is the variance. The effect on a signal's constellation points can be seen in *Fig.* 4a.

$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma^2})^2} \tag{1}$$

Symmetrical Alpha-Stable

Unfortunately, not all scenarios of reality can be replicated with Gaussian noise, many situations bring about a more impulsive noise. In areas where man-made noise is present, it is better to model the noise after the α-stable model. The difference between the Gaussian and the α-stable distribution is that the α-stable distribution has heavier tails the lower α is, see *Fig.* 3, which makes it more suitable for modelling a more impulsive environment. The α-stable distribution is even modifiable by altering the α parameter, which then also includes the Gaussian distribution if α is set to the highest value of $\alpha = 2$ [35, p. 2-3]. Besides the α parameter, the α-stable distribution also has three additional parameters. The skewness of the distribution is determined by $-1 \leq \beta \leq 1$, where a negative value implies skewness to the left whereas as a positive value indicates a skewness to the right. $\delta \in R$ indicates the mean of the distribution and $0 < \gamma$ indicates the scaling, which functions much like the variance in the Gaussian distribution. In Eq (2), (3), and (4) the probability density function can be seen, where *sign* is the signum function. For the scope of this project only the α value will be alternated as the distribution examined is intended to be symmetrical. The parameters will be set to $\delta = 0$, $\gamma = 1$, and $\beta = 0$. Below can also be seen the effect of simulating an impulse channel by applying symmetric α-stable noise to signal symbols, see *Fig.* 4b. Notice that the effect has thrown the symbols more off-course than that of the Gaussian noise.

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \varphi(t) e^{-ixt} dt \tag{2}$$

$$\varphi(t;\alpha,\beta,\delta,\gamma) = exp(i\delta t - \gamma|t|^a(1 - i\beta sign(t)\phi)) \tag{3}$$

$$\phi = \begin{cases} \tan(\frac{\pi\alpha}{2}), & \alpha \neq 1 \\ -\frac{2}{\pi}\log|t|, & \alpha = 1 \end{cases} \tag{4}$$

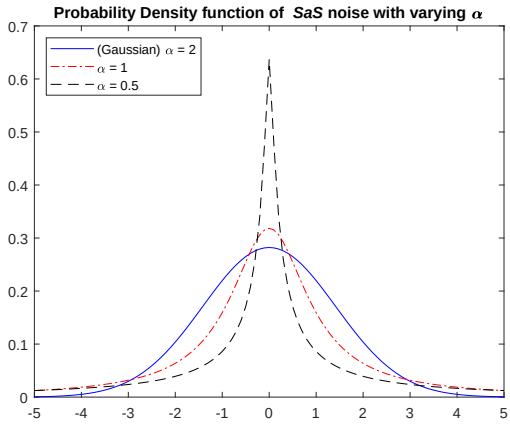
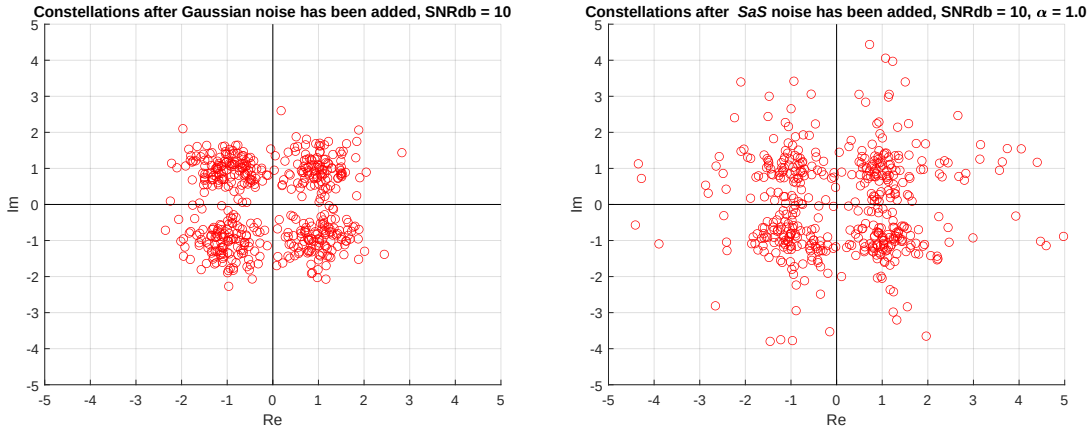


Figure 3 Probability density function for the SaS noise model with varying α.



(a) Channel modelled with Gaussian noise. **(b)** Channel modelled with SaS noise.

Figure 4 The effect on the constellations after the signal has propagated through a channel.

Signal to noise ratio

Regardless of which way the noise is modelled, there is another factor to consider namely signal-to-noise ratio (SNR). The SNR is a measurement of how strong the signal is compared to the noise [13, p. 160]. One can imagine that if the noise is as strong as the signal then the slightest interference will cause heavy corruption in the message.

This is also why it is difficult to use a sensitive modulation scheme in an environment such as the one focused on in this paper, where the noise is highly prevalent. In this report, SNR will be referred to in the scale of decibels (dB).

Training Bits

By adding a known sequence of bits at the beginning of the codeword it is possible to relay some information to the receiver regarding the distribution of the channel noise. This will in some regard lower the code rate but could give performance improvements. An experiment was done in which it was found that adding a sequence equal to the size of 20% of the original message lowered the bit error ratio [26]. However, it is worth noting that this experiment was performed on a system that was not a neural network, which follows that this does not guarantee an improvement in this project.

3.2 Neural Networks

Artificial neural networks are born from the notion of trying to emulate how the brain functions. Much like the brain, neural networks can be seen as complex, nonlinear functions that mostly can be run in parallel [18, p. 5]. The hope of creating neural networks was to create a network that could memorize, learn and even generalize scenarios to create an abstraction of functions which would otherwise seem impossible to define. In essence, this is what a neural network is, a non-linear mapping from one real vector space to another [18, p.15]. Even though computing power today is steadily increasing, and neural networks are designed to mimic brain function, it will take years for neural networks to reach the sheer size of a human cortex. However, for smaller tasks or rather more narrowly defined tasks, neural networks are doing wonders. Some field examples of where neural networks excel are pattern matching, optimization, data mining and classification, among others.

3.2.1 Artificial Neurons (Perceptrons)

Artificial neurons, also known as perceptrons, are the fundamental building blocks that make up all neural networks. *Fig.* 5 shows a neuron as a part of a neural network. The inputs received $\vec{X} = [x_1, x_2, ...x_n]$ by the neuron are either from the environment, meaning the inputs we feed the neural network, or from other neurons.

All the inputs received by the neuron consist of the original input signal multiplied by some weight $\vec{W} = [w_1, w_2, ...w_n]$ associated with each specific signal $x_n w_n = i_n$ [18, p. 6, 17]. In Equation (5) the inputs $\vec{I} = \vec{X} \cdot \vec{W}$ are then summed up and a bias is

added[16, p. 50] by the neuron which processes the inputs and determines, through an activation function f, how strongly to forward the signal. The sum of the inputs and the bias is often referred to as the signal net sum.

$$o = f(\sum x_n w_n + \text{Bias}) \tag{5}$$

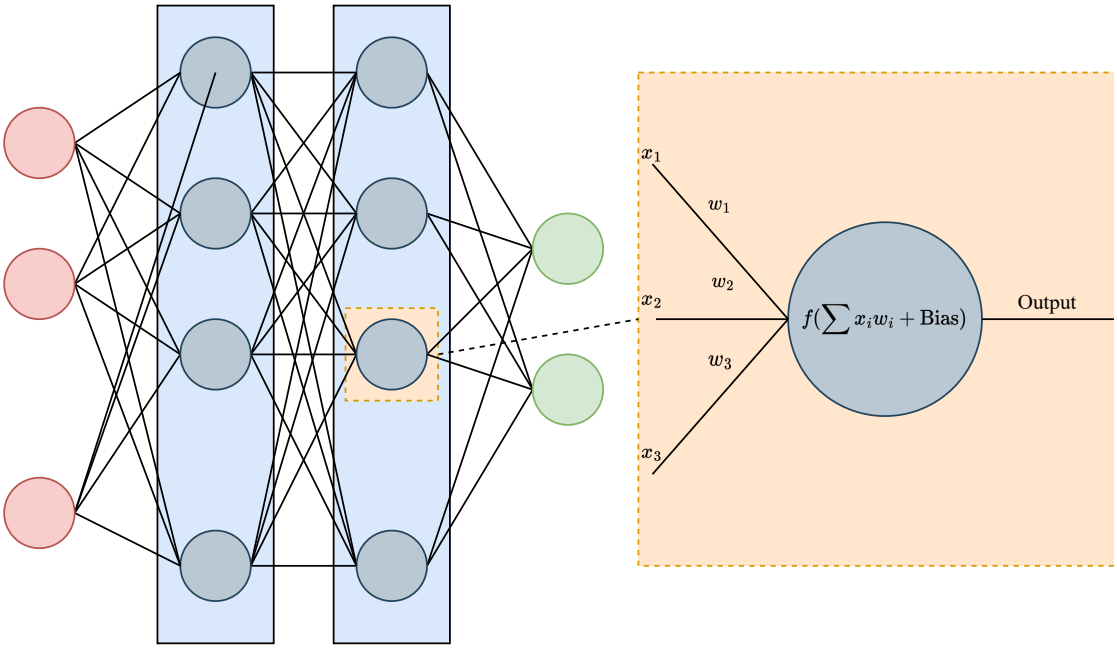


Figure 5 A feedforward network with two hidden layers with a closer look at one of the perceptrons within it.

The weights associated with the inputs are essentially the portion of the neural network that changes for the network to learn. The output signal o from the neurons may either be forwarded to other neurons or be considered as the output of the neural network. What determines how strongly the neurons output ”fires” will be the activation function within the neuron. There is always the possibility to define one’s own activation function, however, there are some functions which are more commonly used in practice. See section 3.3.10 for more information on these activation functions.

3.2.2 Feed Forward Networks

On the left, in *Fig.* 5 a feed-forward network (FFNN) is shown, in this case, a multi-layer perceptron (MLP) in contrast to a single layer perceptron which only contains one hidden layer. Though the network in the image shows a network consisting of two hidden layers, there is no limit to how deep the network may go. The feed-forward network was the first and simplest design of a neural network. Through backpropagation, elaborated on in section 3.2.6, the FFNN can estimate any continuous function, with as little as only one hidden layer given that the width of the layers is sufficient [18, p. 28]. As seen in the *Fig.* 5 all the hidden layers consist of many fully connected neurons. What characterizes a feed-forward network is the lack of cycles due to no feedback connections, where the outputs of the layers are sent back to previous layers in the network.

3.2.3 Convolutional Neural Networks

Convolutional neural networks (CNN) function by using sparse interactions, parameter sharing and projecting the gathered features on different channels [33, p. 147]. To understand how this is significant we must first understand what these different methods are.

Sparse interactions signify that the nodes in the CNNs do not necessarily connect to all the data available in the previous layer. As can be seen in *Fig.* 6 each resulting node only connects to 4×2 nodes from the previous layer. The idea behind this is that in some scenarios data closer to each other is more related than data further away. As the window, referred to as a kernel, traverses the data the nodes are filled with information. In the example in *Fig.* 6 the data is two-dimensional, however, the same concept of CNNs can be applied to one or even three-dimensional space. This concept would naturally overflow the number of weights needing to be trained very quickly if it were not for parameter sharing.

Parameter sharing facilitates, as the name would suggest, sharing of kernels between the nodes of the convolutions layer. Therefore, the kernel will look the same for each node reading from the previous layer, such that the recorded data in the nodes will be some feature belonging to the data in the previous layer. To be able to gather more information from the previous layer more channels are added, each with its kernel of weights. One can see it as every node in every channel containing one feature about a certain segment of the output of the previous layer. The aim of these features is to find some recognizable pattern in the data.

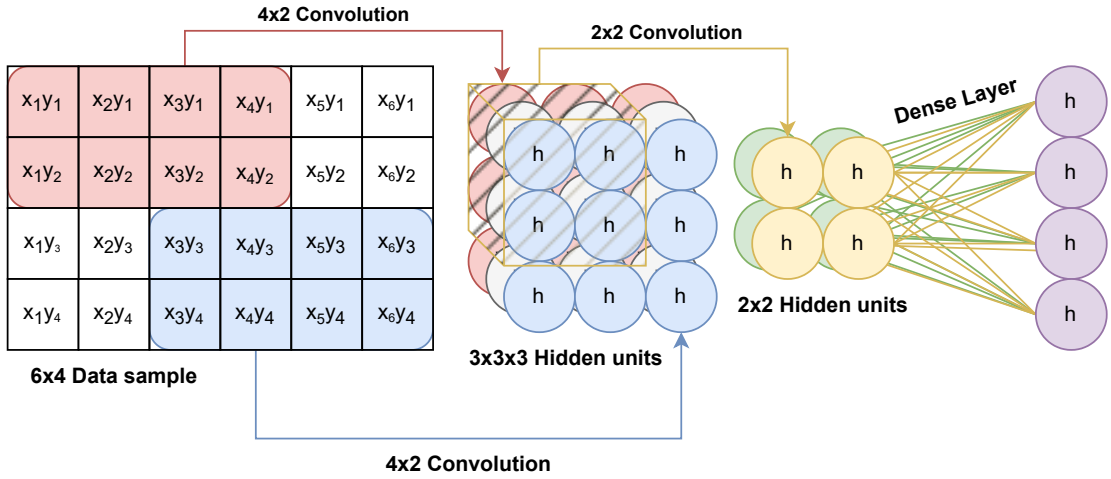


Figure 6 A fully convolutional neural network with a dense layer output. The first convolutional layer performs 4×2 convolution and extracts three features whilst the second performs 2×2 convolution and extracts two features.

By default a convolution of some data will result in dimension reduction, in some cases, this is unwanted for example in combination with residual networks. There are methods to avoid dimension reduction, one such is called zero padding. Zero padding causes the kernel to read the values zero outside of the border of the information, which allows it to create a node space equal to the dimensions of the previous layer. Another parameter of a convolution layer that can be employed is the strides. Previously reads from the data from the previous layer would intersect, because the kernel only jumps one step in the input for each node. By increasing strides we are able to increase this step length, naturally, this will also lower the dimensions of the data further.

As previously mentioned convolutional layers are, optimally, able to track a different feature for each of their channels. This creates a third dimension of the data, for the layers that precede. If the next layer is another convolutional layer, then all nodes of that layer will consider all features of the previous layer as can be seen in *Fig.* 6. The deeper the network stacks convolutions, the higher complexity of features that will be able to be learned by the network.

Closely related to the convolutional layer, and also commonly used in CNNs, is the pooling layer. The pooling layer examines a portion of the data of the previous layer and extracts a value dependent on that data, much like the convolutional layer. However the pooling layer does not calculate any specific weights in its kernel, it simply performs an algorithm on it such as average or maximum.

Both convolution and pooling have their reversed methods, which are referred to as

deconvolution or transposed convolution and upsampling respectively. Where pooling extracts a value from some set of nodes in the previous layer, upsampling will copy a node from the previous layer [54], multiplying the number of nodes with that value. As for deconvolution it is just as complex as a convolution however works in the reverse direction [47]. Where convolution examines for example a 5×5 kernel and decides upon a value dependent on that, a deconvolution will examine a single value and create a 5×5 grid based on a kernel.

