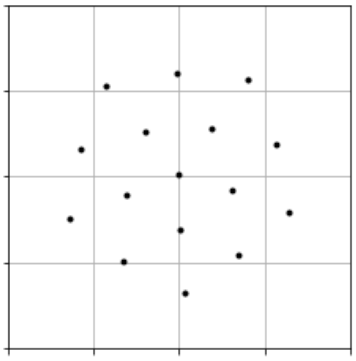
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2019



Project Title:	**Machine Learning for Communications with Short Block Length**
Student:	**Alistair P. S. Wallace**
CID:	**01062642**
Course:	**EEE4**
Project Supervisor:	**Dr Bruno Clerckx**
Project Cosupervisor:	**Dr Morteza Varasteh**
Second Marker:	**Dr Cong Ling**

MUST DO BEFORE SUBMISSION

- Correct Figure 1b Block, Diagram.
  - Transmitter and Receiver the wrong way round.
  - Receiver spelt ”Reveiver”
- Find bug or remove all caps section about unfound bug in BPSK encoding section.
- Got through and find all typos
- Remove pseudo code snippets, not necessary.
- Update abstract to include appropriate content.
- Implement Clerckx comments on the introduction/background section.
- Remove table from safety plan.

Abstract

This report focused on applying recent developments in machine learning to the field of communications to improve performance over channels which are unknown or difficult to model. It has been shown that optimising each stage of a communication system individually gives suboptimal performance, leading to investigating end-to-end learnt communication systems, where the optimal communication system can be learnt for a particular channel, environment and for specific hardware non-idealities.

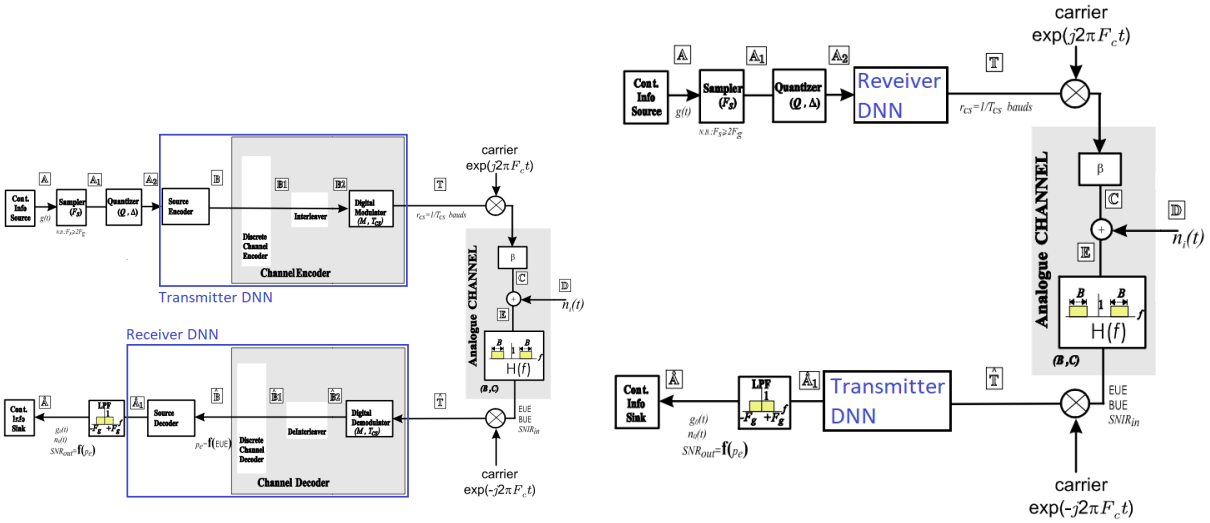
The report reproduces results from two recent papers [1, 2] on the subject, exploring unsupervised models and investigating supervised models with additive white Gaussian noise (AWGN), Rayleigh block fading (RBF). It then goes further by applying the above methods to Ricean fading (RF) channels.

The report produced predominantly similar results to [1] for supervised models, giving identical performance for two of the three (n,k) configurations. However, differences were found, sometimes showing lower performance of the technology in question or less aesthetic t-distributed stochastic neighbour embedding (t-SNE) based constellation diagrams, which the original authors overlooked.

Acknowledgements

This project is the culmination of not only my work, but also the help given by others. I would like to thank my supervisor Bruno Clerckx for his advice, supervision and feedback on the written sections of the report. Secondly I would like to thank my co-supervisor Morteza Varasteh for always being reachable for assistance, for his help with the implementation of the project and for helping me understand the underlying fundamentals of communication.

Contents



(a) A block diagram of a classical communications system. Taken from [4].

(b) A block diagram of a DNN based communication system. Based on a diagram from [4].

Figure 1: A comparison of a classical and DNN based architecture to show how the DNNs simplify the architecture.

1. Introduction

1.1. Introduction to the subject

This project is on machine learning for communications with short block lengths. A typical communication system is shown in Figure 1a. As can be seen, it has many discrete modular parts, all of which are optimised individually, however, it has been shown that this gives sub-optimal performance [3].

The idea of this research is to replace all the individually optimised modules with one deep neural network (DNN) each for the transmitter and receiver. This involves replacing everything within each blue box in Figure 1a with one neural network (NN), as is shown in Figure 1b.

Normally the information goes through two sets of encoding, source encoding to compress the information content, followed by channel encoding to make it robust to channel noise. Whereas using the DNN method all of the communication system's encoding can be learned simultaneously, and the overall communication system can be optimised for hardware non-idealities such as non-linear power amplifiers. It also allows the locally optimal communication system for a given channel to be learnt, without trying to categorise or