3.2.4 Recurrent Neural Networks

Recurrent neural networks (RNN) are networks which have layers with feedback connections. This facilitates memory for the layers with such connections [22], allowing layers to retain information about past contexts. Given data where context is important, such as speech recognition, RNNs can excel. There are different ways for these feedback connections to be implemented, one such implementation is the long short-term memory (LSTM). The LSTM is part of a family of RNNs called gated RNNs. In practice the memory that an RNN can hold is quite limited and suffers from the input of the layer either decaying or blowing up between the network's recurrent connections, a problem referred to as *exploding gradient problem vanishing gradient problem*. Even though the event of the gradient blowing up is solved quite simply with various clipping strategies, the problem of vanishing gradient still remains [43, 38]. LSTMs attempt to address this problem by adding connected subnets, which act as memory blocks or states, each with a set of internal units activated using an input, forget and output gate. By creating such a gate system the LSTM memory can avoid information being overwritten and is only being accessed when necessary. Modern LSTMs also utilize *peephole connections* [44] from their internal units to the same unit gates to learn the timing of the outputs.

3.2.5 Residual Networks

A problem with stacking dense layers to learn more complex function approximations is diminishing backpropagation. This causes the early layers of the model to not get updated and will therefore not pass on any important information to the layers that are able to learn, commonly referred to as the *degradation problem* [24]. In an attempt to solve this problem, residual networks were introduced. The residual network achieves this by implementing shortcut or skip connections, see *Fig.* 7. By doing so it is possible to create deeper networks, yet still utilize the same backpropagation and optimization method (SGD or such), therefore reducing the training error and increasing the test accuracy in comparison to previous attempts of networks with the same depth.

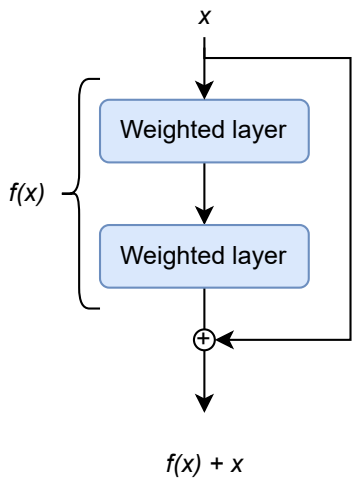


Figure 7 Example of a skip connection which can be found in a residual neural network.

3.2.6 Network Learning

There are different ways in which a model may receive feedback and develop an understanding of the function it is attempting to approximate. These are *reinforcement learning*, *unsupervised learning* and *supervised learning* [18, p. 21], the latter being what will be used in this research. In supervised learning, the network is trained on a dataset where the expected output is provided and thus already known by the network [18, p. 27]. The goal is to train the network such that the weights minimize the difference between the expected output and the actual output of the network. Once training has ended the hope is that if the network is given a sample of inputs, similar to that of those in the training dataset, it will be able to correctly estimate the desired output.

Backpropagation is the method which facilitates the entire learning process for a neural network. As previously mentioned, for the network to approximate the function the weights therein must be tuned properly. This is done by examining the desired output of the network and then tweaking the weights as to force them to give the desired output, given the same input. In such a manner it is possible to move backwards in the network and tweak each weight associated with a neuron, a so-called backward pass [39]. Because all neurons of a preceding layer have certain wishes for those of the prevision layers, all these changes must be taken into account when changing the weights. Once an idea of how the changes to be made should be realized the algorithm continues to the next layer. Even though the inner layers may only have an estimation of error it is usable as a reference to how they should expect their prior layer to fire.

3.2.7 The Curse of Dimensionality

In regards to constructing a neural network demodulator or decoder in an end-to-end communication system, there arises a complexity issue [23]. The issue is referred to as the *curse of dimensionality* and raises awareness of one of the limitations of neural networks. Since whatever binary message is being sent from the transmitter must be classified by the receiver, there must exist 2^n classifications to be able to interpret all messages. In modern communication systems, short error correcting codes (ECC) such as Hamming 7.4 are not a viable option for transferring data, as they lack robustness. However, creating a network which can classify high-bit-lengthed messages will be too complex, have too many weights and thus be much too memory intensive for practical use.

3.3 Designing and Training Models

Since neural networks often are considered a black box, albeit a box which one can peek into, there is no textbook way to define a network or which parameters to use for the best results. One can only speculate over, mimic and compare different networks to find a solution that will handle the provided data and give the desired predictions. However, there are a variety of topics to consider when attempting to optimize a neural network. These can relate to changing the architecture of the model entirely or simply how much or even how training is done. Many of these features have already been implemented in the frameworks used today and are commonly referred to as hyperparameters.

3.3.1 Scaling and Normalization

The first step to consider is if the data being used is properly defined for the model. If the features being fed into the network are not of relevance or perhaps even corrupted this must be handled before the learning process even begins. Because the data in this project is completely synthesized many of these methods become redundant. However, one such preprocessing method may be necessary to do, which is to simply scale the data sent to the network. The goal is to scale the values in such a way that they are within the range of all used activation functions see section 3.3.10, this method is referred to as amplitude scaling [18, p. 102]. When utilizing supervised learning it may greatly increase the performance of the algorithm if the output target data is scaled properly.

3.3.2 Data Splitting

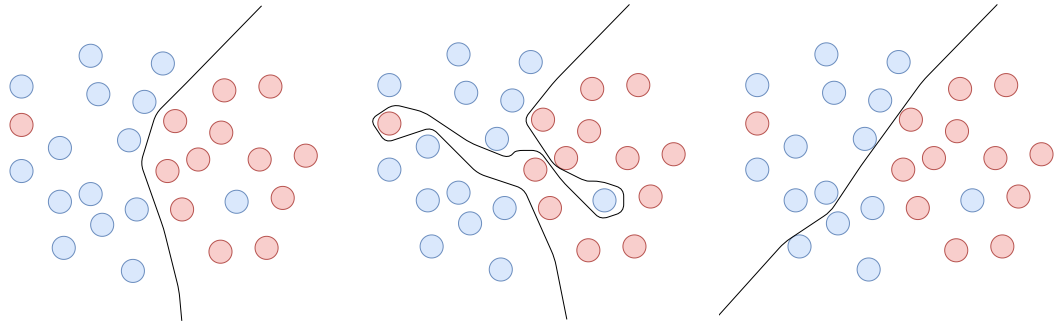
A common practice when training networks is to perform a data split. Data splitting is done by extracting a subset from the available data to train the model and discarding it during training. Once training has been performed the subset is used to perform a sort of cross-validation [29, p. 198]. The idea is that the subset should mimic the data which the model has been trained on, however, it should also be data which the model has never encountered before. The performance of the subset, often referred to as a test set or hold-out set, will then be evaluated and compared to that of the training set.

How large of a percentage of the original data is used as a test set differs, depending on how much data is available for training. However many neural network modellers find themselves using around 10-30% of the entire dataset [36, 50]. The reasoning behind this method is to examine if the model is being overfitted, see section 3.3.3, or is able to further converge to a better solution. The goal of a model is usually not to exact a function based on data but rather approximate, to capture features of the input data and interpret it.

3.3.3 Overfitting

Further training a model on a dataset will indeed create a better fit for the data which is present in the dataset. However, a better fit for the trained dataset does not necessarily follow a better fit for the data seen in future predictions [14], see *Fig.* 8.

In section 3.3.2 it is explained that data is often split into a training and test set. By examining the difference in loss between the two sets we can determine if the model has been overfitted or not. If the loss during training is lower than the loss during testing it



(a) Good fit **(b)** Overfitting **(c)** Underfitting

Figure 8 Three different degrees of data fitting in a classification problem, the line represents the border of the decision.

is usually an indication that the model has been overfitted. What has occurred is that the model has memorized the training set and lost the ability to generalize. Overfitting may materialize due to a too-deep network, meaning too many weights to train and too many redundant input data points. Another reason is that the model is trained for too long. As a consequence, the model begins to memorize all the data, even the noise present in the dataset. It is possible to determine how long is ”too long” for a model to be trained by examining the loss of the training set compared to the test set between each epoch. Once the loss of the test set begins to increase, whilst the training set loss is still decreasing, it is evident that the model has begun to overfit [18, p. 96]. There are a couple of tools which can be used such as regularization or dropout which attempt to lower the chance of overfitting.

3.3.4 Regularization

Regularization is done by adding a penalty factor to the loss function, changing the total error to $E_{\text{total}} = E_{\text{loss}} + \lambda E_{\text{penalty}}$ where λ scales at which rate the penalty factor contributes to the total error [18, p. 110]. The penalty term is meant to penalize a complex network (large network), in an attempt to lower the chance of overfitting. There are different ways to calculate the penalty factor, all with different intentions and ideas on how to manipulate the weights. One such is weight decay $E_{\text{penalty}} = \frac{1}{2}\sum w_i^2$, where w are the weights in the network. The goal with weight decay is to force small weights to be equal to zero. The regularizors used in this project will be **L1**, **L2** and **L1L2** [5]. Equations (6) and (7) below show how these are calculated, where L1L2 is simply applying both penalty factors.

$$\text{L1}_{\text{penalty}} = \lambda \sum w_i \tag{6}$$

$$\text{L2}_{\text{penalty}} = \lambda \sum w_i^2 \tag{7}$$

3.3.5 Dropout

Because the performance of a neural network is stochastic in nature it is reasonable to assume that given the same architecture and the same training data the weights will not always converge to the same values. One way to combat this is to train several models and then average the weights of the models [33, p. 155], to reduce the variance between the models and by doing so also reduce overfitting. However, since training a single model may take a substantial amount of time it is not always feasible to do so. Dropout is a technique which allows for this to be done without the need to separately train the networks. Instead, we use the original architecture and drop some of the units in the network and train the network with a new iteration of these subnetworks for each batch. It is worth noting that since the connections, and by extension their weights, for some nodes have been removed they will not be updated during that batch. The dropout technique can be used to the degree the modeller wishes, meaning it can be layer specific or used on every layer in the entire network [1].

3.3.6 Learning Rate

This value dictates how greatly the weights are altered with each update. If the learning take is too large then the update may overshoot the local or global minimum, on the other hand, if it is too small the steps convergence may take an unnecessary amount of steps [18, p. 107]. Smaller learning rates also run the risk of ending up in a local minimum, even though a better local minimum is close nearby. Since both magnitudes of learning rate cause problematic behaviour it is common practice to dynamically change the learning rate during training. The idea is to compare the loss of the current state and the previous state. If the network is converging too slowly then increase the learning rate and if the error is not decreasing fast enough or, perhaps is increasing, then lower the learning rate. Some optimizers already take this into consideration inherently with their own methods for altering the learning rate between epochs. The learning rate may be applied in such a specific manner as to assign a learning rate for each weight. The learning rate, therefore, gives the modeller a tool to change the impact of certain data on the model when fitting the model to a dataset.

3.3.7 Optimizers

The optimizers are used to decide what adjustments should be made to the weights of the model and as such has a large impact on the performance of a model [18, p. 109]. It is the method which defines how to move the weights in order to, hopefully, reach the global minimum of the loss function. There is always the possibility to write ones own optimization function, however in this work functions that are already defined will be the focus such as those given by the Keras framework [4]

3.3.8 Loss Functions

The function which defined how near the predicted output is from the desired output that is given $f(y, \hat{y})$. The goal is for $\hat{y} \rightarrow y$, and the closer this is the smaller the loss function is. From the loss function, a cost function is defined, which is the average loss over all the training data [33, p .40]. The goal of training is then as simple as finding the weights and biases which give the lowest value from the cost function. Naturally one can define one's own loss function, however, there are many predefined loss functions which are available for use in most machine learning frameworks [2].

3.3.9 Epochs and Batches

A batch refers to a subset of the training set. During training, the model only updates the weights and biases after an entire batch has been gone through [33, p. 125]. The size of a batch can be chosen by the modeller to be as large as seen fit, this should be experimented on to see what gives the best results. It is important that the batches are balanced and represent the entire training set as close as possible. Once the model has gone through every batch of the training set it is seen as completing one epoch. By increasing the number of epochs we essentially fit the model repeatedly on the training set, this is beneficial until the point of overfitting.

3.3.10 Activation Functions

As described in section 3.2.1, the activation function determines how the inputs of the neuron are handled and how strong the output signal of the neuron will be [18, p 17-19]. The only requirement is that the activation function has to be everywhere differentiable, otherwise one cannot perform back propagation [16]. For example, an activation function could be a simple linear function, a sigmoid function, a reLU function or perhaps

a leaky reLU function. The definition for these can be seen below in Eq (8), (9), (10) and (11), where λ is a scalar. By experimenting with different activation functions in different layers we are able to add another parameter that can be modified to increase the performance of the model.

$$f_{\text{linear}}(x) = \lambda x \tag{8}$$

$$f_{\text{sigmoid}}(x) = \frac{1}{1 + e^{-\lambda x}} \tag{9}$$

$$f_{\text{relu}}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{10}$$

$$f_{\text{leakyrelu}}(x) = \begin{cases} x, & x \geq 0 \\ \lambda x, & x < 0 \end{cases} \tag{11}$$

3.3.11 Adding Noise to a Data-set

Injecting noise into the data can help when trying to generalize the function of the network. Especially when there are a limited number of patterns to train on since this will generate new data for the network to be fit against [18, p .105]. As long as the noise is of normal distribution, with a small variance and zero mean the effects on the output should have little change. The result should be that the convergence of the network to an optimal solution is faster.

However, in this project, the signals are already by default injected with noise as a part of the data synthesizing. It has been is found that too heavy noise can disrupt the network, causing it to learn something completely different than the intended objective [42]. It is therefore worth examining at which point increasing the range of SNR or α is detrimental to the network, such that the noise is stronger in comparison to the signal.

3.3.12 Ordered Training

As previously mentioned in section 3.3.11, too much noise can negatively impact a network even though it is part of the dataset which contains signals that the model can typically expect to see. One method to iterate on the dataset, which is to be tested, is to order the data when training and lower the learning rate, see section 3.3.6, during training such that the noisier signals have less impact on what is learned by the network. An example of this can be found in [21] where simply changing the ordering of how the model was trained prompted an increase of accuracy on the entire dataset, including the signals with higher noise presence.

3.4 Signal Processing Model Background

As previously mentioned in section 3.3 there is no clear-cut way to develop a neural network to get the optimal performance. Therefore, in this project, a study was conducted to explore the design approaches used by others who have implemented similar functionalities in their neural networks. As a result of this research, Table 1 was created containing various implementations of network architectures as well as their accuracy. The prospects of this table can be seen in *Figs.* 9 and 10. It is worth mentioning that the different model alternatives explored goes beyond this table, however not all sources provided sufficient performance metrics and thus have been excluded from the table.

Table 1: Review of the related neural network models.

Source	Purpose	Noise Type	Network Type	Performance Metric	Year
[37]	Channel estimation	AWGN	DNN	BLER	2017
[53]	Channel estimation	AWGN	DNN	BER	2019
[23]	Decoding	AWGN	DNN	BER	2017
[28]	Decoding	AWGN	DNN	PER	2019
[36]	Decoding	AWGN	CNN	BER	2016
[17]	Decoding	AWGN	DNN	BLER	2018
[20]	Decoding	AWGN	DNN	BLER	2019
[45]	Demodulation	AWGN	RNN	BER	2020
[51]	Demodulation	AWGN	CNN	BER	2020
[56]	Demodulation	AWGN	CNN	BER	2020
[50]	Demodulation	AWGN	ResNet(CNN)	BER	2022
[55]	Demodulation	AWGN	CNN	BER	2018
[41]	Demodulation	SaS	DNN	SER	2019
[49]	Demodulation	SaS	CNN/RNN	BER	2019
[30]	Demodulation	AWGN	CNN	BER	2020
6.1	Demodulation	SaS	ResNet(CNN)	BER	2023
6.1	Decoding	SaS	ResNet(CNN)	BER	2023
6.2	Demodulation	SaS	CNN	BER	2023
6.2	Decoding	SaS	CNN	BER	2023

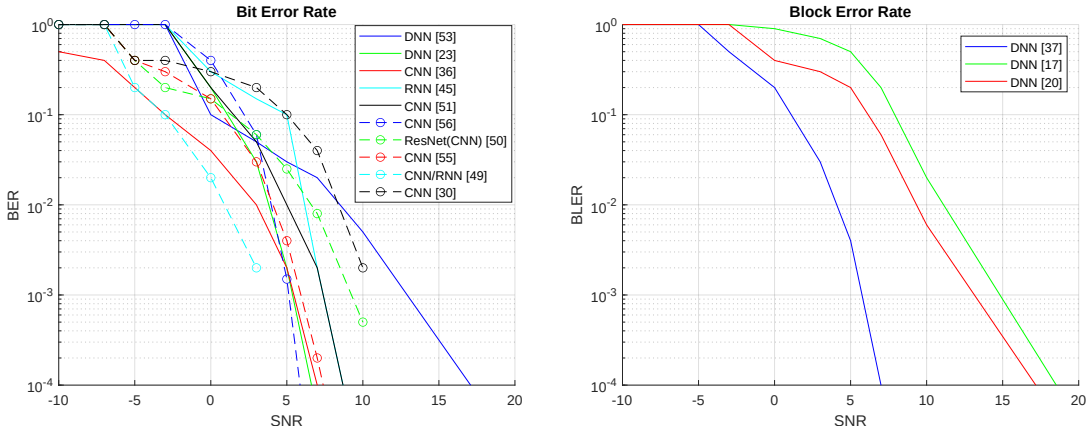


Figure 9 Bit and Block Error Rates of Source Models.

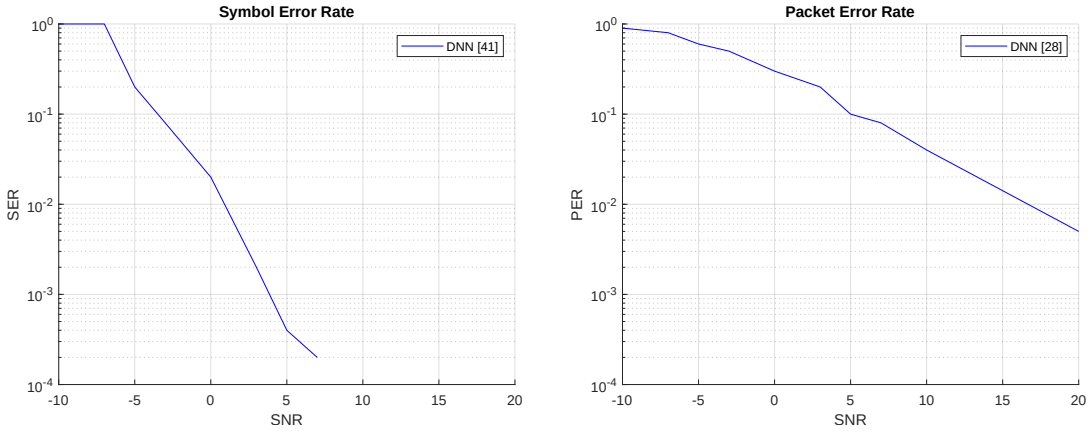


Figure 10 Symbol and Packet Error Rates of Source Models.

4 Methodology

With all the new knowledge assembled from researching the topic of neural networks a set of different tests were designed to attempt to create an enhanced solution, giving neural networks a fair chance to compete with standardized methods. The tools utilized, the data synthesizing process and which different tests were to be performed was fully defined from the beginning of the testing phase.

4.1 Software Tools

The software used in this project has mainly been Python-based, with an exception for encoding and decoding which is Matlab-based. Although the Matlab engine [9] was utilized to encode the data it was always done from within Python code, which facilitated the entire process of training and testing models. Keras [3], the python deep learning API, facilitated the neural network creation and learning procedures. Scipy was used to generate the α stable noise [11] as well as for Z-score normalization [12].

4.2 Synthesizing Data

In supervised learning, it is required that the model is provided with both the input data and the expected output data. Since it is easy to mimic data that a demodulator within a receiver typically would get it was decided that the data which the network would train on was to be synthesized complex baseband in-phase and quadrature components (IQ) data. This choice is reinforced by the fact that the comparison systems use the same type of data, see section 4.3.3.

To create the signal message a random sequence of bits is created using the numpy random number sequence generator [10], which is then fitted to either high or low bits. The length of this sequence was chosen to be 120, as this value is suitable for both encoding types. Initially, a python-based Hamming encoder was utilized, however as this encoder lacked a decoder which took into consideration soft demodulation choices it was decided to move over to a Matlab-based encoder [7]. To encode LDPC encoding, no properly functioning python method was found, therefore a Matlab LDPC encoder [8] was utilized instead. In order for the decoding and encoding to use the same parameters, for every signal, the same Hamming and LDPC parity matrices were used at all times. If training bits are added to the signal they are added as the next step after encoding and are always sampled from the same sequence of bits, as new sequences would naturally defeat the purpose of this method. This sequence of bits is generated beforehand by

uniformly randomizing values of ones and zeroes.

As the purpose of this work is to explore the possibility of implementing an adaptive demodulator including or excluding decoding for an environment with noise present it is also reasonable to choose modulation types which are not as impacted by disruption as others. This infers fewer nodes that are further apart from one another, therefore QPSK was chosen. The modulation is done by converting each set of bit samples to an IQ data sample, essentially performing QPSK. The manner for how this was done can be seen in Eq (12), which shows the generation of each IQ data point IQ_p from a given bit message sample B.

$$\mathrm{IQ}_p = 2\mathrm{B}_{2p} - 1 + (2\mathrm{B}_{2p+1} - 1)i \tag{12}$$

The noise to be added onto the modulated signal was created by using the levy stable randomized number sequence generator provided by Scipy [11]. Two sequences are generated and added to the signal in order to simulate channel propagation, one for the real portion and one for the imaginary portion. In order to ensure that this was the same noise model studied by *Kristoffer Hägglund* and *Erik Axell* [26], which experimented on the systems that this is to be compared to, the noise was generated using the same method as that project. It was found that the probability density function of the different methods [6, 11] were the same and that they therefore could be used interchangeably. Given that they are equal and the workflow is mostly python-based it was decided to use the Scipy version.

The original signals, either encoded or unencoded depending on the scenario, as well as the signals to be demodulated or demodulated and decoded, are then sent to the model for training. The final step of preparing the data is to normalize the values. This is done as a pre-processing step as the receiver will most likely have to do this whenever the signal is received if this system ever sees practical use.

4.3 Evaluation Methods

As this project proposes four different models it will also require four different methods, one for each model, to interpret the predictions made by these models and evaluate their performance. However, the data generated for these models will be done in the same manner for all four evaluation methods where the target data generated will be the original message. Two of the evaluation methods are quite similar, demodulation including decoding for Hamming and LDPC. The signals from the generated data are fed into the models and since the model prediction is always a soft prediction the values

are rounded to be fit to either low or high. The rounded predictions are then compared to the original message to evaluate the performance, the metrics of which are described below in section 4.3.1.

The two remaining proposed models where the neural networks only perform demodulation still require decoding to be evaluated. Both decoders expect to receive LLR values and because the model outputs a likelihood that each bit is high, $P(\mathrm{h} = 1)$, we must first convert these predictions to LLRs. This is done by using each prediction to calculate the corresponding LLR value, as seen in Eq (13).

$$\mathrm{llr}_i = \log_{10}(\frac{1 - (P(\mathrm{h}_i = 1))}{P(\mathrm{h}_i = 1)}) \tag{13}$$

Once converted the values are sent to either an LDPC decoder or a Hamming decoder depending on which encoding is being used. The same Hamming and LDPC parity matrices that were used when encoding are loaded and used to decode the messages. The output from the decoders is compared to the target message to evaluate the model's performance according to the metrics described in section 4.3.1. For each test, 1000 iterations per SNR and per α were made in order to lower result bias.

Initially, the criteria for a functioning system was to consider a BER of 10^{-4} to be sufficient for use. A system would be considered better at a certain α if it passed this criterion at a lower SNR than another. However, during the preliminary testing, it was discovered that achieving this criterion in early development was unlikely or even impossible due to the poor performance of the models. Therefore, each curve was inspected in regards to each other and not only evaluated after how quickly the model was able to achieve a BER of 10^{-4}. As the goal of the project is to create an adaptive demodulator or adaptive demodulator and decoder it is important that the performance on one side of the α spectrum not be neglected in order for the performance on the other to excel. To ensure that this is not the case the entire 3D graphs, depicting the performance for each individual α on every SNR, had to be inspected and evaluated against each other to determine whether an improvement had been made with a certain change.

4.3.1 Performance Metrics

Bit error ratio (BER) is the ratio at which errors occur within a message. Given a message of length l where e errors have occurred the error rate can be calculated as $r = \frac{e}{l}$. The bit error ratio average is of importance when measuring the performance of a system. The average ratio can be calculated as the sum of all error rates for each message divided by the number of messages m, see Eq (14).

$$\text{BER} = \frac{\sum_{i=1}^{m} r_i}{m} \tag{14}$$

4.3.2 Interpreting Graphs

It is common practice when evaluating a system to present a BER depending on the SNR. This is because communication systems are often evaluated over different SNRs and it is important to see the performance of the system depending on this ratio. Due to this project containing another variable which alters the signal, namely the α parameter in the SaS distribution mentioned in section 3.1.3, the models must be evaluated in regards to BER with alternating α value and SNR. In addition to the three dimensional (3D) graph, a two-dimensional (2D) graph will be created in hopes of making the comparison of different model layouts simpler to convey. This 2D graph will contain the mean of the performance of the models for $1.0 \leq \alpha \leq 2.0$, the performance of Gaussian noise as well as the performance when $\alpha = 1.5$.

4.3.3 Final Comparison Scenario

The solutions utilizing this project's proposed models will be compared to five different techniques. These are a residual convolution network [50], a dense neural network [41] and the standard demodulator from the MathWorks communication toolbox, where the standard demodulator is provided with either the exact noise distribution, an estimate of the noise distribution based on the noise applied to the signal or the Gaussian distribution.

As this project proposed solution is a neural network-based solution, it stands to reason that it is also compared to other neural network-based solutions. These models have to be modified slightly to be used in this scenario, see section 5, but the architectural design and flow of the models are the same. The models compared against are a dense neural network and a residual convolutional network as they appeared to perform the best out of those researched in section 5. In regards to data trained on, SNR range, and

α range these models will be using the same parameters as this project's proposed models. In addition, the solution will be compared to a Gaussian standard demodulator and the Genie demodulator where in one scenario Genie is given the exact noise distribution and in another, it is given an estimate of the distribution. The difference between these cases is the demodulator's understanding of the noise distribution. Given the exact distribution, which Genie is given, the demodulator has better conditions to demodulate the message. This follows that the Gaussian standard demodulator, which assumes AWGN noise, will have worse conditions for demodulation when α is varied.

It is anticipated that methods based on neural networks will outperform those that are not based on neural networks in terms of demodulation and decoding time. Therefore, the BER will be employed as an important metric to compare the methods. In regards to the final evaluation and comparison with existing techniques, for demodulation or demodulation and decoding, it is decided to evaluate the performance of each method using Gaussian noise and SaS noise where the α value has been uniformly randomized between the values one and two. This will create a similar graph to the 2D graph used when comparing different models, excluding the curve displaying performance on strict $a = 1.5$. This is done because, for some of the existing techniques, it is not feasible to evaluate all α values separately as well as SNR, as this would take too long. The iteration count was increased to 10000.

5 Model Implementation

The process of developing a model architecture and tuning its hyperparameters to improve performance with respect to BER will be described in the following.

Firstly the objective of the model had to be decided. Since it already had been decided that the model should receive IQ data as input the next decision would be what the output should be. Due to the curse of dimensionality, see section 3.2.7, it was attempted to create a model where each output represented the probability that a bit is high. By doing so it is possible to decrease the number of output nodes from n to 2^n where n is the length of the codeword or message, see *Fig.* 11. This in turn infers that the model is expected to approximate the code structure.

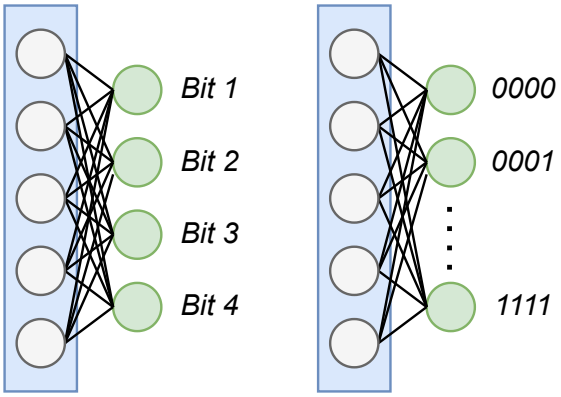


Figure 11 Output layer of two neural networks the one to the left has an output vector of length n and the one to the right an output vector of length 2^n.

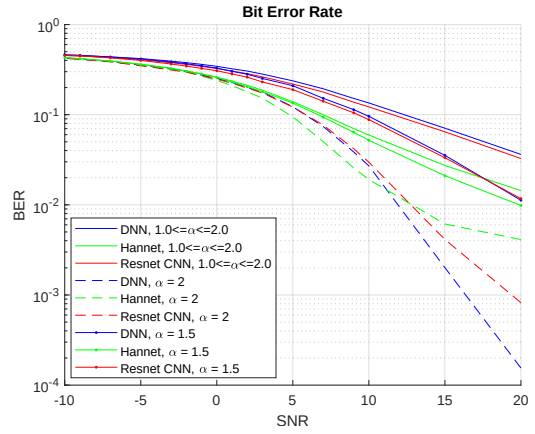
The next step was to test the models mentioned in section 3.4 in this new scenario. Even though the implementations were similar, meaning they accomplished demodulation or decoding, their input data did not necessarily match that of this project. The models, therefore, were modified to fit the parameters of this scenario. As the models were originally tested on a different scenario we must be aware that the same performance is not guaranteed for the scenario set before them now. Once the models had been recreated they were tested to figure out what neural network features could be considered useful for this task. The most prominent model architectures, and their characteristics, were then used as inspiration to create this project's proposed models. These models were iterated by adding or removing layers, dropout, tweaking layer parameters and adding skip connections to improve them.

The final architectural solution, shown in section 6, differs depending on how the message is encoded, through Hamming or LDPC, resulting in two different architectures,

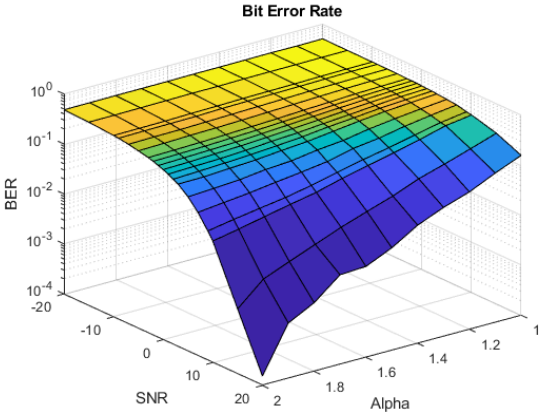
Hannet and Lannet. As can be seen in *Figs.* 12 and 13 it outperforms the replicas of the best-performing neural networks gathered previously during the initial research. Once the architecture was decided the possible improvement methods described in section 3.3 were evaluated.

In the case of LDPC encoding, the 2D graph in *Fig.* 13 would suffice to evaluate the performance of this project's proposed LDPC model, Lannet, compared to the other models. However, to understand the choice in Hamming model architecture the 3D graphs in *Fig.* 12 must be examined. The results show that the proposed Hamming model, Hannet, outperforms the other models on average over all α values. Therefore, Hannet was chosen as the model architecture to proceed with.

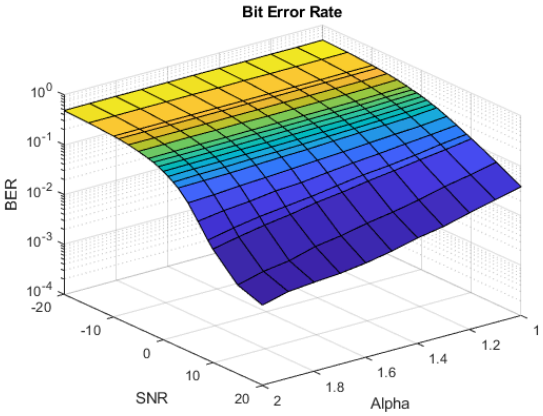
As the model architectures had now been decided the next step was to change various aspects of training, such as tuning training data and altering hyperparameters. Numerous tests were performed to determine which training solution made the models perform best. Section 3.3 mentions a variety of different techniques for improving a model's performance. These are all, with the exclusion of dropout and activation functions which already had been tested, explored in the tests below.



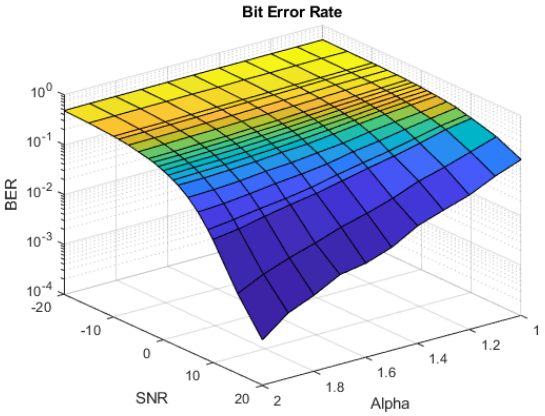
(a) Best model architectures



(b) DNN



(c) Hannet



(d) ResNet CNN

Figure 12 Bit Error Rates of Best Model Architectures for Hamming Encoding

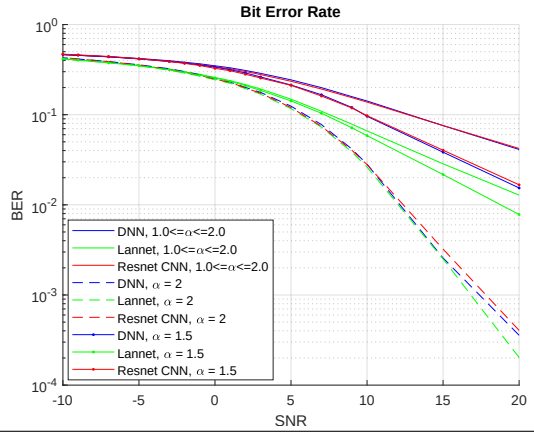
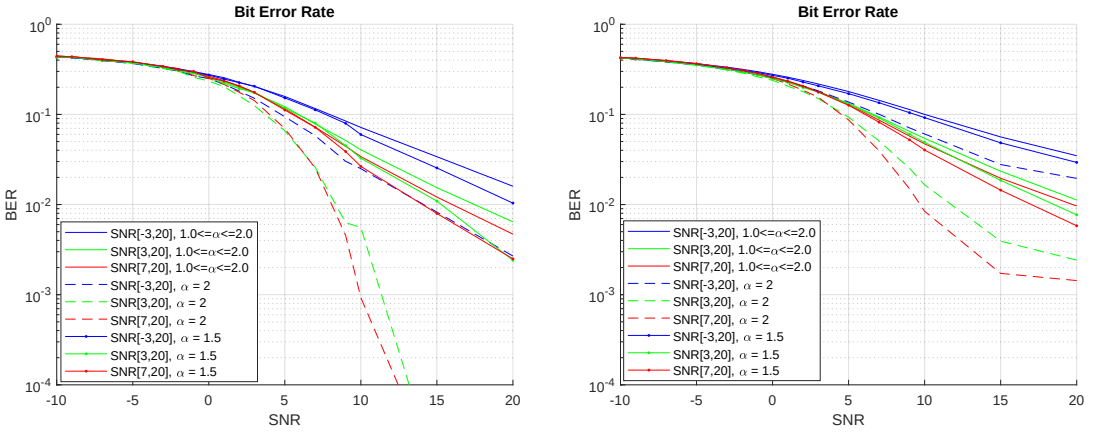


Figure 13 Bit error rates of best model architectures for LDPC encoding.

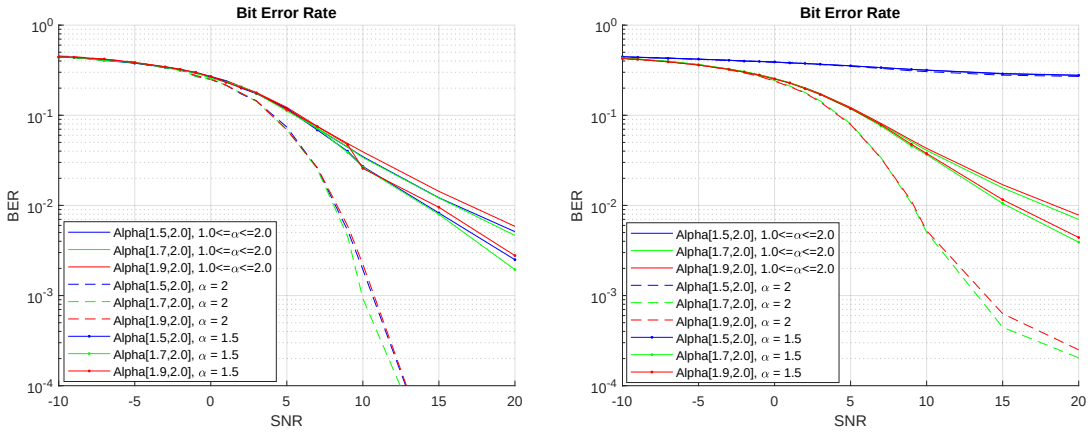
5.1 Decision of SNR and Alpha Spectrum

As mentioned in section 3.3.11, too noisy input data may decrease the performance of the model. Therefore it is important to limit the input spectrum in regards to SNR and perhaps even α, as α dictates how impulsive the noise is. When testing, it was found that for both Hannet and Lannet a range of 7-20dB in SNR performed the best. As for α, the range differs where Hannet performed the best with the range [1.7, 2.0] and Lannet with [1.8, 2.0], see *Figs.* 14, 15, 16 and 17. Even though the data can be deemed ”unseen” by the model it still outperformed a model trained on only $\alpha = 1.5$ or a range of values close to 1.5 in regards to its accuracy on that α value.



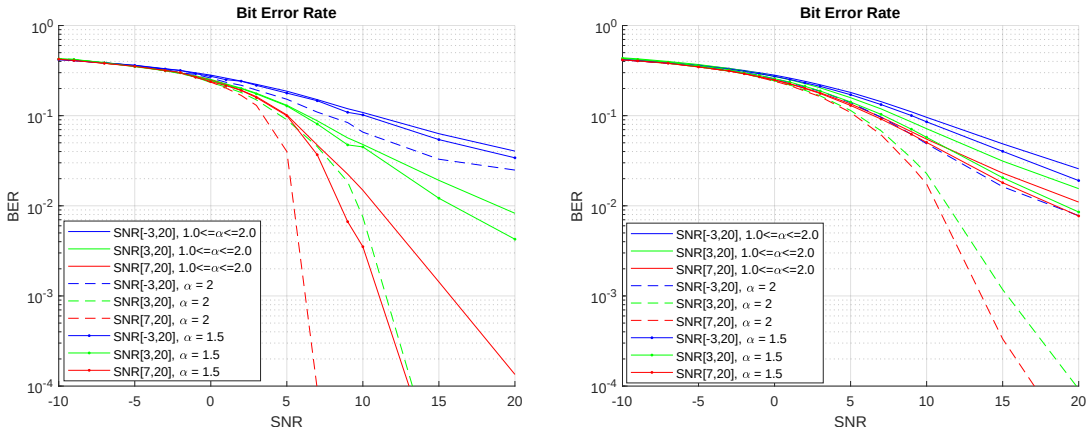
(a) **(b)**

Figure 14 Bit error rates of Hannet for choice of SNR training range. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.



(a) **(b)**

Figure 15 Bit error rates of Hannet for choice of α training range. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.



(a) **(b)**

Figure 16 Bit error rates of Lannet for choice of SNR training range. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

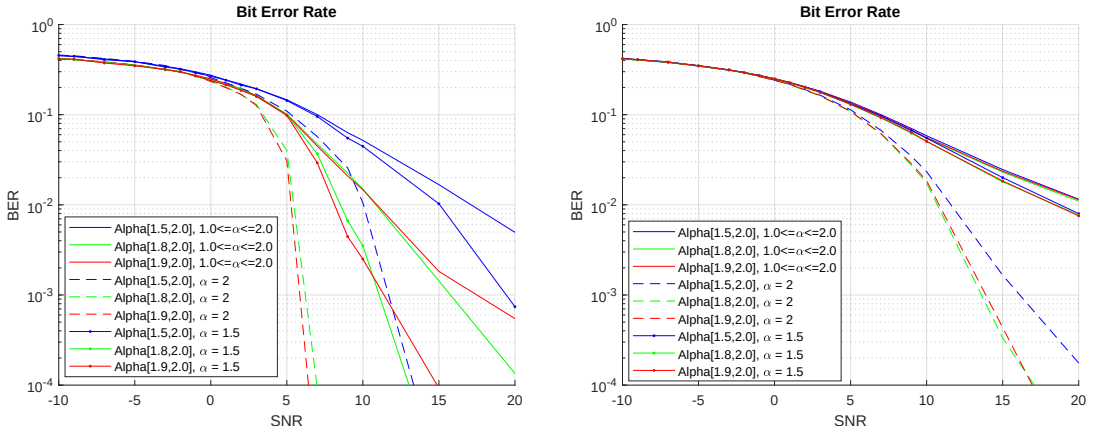
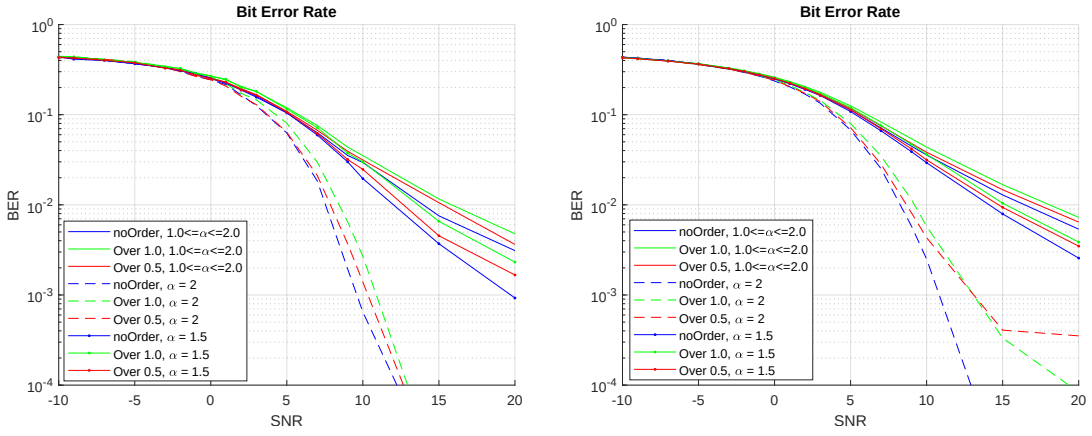


Figure 17 Bit error rates of Lannet for choice of α training range. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

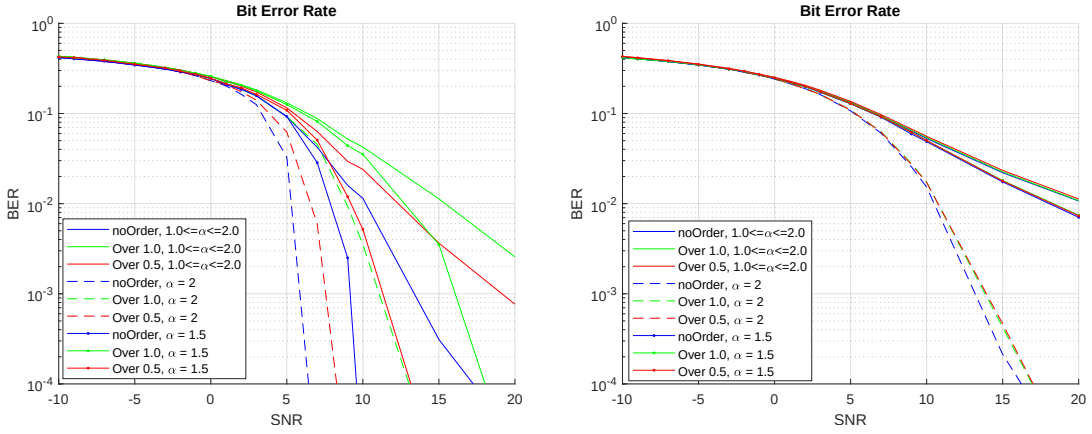
5.2 Results of Ordering the Training on the Data-set

By ordering the SNR when training we can find performance improvements. What is more curious is if the same can be said for ordering the α value. Based on what was mentioned in section 3.3.12, an attempt was made to order the α value and SNR combinations as well. The orderings were calculated by generating 10^7 samples of *SaS*-modelled noise for each combination of SNR and α to estimate it's impact on the signal. The orderings were then realized by some definition disruptions, meaning how far away from the original constellation the noise would push the symbol sample. The different types were greatest disruption, number of disruptions with a magnitude of 0.5 and above, number of disruptions with a magnitude of 1.0 and above, and the standard deviation of all samples. The results can be seen in *Figs.* 18 and 19 where it was found that no devised ordering schematic had any clear beneficial improvements. The orderings were tested on the range of SNR 7-20dB and $\alpha \in [1.8, 2.0]$ or $\alpha \in [1.7, 2.0]$ based on the previous test.



(a) **(b)**

Figure 18 Bit error rates of Hannet for choice of training order. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding. The orderings are based on the number of disruptions, caused by the noise in each data-set, with a magnitude of over 1.0 and 0.5 respectively.

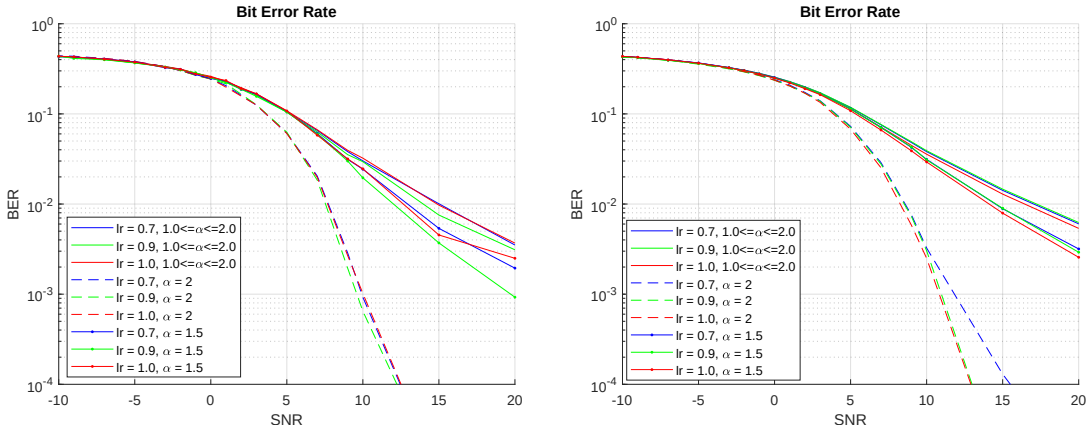


(a) **(b)**

Figure 19 Bit error rates of Lannet for choice of training order. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding. The orderings are based on the number of disruptions, caused by the noise in each data-set, with a magnitude of over 1.0 and 0.5 respectively.

5.3 Outcome of Lowering Learning Rate Between Fittings

On top of the already lowered learning rate done by most optimizers for each epoch, it was also attempted to lower the learning rate between each fitting of the data. However, as can be seen in *Figs.* 20 and 21 changing this had no clear improving effect, except for in the case of Hannet performing only demodulation where weighted decay by a factor of 0.9 seemed to be the best alternative. The learning rates were altered by a weighted decay which essentially decreased the learning rate by a factor of 0.9 or 0.7 between each data fitting.



(a) **(b)**

Figure 20 Bit error rates of Hannet for choice of learning rate decay, where lr is the factor of which the learning rate decays between fittings. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

5.4 Effect of Fitting the Output to the Sigmoid Function

Section 3.3.1 mentions that scaling the target data for a neural network is another way to improve performance. Both models utilize the same output function, namely the sigmoid function. The sigmoid function can output a value in the $[0, 1]$ range, and reaches these limits as x approaches infinity and negative infinity respectively. However, as reaching a weight of infinity may create mathematical issues when training, it is reasonable to lower the output target from the desired 0 and 1.0 to something more reasonably achievable. The values are therefore altered to be 0.05 and 0.95 in order to facilitate this for the sigmoid function. As can be seen in *Fig.* 22, Hannet performing demodulation and decoding has decreased performance when scaling the output whilst when performing only demodulating the performance is increased. On the contrary, Lannet

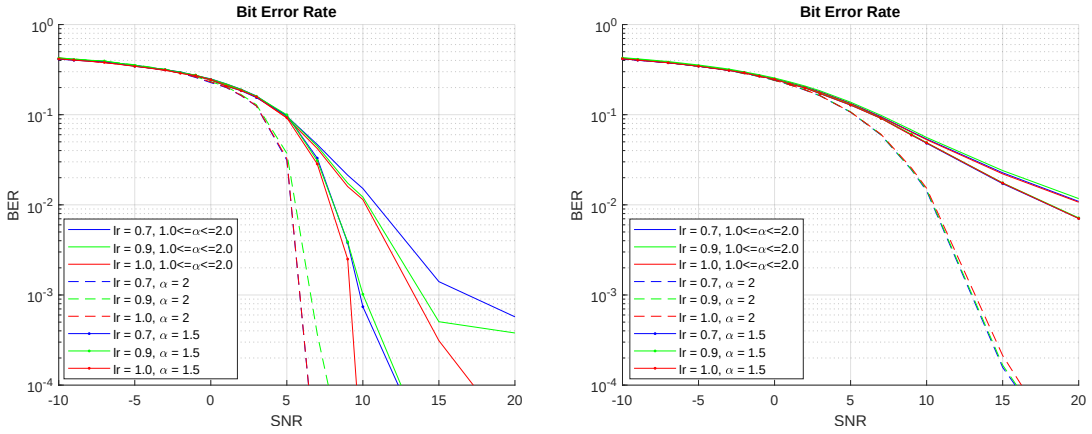
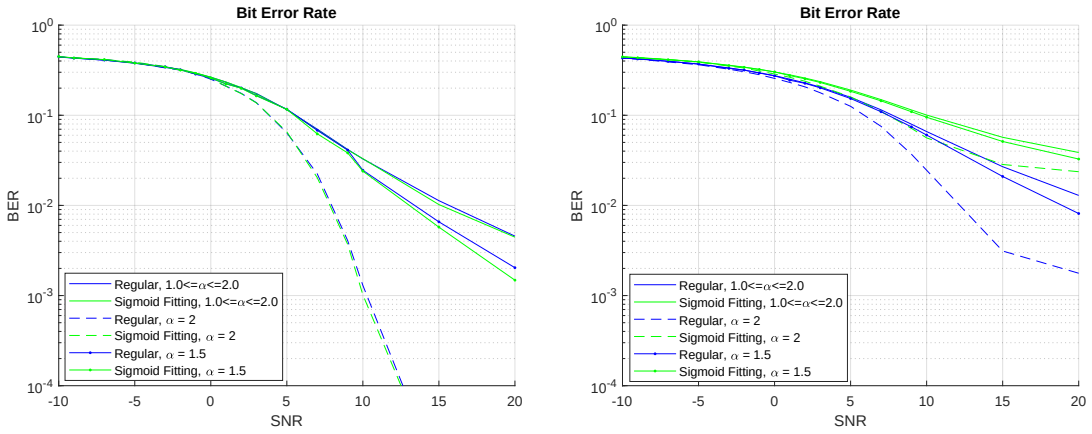


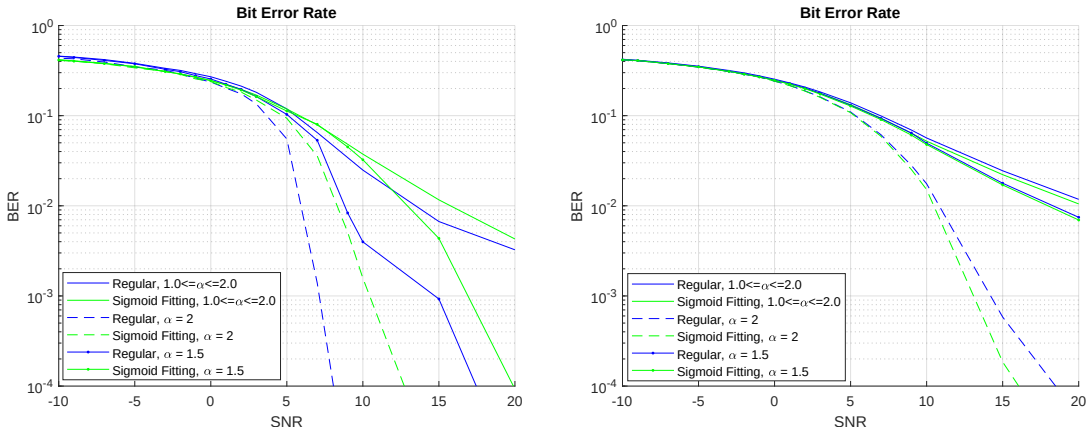
Figure 21 Bit error rates of Lannet for choice of learning rate decay, where lr is the factor of which the learning rate decays between fittings. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

performing both demodulation and decoding in *Fig.* 23 has improved performance when scaling the output whilst when only performing demodulation it has drastically lower performance.



(a) **(b)**

Figure 22 Bit error rates of Hannet for choice of sigmoid fitting. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.



(a) **(b)**

Figure 23 Bit error rates of Lannet for choice of sigmoid fitting. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

5.5 Choice of Loss Function and Optimizer

Originally binary cross entropy (BCE) was used during the previous tests as this is considered a well-performing loss function in a multi-label classification problem. However, since no scientific evidence was found to support this fact a test was conducted to evaluate a larger selection of loss functions. Most functions provided by Keras [2] were able to provide a suitable fit, often performing on par with BCE. However, for three of the four models BCE was beaten by a slight margin, as seen in *Figs.* 24 and 25. The mean squared logarithmic error (MSLE), for Hannet and Lannet performing only demodulation, and the Huber loss function, for Hannet also performing decoding, gave the best results.

The performance of the optimizers differed depending on which encoding was used. The Adam optimizer performed best when the message was encoded with LDPC encoding, see *Fig.* 27. However, for the Hamming encoded messages the RMSProp optimizer performed the best for Hannet when demodulating and decoding whilst Adam performed best for Hannet when only demodulating, see *Fig.* 26.

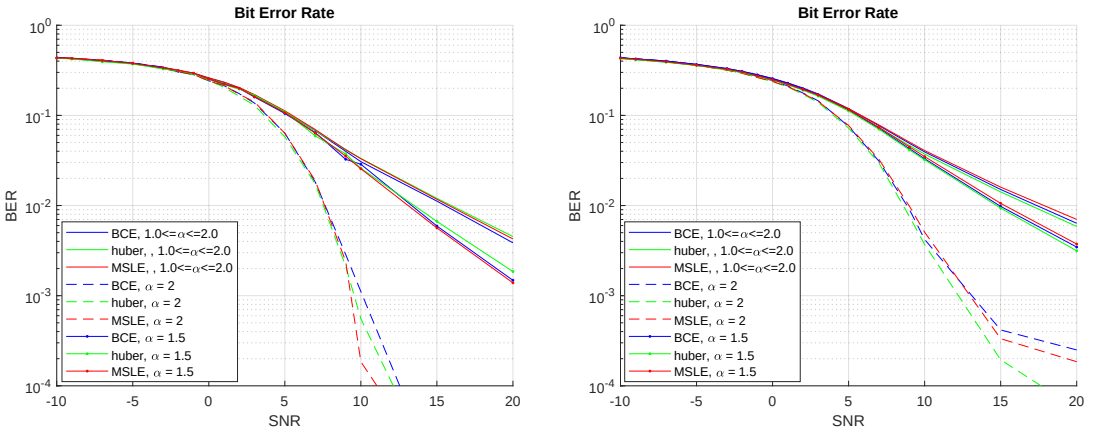
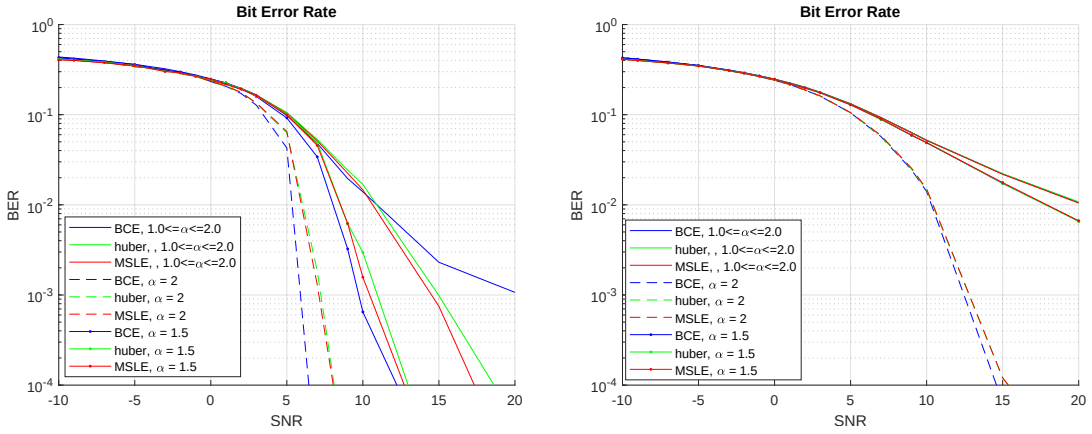
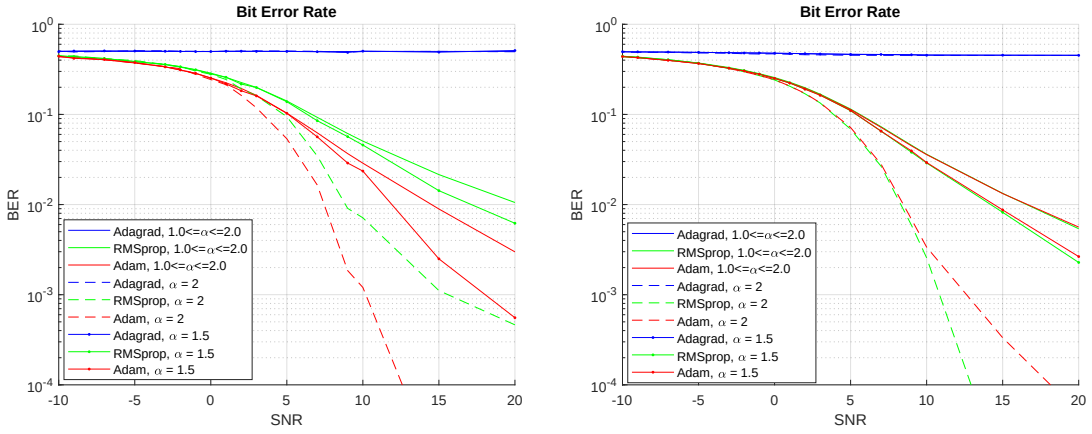


Figure 24 Bit error rates of Hannet for choice of loss function. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.



(a) **(b)**

Figure 25 Bit error rates of Lannet for choice of loss function. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.



(a) **(b)**

Figure 26 Bit error rates of Hannet for the optimizer choice. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

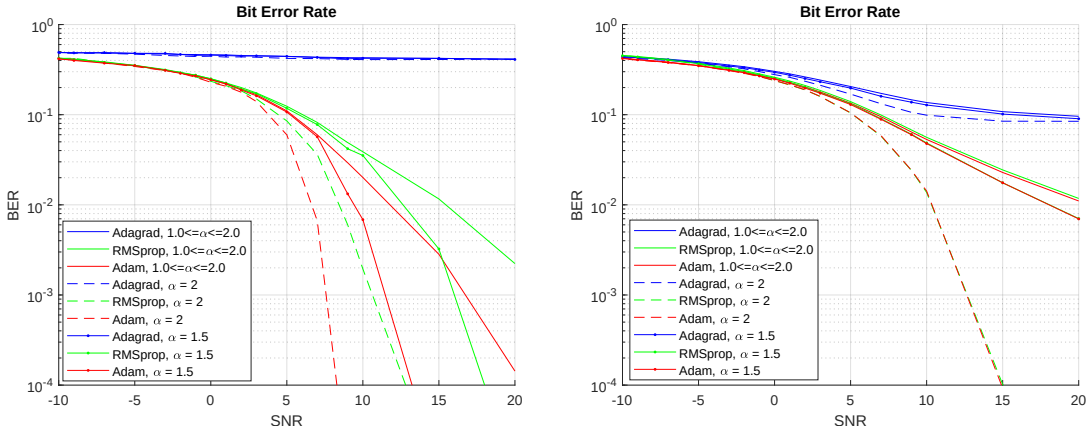


Figure 27 Bit error rates of Lannet for the optimizer choice. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

5.6 Normalization Results

The normalization used is provided by the Scipy library and is referred to as Z-score normalization. The general idea is to update each value in the set such that $V_{\text{new}} = \frac{(V-\mu)}{\sigma}$, where V is a given value in the set, μ is the mean and σ the variance. By doing so the goal is to give the dataset a mean of 0 and a variance of 1, which may ease the learning process for the models. However, as can be seen in *Figs.* 28 and 29, it had no greater effect on higher α values and decreased the performance over all other values for both Lannet and Hannet. The same test was also done using L1 and L2 normalization which ultimately gave the same results.

5.7 Effect of Regularizing the Layers

As for regularization choice, this method had already been iterated upon on a layer-specific level during the architecture design phase without giving any beneficial results. Nevertheless, a test was constructed where each layer of the model would be assigned the same regularization method. Because regularization expects the mean of each feature to be 0 it was decided to be tested in combination with normalized data. This safety precaution is rather redundant as the data should already have a mean of 0, therefore a test was also conducted without normalizing the data and the results were found to be similar to that of the test results presented in *Figs.* 28 and 29. It was found that utilizing regularization caused the model to more often get stuck in a local minimum, as can be seen in *Fig.* 29. If the model did not get stuck in a local minimum it performed worse or

on par with a model which did not have regularization added to it, and therefore it was decided to not utilize this neural network feature.

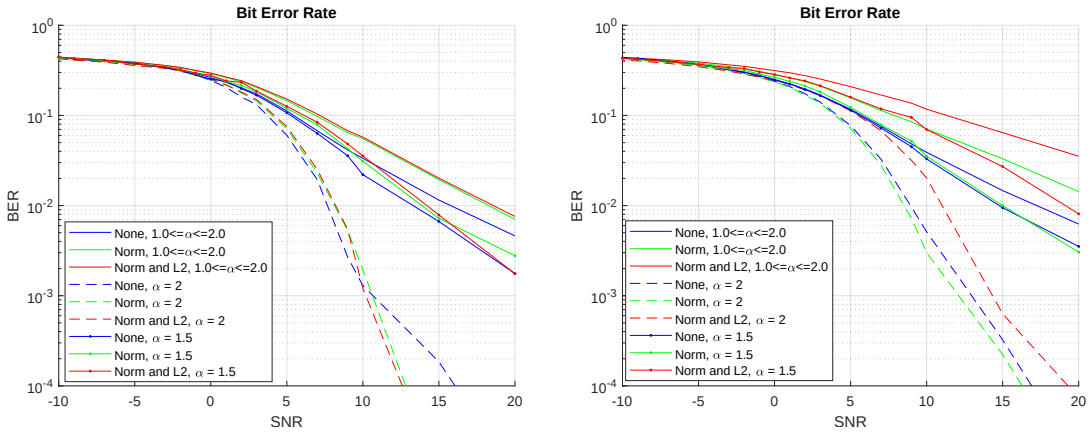


Figure 28 Bit error rates of Hannet for choice of normalization and regularization. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

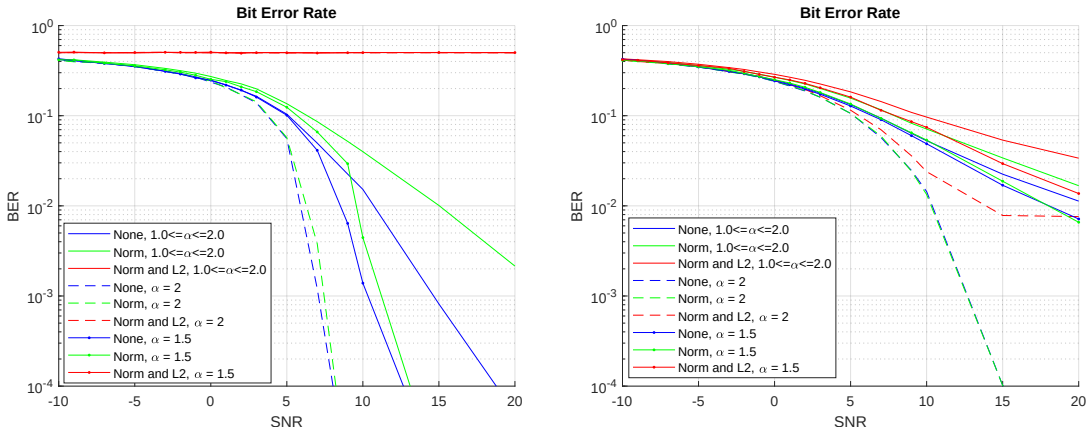
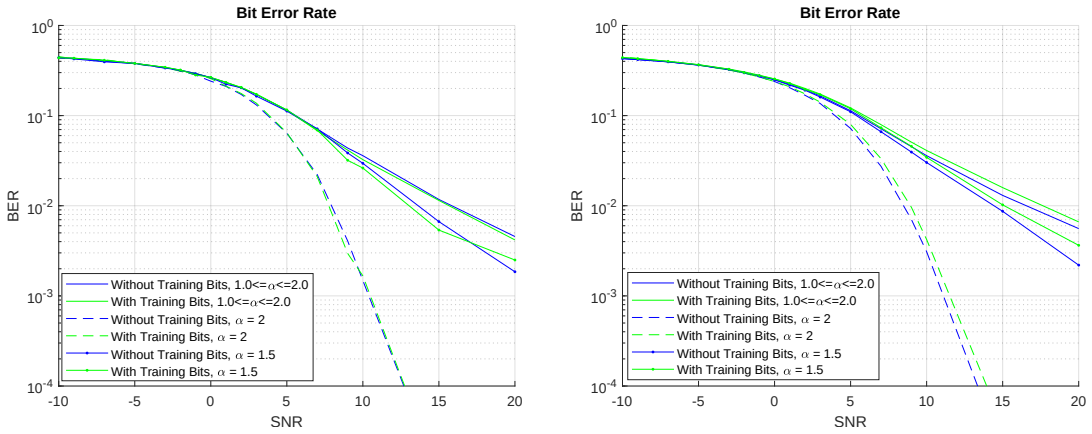


Figure 29 Bit error rates of Lannet for choice of normalization and regularization. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

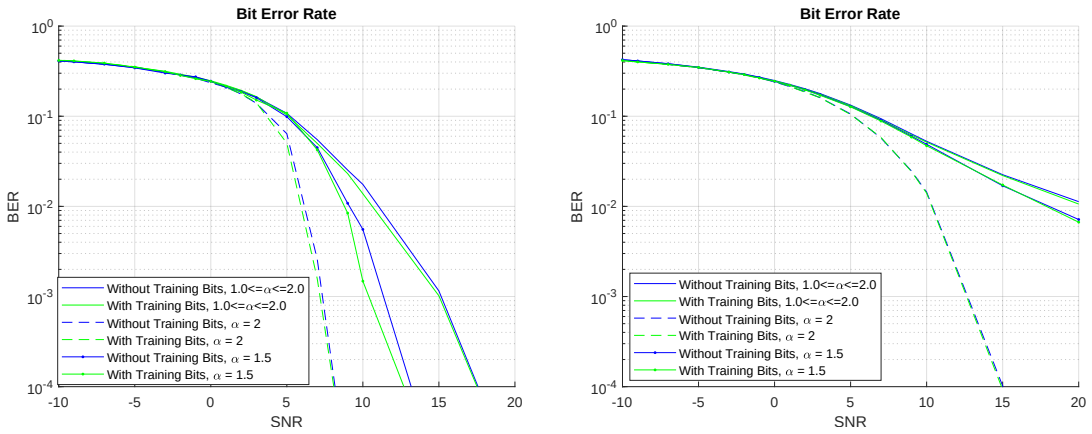
5.8 Outcome of Adding Training Bits

In the comparison in *Figs.* 30 and 31 it can be seen that the training bits increase the performance of the higher valued α but in return, since the mean does not change, the lower α values performance is decreased. The length of the training bit sequence was 20% of the encoded message length. Adding training bits gave no concrete improvements to the model, however, did inherently add more overhead data to the message payload and in turn decreased the throughput of the system. Therefore, it was decided to not add training bits to the data.



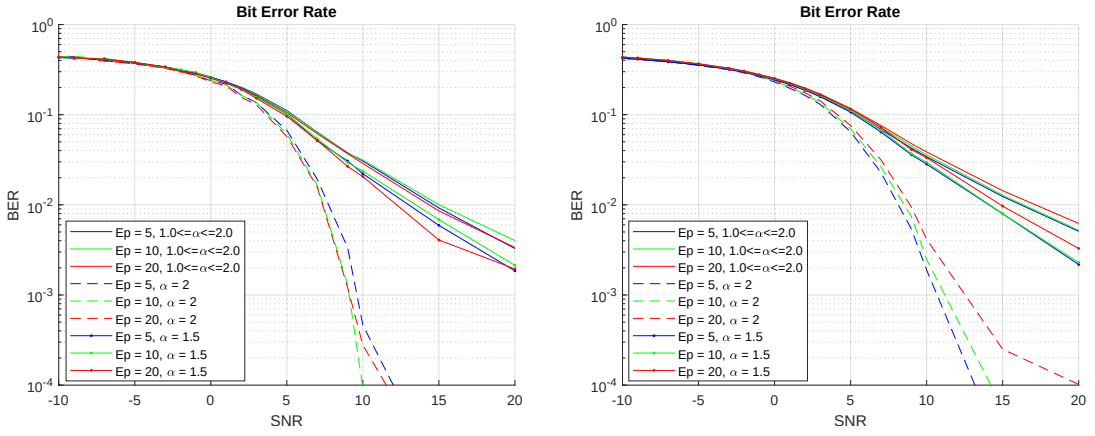
(a) **(b)**

Figure 30 Bit error rates of Hannet for choice of training bits. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.



(a) **(b)**

Figure 31 Bit error rates of Lannet for choice of training bits. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.



(a) **(b)**

Figure 32 Bit error rates of Hannet for choice of epochs, where Ep is the number of epochs trained on each data-set. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

5.9 Effect of Epochs, Batch Size and Data-set Size

Three different epoch counts were examined which can be seen in *Figs.* 32 and 33. As the amount of data is very high the epoch values are also quite low since adding more would only result in overfitting. For Hannet, it was found that 20 epochs for demodulating and five epochs for demodulating and decoding gave the best results. As for Lannet, ten epochs seemed to give the best performance regardless of whether the model performed only demodulation or also decoding. However, in all cases, it would seem that increasing the epoch count over ten has little to no effect on the model, as no clear trend is found, and they all converge to the same local minimum regardless.

Lowering the batch size drastically increases the training time of the models, because the number of times for which the model weights are updated is also increased. For most models, it would seem a lower batch size value would increase performance all around, see *Figs.* 34 and 35. However, for Lannet performing only demodulation, the best batch size is 32. Even though the tests show a trend of better performance when lowering the batch size it can also be argued, due to the results of the different iterations of the models being so similar, that any of the three batch sizes would be sufficient.

Increasing the training data set size also increases the performance of the model, albeit very slightly in the case of the Lannet performing demodulation and decoding, as can be seen in *Figs.* 36 and 37. The measurement shown is the amount of data for each SNR that the model is trained on.

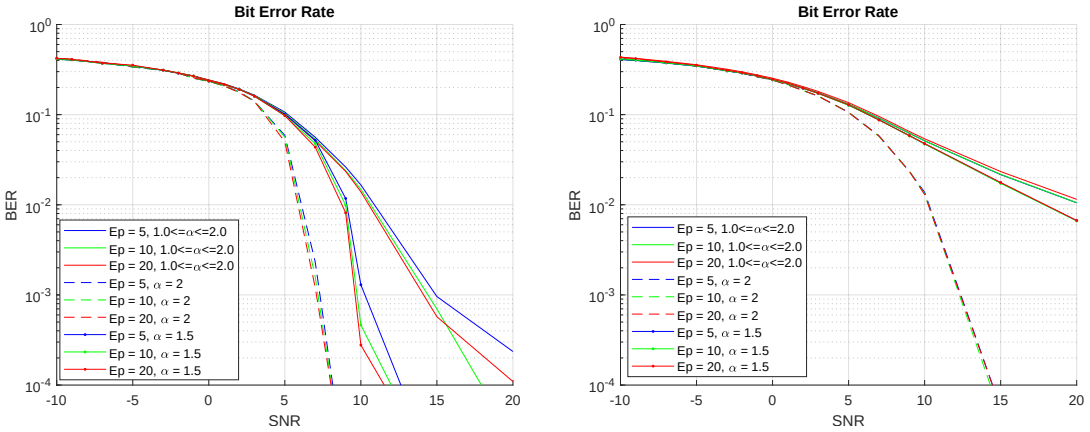


Figure 33 Bit error rates of Lannet for choice of epochs, where Ep is the number of epochs trained on each data-set. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

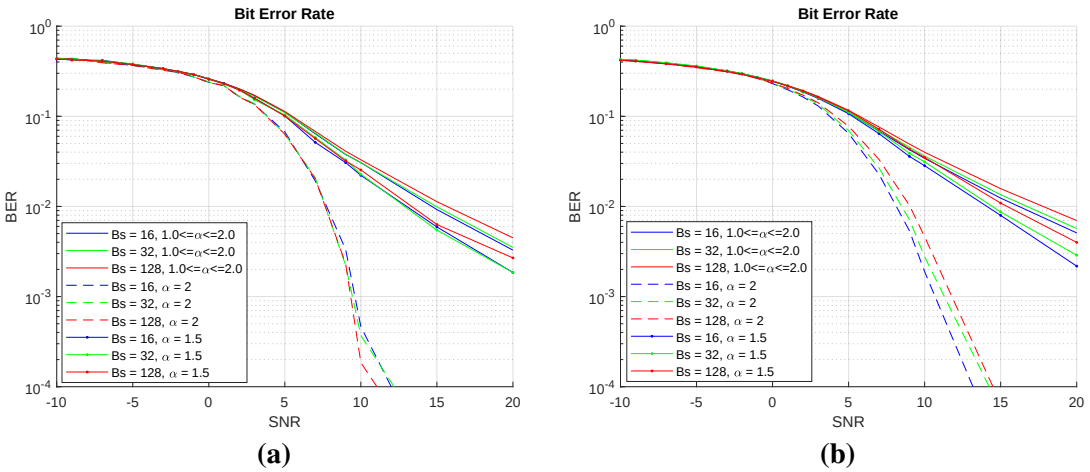
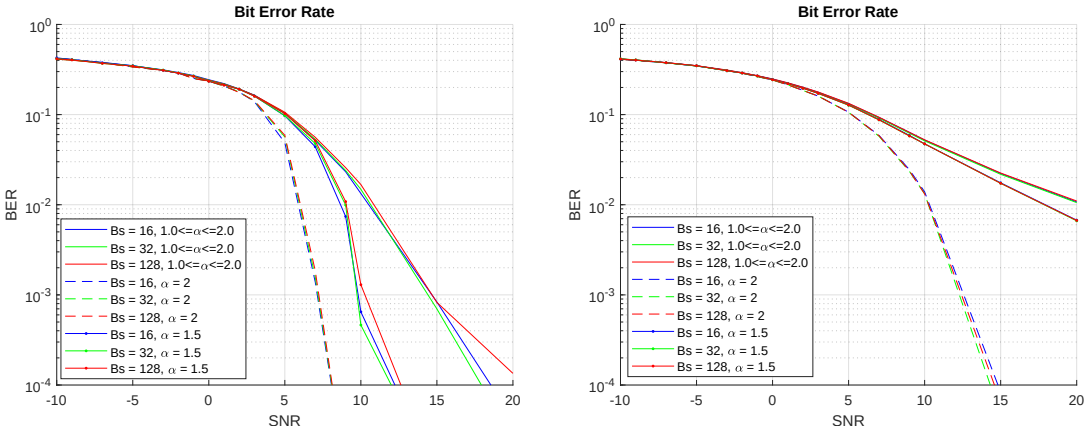
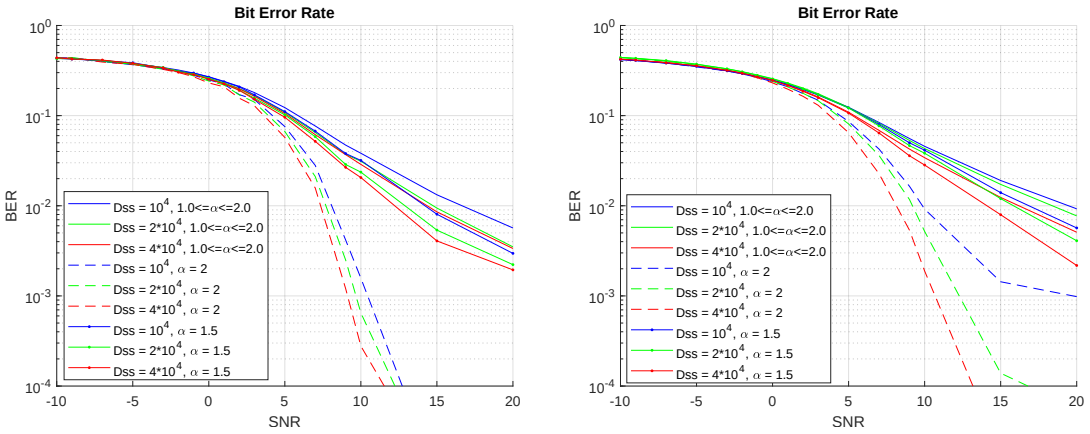


Figure 34 Bit error rates of Hannet for choice of batch size. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.



(a) **(b)**

Figure 35 Bit error rates of Lannet for choice of batch size. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.



(a) **(b)**

Figure 36 Bit error rates of Hannet for choice of data set size, where dss is the size of each data-set for each SNR trained on. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

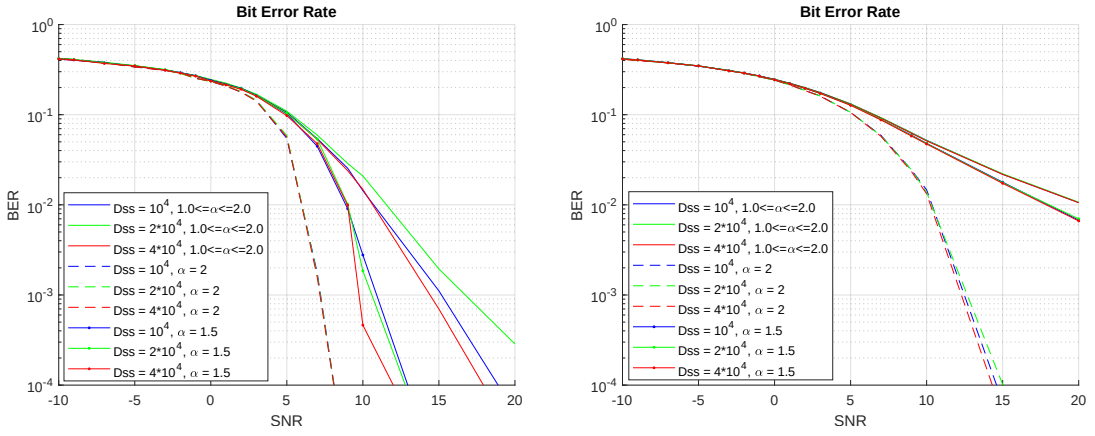


Figure 37 Bit error rates of Lannet for choice of data set size, where dss is the size of each data-set for each SNR trained on. Neural network solutions in (a) perform only demodulation whereas in (b) they also perform decoding.

6 Model Architecture

Depending on whether the neural networks demodulate and exclude or include decoding, as well as which encoding is used, the design of the models will differ. The largest difference is the architectural design of the models which changes depending on which type of encoding is used. In the following sections, the different types of model architectures will be outlined as well as which hyperparameters are used depending on if the model is to perform only demodulation or also decoding. All the parameters are set in a particular manner in accordance with the tests laid out in section 5, unless otherwise specified. By changing the output and input of the models the hidden layers will be dynamically altered, however, there is no guarantee that the performance will be the same if the models are trained with a different size than that used in this thesis. The models were trained on datasets with different SNRs, starting with the highest SNRs and lowering it for each fitting. Table 2 lists the parameters used when training the models.

6.1 Hannet Model

This project proposes a model architecture for Hamming encoding that consists of four blocks, the first and third being identical. The first block consists of an upsampling layer followed by two convolutional layers, seen as the purple block in *Fig.* 38a, and the second block consists of a skip connection preceding two convolutional layers, seen as the blue block in *Fig.* 38a. The second convolution in the first and third blocks is used to

Table 2: Proposed model parameters. Dataset size refers to the size of each dataset per SNR trained on.

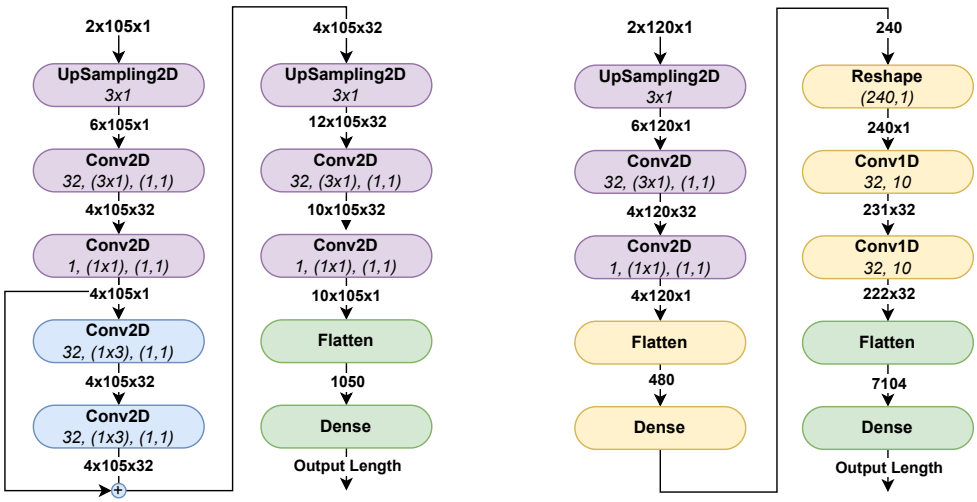
	Demodulator		Decoder	
Encoding	Hamming	LDPC	Hamming	LDPC
Optimizer	Adam	Adam	RMSProp	Adam
Dataset size	40 000	40 000	40 000	40 000
Batch size	16	32	16	16
Epochs	20	10	5	10
Training Bits	No	No	No	No
Normalization	No	No	No	No
Regularization	No	No	No	No
Output fitting	Yes	No	No	Yes
Learning rate decay	0.9	1.0	1.0	1.0
SNR range	7-20dB	7-20dB	7-20dB	7-20dB
Alpha range	[1.7, 2.0]	[1.8, 2.0]	[1.7, 2.0]	[1.8, 2.0]
Loss function	MSLE	MSLE	Huberloss	BCE

summarize the features and make for a faster learning process as opposed to replacing it with a dense layer which would create a lot more weights. Finally, the fourth block, seen as the green block in *Fig.* 38a, is the output which flattens the data and uses a final dense layer with a Sigmoid activation to manifest the multi-labeled classification. After each convolution follows an activation layer which uses the function rectified linear unit (ReLU) as well as a batch normalization layer. The output length differs depending on whether the model is performing demodulation or also decoding. If the model is performing demodulation the output will be of 210 nodes long, however, if decoding is included then the length will be 120 nodes. The codeword length of the Hamming model remains seven bits long although, each message sequence is 210 bits in length meaning that for each transmission 30 Hamming codewords are sent.

6.2 Lannet Model

This project proposes a model architecture for LDPC encoding that consists of three blocks. The first block is the same as the one used in the Hamming model, seen as the purple block in *Fig.* 38b, whilst the second block flattens the data then follows with a dense layer, a reshape and two convolutional layers, seen as the yellow block in *Fig.* 38b. Finally, the model flattens the data yet again after the convolutions and outputs the data using a final dense layer with Sigmoid activation, seen as the green block in *Fig.* 38b. After each 2D Convolution (convolutions in purple blocks) follows an activation layer

which uses the ReLU function as well as a batch normalization layer, precisely as that of the Hamming model. The second block utilizes no specific activation function after each layer and thus defaults to linear activation. The output of the model is 240 nodes when the model performs only demodulation and 120 nodes when the model also performs decoding. The codeword length for the LDPC model is 240 bits long, with a code rate of $\frac{1}{2}$.



(a) Hamming model **(b)** LDPC model

Figure 38 Schematics for the two proposed model architectures designed for different encodings.

7 Results and Discussion

In this section, the performance of this project's proposed model solutions, described in section 6, will be presented and discussed. They are compared to standardized techniques used today such as the Gaussian and Genie demodulator in combination with a respective decoder depending on the encoding type. Another comparison made is between other well-performing model architectures and that of the one suggested in this paper. In both *Fig.* 39 and *Fig.* 40 the left-hand graph represents the model's performance when Gaussian noise is added to the signal whereas the right-hand graph represented the model's performance when SaS modelled noise has been added to the signal. The Genie demodulator is tested with both an estimated α as well as the known α. All the tests on the different techniques were made using the same dataset.

The Genie demodulator with known α is hard to see in the graphs displaying BER

curves for $\alpha = 2$, this is because when Genie is presented with this α value it defaults to the Gauss solution and is thus the same solution in that particular scenario.

When the noise is Gaussian and demodulation including decoding is performed through neural networks, Hannet and Lannet outperform the other models, however, when only demodulation is done through neural networks they are outperformed by the other models tested. The Gaussian demodulator, and in extension the Genie demodulator with known α as it defualts to the Gaussian demodulator when the noise is Guassian, outperforms all models. However, when the Genie demodulator is given an estimation of the noise distribution it, in the case of LDPC encoding, is outperformed by one of the neural network-based solutions, namely the residual convolution network.

When performing only demodulation, if the α value is varied, Hannet and Lannet yield considerably better results than every solution except for Genie with known or estimated α. However, when Hannet and Lannet perform decoding in addition to demodulation, their performance is slightly inferior in all cases. In the case of Hamming encoding, Hannet exhibits better performance than all solutions except for Genie with known and estimated α, while for LDPC encoding, Lannet is unable to outperform the Gaussian demodulator.

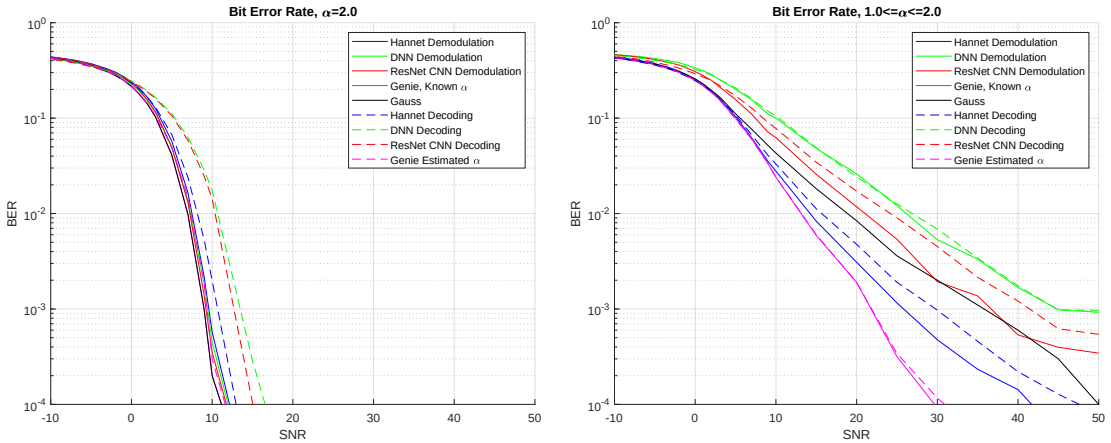


Figure 39 Comparison of solution using the proposed model, Hannet, and existing techniques using Hamming encoding. In (a) the performance on Gaussian modelled noise is compared whereas in (b) the noise is instead modelled with the SaS distribution, with $1.0 \leq \alpha \leq 2.0$. Demodulation refers to a model performing only demodulation whilst decoding means that they perform both demodulation and decoding.

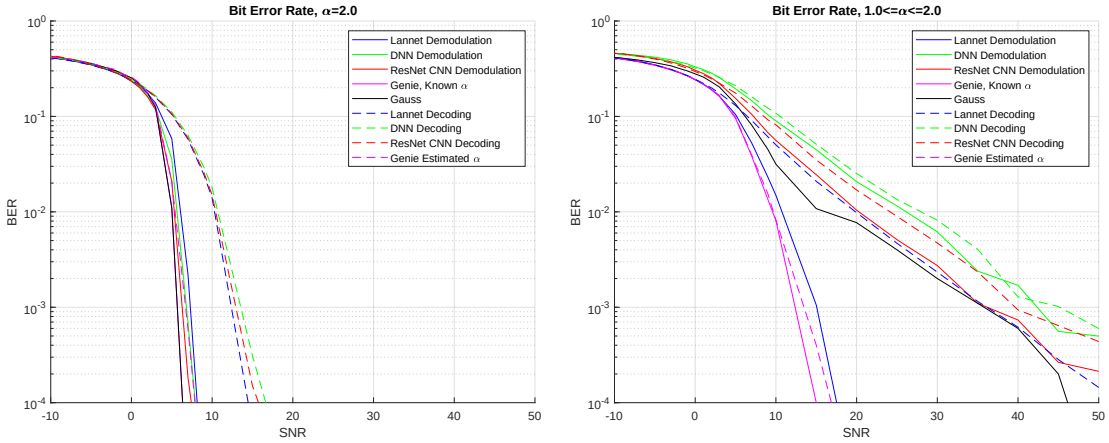


Figure 40 Comparison of solution using the proposed model, Lannet, and existing techniques using LDPC encoding. In (a) the performance on Gaussian modelled noise is compared whereas in (b) the noise is instead modelled with the SaS distribution, with $1.0 \leq \alpha \leq 2.0$. Demodulation refers to a model performing only demodulation whilst decoding means that they perform both demodulation and decoding.

The results show a rather satisfactory performance, where Lannet performing only demodulation on LDPC code seems the most promising solution. It was always expected that this project's proposed models performing demodulation would outperform the Gaussian demodulator when α is varied, as the Gaussian demodulator assumes the noise to be AWGN. However, it is by a larger margin than expected. The models are able to perform almost on par with that of the Gaussian demodulator when the noise is Gaussian, though, it does have a slight edge. Whilst this is the case, when the α value is random Hannet and Lannet outperform the Gaussian demodulator in all cases but one. As for Lannet, it seems to struggle to decipher the LDPC encoding, perhaps because it is more complicated to estimate the function of LDPC encoding as opposed to Hamming encoding.

Another interesting remark is the difference in the performance of the Genie demodulator with known and estimated α. The known α given to the demodulator is the same value used when generating the signal that it is to demodulate, thus it should be optimal. However in practise, the generated noise is finite and because of this, even though it was generated with regard to one distribution, it may be better suited to belong to a different one. This means that when the method instead estimates the α value it may give a better representation than that of the actual α used. This was found after experiencing that the estimated α method in some scenarios outperformed the known α method and therefore the estimated method may perform almost on par with that of the known α. Additionally, it is important to acknowledge that the Genie demodulator requires the α

parameter to be known, which is impossible in practise, whilst the estimator requires a set of training bits to assume the noise distribution. Neither of these are needed for the solution utilizing a neural network demodulator.

It was not expected that this project's proposed models would outperform the Genie demodulator since the Genie demodulator possesses complete knowledge of the noise distribution, which enables it to provide optimal demodulation. The goal was rather to see how close to the performance of the Genie demodulator a model could achieve. The reason is that neural networks, even though they have a larger overhead of training, have a lot shorter execution time than the Genie demodulator.

Whether the performances shown in these results are adequate for use or not is for the user to determine, it highly depends on the system's requirements. If the system can guarantee a higher SNR or the BER requirement allows for higher error rates, then the neural network solution is a valid alternative due to it being faster than standardized methods. Regardless, it seems as if machine learning, specifically a neural network, based approach to demodulating is a good alternative to Gaussian demodulation, due to the BER performance gain in an impulsive environment seen in *Figs.* 39 and 40.

Another topic which should be discussed is an interesting one regarding strictly demodulation models. It was found, but not quite proven, that the models will in some way take into account the encoding when demodulating. This follows that the models can make assumptions based on the provided data that a standard demodulator cannot as they have no concept of encoding. Whether this is a good or bad trait is uncertain as the model's encoding-based assumptions may be false and end up creating an output which is inauspicious when forwarded to the standard decoder. Continuing on the topic of demodulation models, it is worth mentioning that the architectural model design for each encoding type was decided upon prior to that of the training method and hyperparameter tuning. What this in turn means is that Lannet may now be able to demodulate a message encoded with Hamming code better than Hannet as it saw larger improvements than Hannet, even though Lannet seems to make demodulation choices based on the assumption that the message has an LDPC code structure. It might even be the case that Lannet, when instead trained on Hamming code, is better at demodulation including decoding than Hannet on a message encoded with Hamming code.

Even though many of the tests did not grant any particular groundbreaking BER reductions they are still valuable going forward regarding the knowledge of what to focus on when designing a neural network. It is also worth noting that even though the best-performing parameter was used in the final product, and each test moving forward, many of them are considered equal and it may be the case that the parameter had no impact yet the model coincidentally found a better local minimum.

The tests that should be repeated and elaborated, as this research moves forward, are that of the architecture in combination with loss function, SNR and α spectrum and perhaps using a different output, meaning a new output fitting. These were also the most time-intensive tests as designing the architecture of a neural network can branch off in so many directions and honing in on the perfect spectrum of SNR and α may be architecturally specific.

There is also one test which may need revisiting, namely training bits. In this test the training bits length is meant to be part of a synchronization phase in practical use. This means that the length of the training bits segment can be much larger, and should be in order to find meaningful data, without losing too much throughput. However, in the tests performed in this work they were designed to be 20% of the length of the transmitted signals, this results in a segment of roughly 20 symbols of data as grounds for the noise estimation and is perhaps not enough for the model to utilize it.

Recurrent neural networks performed a lot worse than anticipated. However, in practice, recurrent layers may be a boon rather than what is seen in the tests in this work. The reason behind this is that the data tested in this work consists of messages with no inherent structure other than encoding, meaning the messages are randomized bits and lack context. RNNs, due to their ability to keep track of state, may be of great use in a practical setting if the messages sent are well-defined beforehand such that the network may learn the patterns.

8 Conclusions and Future Work

This thesis explores an alternative noise model to that of the AWGN model, namely the SaS-model, which in theory more accurately reflects a channel in today's society, where more electronic devices are prevalent. The SaS-model contains a parameter, α, which dictates how impulsive the noise is, the goal is to create a receiver that performs well over all α. To implement this receiver, two different adaptive neural network-based demodulators, one including decoding and the other excluding decoding, are proposed for both encoding types, resulting in four different models.

Judging from the results it is possible to create an adaptive demodulator with the use of neural networks. As it outperforms the Gaussian demodulator over most α and performs on par with it when $\alpha = 2.0$, it may be considered preferable to the Gaussian demodulator. The impact of encoding choice is significant to the solution where Lannet performing only demodulation on an LDPC encoded message performs better than Hannet doing the same on a Hamming encoded message. As for an adaptive demodulator and decoder, it is not as successful, where the models fails to keep up with the Gaussian demodulator in regards to $\alpha = 2.0$ and only Hannet outperforms the Gaussian demodulator when Hamming encoding has been used. This leads to the assumption that the models are not able to fully learn the decoding pattern of LDPC encoding.

Even though the models are not able to fully learn the decoding algorithms required for LDPC and Hamming encoding there is definitely an attempt being made to learn the encoding. Because the LDPC encoding is more robust than the Hamming encoding it follows that the Genie demodulator, as well as this project's proposed model, performs better when the coding is LDPC. However, since the LDPC encoding is also more complex the reversed effect can be witnessed when examining demodulation including decoding for Hannet and Lannet.

The Genie demodulator with known or estimated α outperforms this project's proposed models. However not by a very large margin in the case of the demodulation models and depending on what BER requirements and SNR expectations a system has the neural network solutions are a viable alternative when searching for a faster algorithm, especially since they require less knowledge of the noise distribution.

Another valuable insight is that tailoring a model towards a specific objective gives tremendous performance improvements and should therefore be a process that the model designer dedicates a great portion of their time towards, should this experiment be elaborated upon.

8.1 Future Work

There are many different branches this work can take if elaborated, below are listed a couple of examples that could be tested in the future.

When examining the difference in the performance of the LDPC encoded results contra the Hamming encoded results it is clear that the robustness of the system correlates to the the encoding chosen. It would be interesting to examine if decreasing the code rate but also increasing the order of modulation, such that the throughput remains the same, could yield performance improvements. Experiments were made where simply duplicating the signal sent gave performance improvements as the impulses were unlikely to disrupt the same bit sequence twice. Increasing the order of modulation would also give valuable insight into how capable a neural network demodulator may be. Another avenue this research can take is a fully adaptive model. This means that the model should be defined for a given amount of encodings and not changed depending on which encoding is used in the scenario. In practice, the encoding and modulation type is often defined beforehand but it would be interesting to see if neural networks can dynamically detect what encoding and modulation type is being used and make predictions accordingly.

During this work, two different encoding techniques were used, LDPC and Hamming. The performance of these differed greatly and therefore exploring other alternative encodings is a great next step, such as Turbo code. To expand on encoding one could also train the demodulators in the same manner as in this work but then also feed those LLR outputs into a new network which is trained to take those LLR outputs as inputs and predict what the original message sent was. This would in essence create two different networks, one demodulator and one decoder.

In regards to the shorter codewords used, namely the Hamming 7.4 codeword, it would be of interest to alter the output such that each output is a realization of the codeword instead of a classification of each bit individually. As the code word is rather short this would only require 2^7 classes and would also transfer the problem from a multilabel classification problem to a simple classification problem.

Sorting the training sequence by SNR was found to aid the model in the learning process, however, no α sorting schematic was found. This process was quite briefly glanced over and tested with some basic scenarios but it might possibly exist some sorting definition which would improve the learning process of the model and increase robustness. Another rather simple elaboration, on training data-set modifications, is to expand on the SNR range. The test used an upper limit of 20dB SNR, however, it is quite possible that increasing this limit may help decrease the BER further.

Certain loss functions are specifically tailored to the Sigmoid function which can be found in the Keras library, namely Sigmoid cross entropy and Sigmoid cross entropy with logits. These were not found until the results were already finalized and as such have not been experimented with, but as the output activation function is a Sigmoid function it would stand to reason to try this loss function.

References

[1] "Keras dropout," https://keras.io/api/layers/regularization_layers/dropout/, accessed: 2022-11-09.

[2] "Keras loss," https://keras.io/api/losses/, accessed: 2022-11-19.

[3] "Keras main page," https://keras.io/, accessed: 2022-01-12.

[4] "Keras optimizers," https://keras.io/api/optimizers/, accessed: 2022-01-12.

[5] "Keras regularizers," https://keras.io/api/layers/regularizers/, accessed: 2022-11-05.

[6] "Matlab alpha stable noise generator," https://uk.mathworks.com/help/stats/stable-distribution.html, accessed: 2022-01-12.

[7] "Matlab hamming encoder / decoder," https://uk.mathworks.com/matlabcentral/fileexchange/42953-soft-hamming-decoder, accessed: 2022-01-12.

[8] "Matlab ldpc encoder / decoder," https://uk.mathworks.com/help/comm/ref/comm.ldpcencoder-system-object.html, accessed: 2022-01-12.

[9] "Matlab python engine," https://uk.mathworks.com/help/matlab/matlab-engine-for-python.html, accessed: 2022-01-12.

[10] "Numpy random sequence generator," https://numpy.org/doc/stable/reference/random/generated/numpy.random.rand.html, accessed: 2022-01-12.

[11] "Scipy alpha stable noise source," https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.levy_stable.html, accessed: 2022-01-12.

[12] "Scipy z-score normalization," https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html, accessed: 2022-01-12.

[13] L. Ahlin, J. Zander, and B. Slimane, in *Principles of Wireless communications*, 2006, p. 11.

[14] B. Christan and T. Griffiths, "Algorithms to live by, the computer science of human decisions." New York: Henry Holt and Company, 2016, ch. 7, pp. 149–168.

[15] L. Clavier, T. Pedersen, I. Larrad, M. Lauridsen, and M. Egan, "Experimental evidence for heavy tailed interference in the iot," *IEEE Communications Letters*, vol. 25, no. 3, pp. 692–695, 2021.

[16] C. W. Dawson and R. Wilby, "An artificial neural network approach to rainfall-runoff modelling," *Hydrological Sciences Journal*, pp. 47–65, 1998.

[17] S. Dörner, S. Cammerer, J. Hoydis, and S. ten Brink, "Deep learning based communication over the air," 2018.

[18] A. P. Engelbrecht, *Computer Intelligence, an introduction*, 2nd ed. John Wiley Sons, Ltd, 2007.

[19] L. Fang and L. Wu, "Deep learning detection method for signal demodulation in short range multipath channel," in *2017 IEEE 2nd International Conference on Opto-Electronic Information Processing (ICOIP)*, 2017, pp. 16–20.

[20] R. Fritschek, R. F. Schaefer, and G. Wunder, "Deep learning for channel coding via neural mutual information estimation," 2019.

[21] D. George and E. A. Huerta, "Deep neural networks to enable real-time multimessenger astrophysics," 2018.

[22] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," p. 5, 2008.

[23] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," 2017.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–772, 2016.

[25] Y. Hu, L. Zhao, Z. Yan, A. Kaushik, Y. Hou, and J. Thompson, "Gatednet: Neural network decoding for decoding over impulsive noise channels," *IEEE Communications Letters*, vol. 23, no. 8, pp. 1381–1384, 2019.

[26] K. Hägglund and E. Axell, "Adaptive demodulation in symmetric alpha-stable impulse noise channels," *IEEE Transactions on Vehicular Technology*, 2020.

[27] K. Hägglund and E. Axell, "Adaptive demodulation in symmetric alpha-stable impulse noise channels," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, 2020, pp. 1–5.

[28] A. Irawan, G. Witjaksono, and W. K. Wibowo, "Deep learning for polar codes over flat fading channels," 2019.

[29] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*, 2nd ed., 2021.

[30] M. Kozlenko, I. Lazarovych, V. Tkachuk, and V. Vialkova, "Software demodulation of weak radio signals using convolutional neural network," 2020.

[31] I. Landa, M. Vélez, and A. Arrinda, "Impulsive noise measurements from consumer electronic devices," in *2018 IEEE Conference on Antenna Measurements Applications (CAMA)*, 2018, pp. 1–4.

[32] X. Li, Z. Chen, and S. Wang, "An approximate representation of heavy-tailed noise: Bi-parameter cauchy-gaussian mixture model," in *2008 9th International Conference on Signal Processing*, 2008, pp. 76–79.

[33] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, *Machine Learning, A First Course for Engineers and Scientists*. Cambridge: Cambridge University Press, 2022.

[34] T. K. Moon, *Error Correction Coding, Mathematical Methods and Algorithms*, 2nd ed. John Wiley Sons, Ltd, 2005.

[35] C. L. Niklas and M. Shao, in *Signal processing with alpha-stable distributions and applications*, 1995, pp. 2–3, 109.

[36] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," 2016.

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

[38] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," p. 2, 2013.

[39] D. C. Plaut, S. J. Nowlan, and G. E. Hinton, "Experiments on learning by back propagation," pp. 1–9, 1986.

[40] J. G. Proakis and M. Salehi, *Digital Communications*, 5th ed. Mc Graw-Hill Higher Educations, 2008.

[41] T. Qiuyue and Z. Ling, "Mppsk demodulation based on neural network under impulsive noise," 2019.

[42] S. Ramjee, D. Yang, A. E. Gamal, S. Ju, X. Liu, and Y. C. Eldar, "Fast deep learning for automatic modulation classification," 2019.

[43] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Light gated recurrent units for speech recognition," p. 2, 2018.

[44] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," pp. 1–2, 2014.

[45] Y. Shi, P. Liu, D. Yan, Y. Chen, C. Li, and Z. Lu, "A hopfield neural network based demodulator for apsk signals," 2020.

[46] O. Simeone, "A very brief introduction to machine learning with applications to communication systems," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 4, pp. 648–664, 2018.

[47] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," p. 4, 2014.

[48] P. Stenumgaard, J. Chilo, J. Ferrer-Coll, and P. Angskog, "Challenges and conditions for wireless machine-to-machine communications in industrial environments," *IEEE Communications Magazine*, vol. 51, no. 6, pp. 187–192, 2013.

[49] Q. Tan and L. Zhao, "Msk demodulator and impulsive noise depression based on convolutional neural network with gated layers," 2019.

[50] Wen-jie Chen, J. Wang, and J. Li, "End-to-end psk signals demodulation using convolutional neural network," 2022.

[51] H. Wu, Q. Peng, J. Wang, R. Ma, and J. Ou, "A novel demodulation network for binary partial response cpm signals," 2020.

[52] I. Wu, H. Ohta, K. Gotoh, S. Ishigami, and Y. Matsumoto, "Characteristics of radiation noise from an led lamp and its effect on the ber performance of an ofdm system for dttb," *IEEE Transactions on Electromagnetic Compatibility*, vol. 56, no. 1, pp. 132–142, 2014.

[53] Y. Yang, F. Gao, X. Ma, and S. Zhang, "Deep learning-based channel estimation for doubly selective fading channels," 2019.

[54] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," p. 3, 2013.

[55] M. Zhang, Z. Liu, L. Li, and H. Wang, "Enhanced efficiency bpsk demodulator based on one-dimensional convolutional neural network," 2018.

[56] S. Zheng, X. Zhou, S. Chen, P. Qi, and X. Yang, "Demodnet: Learning soft demodulation from hard information using convolutional neural network," 11 2020.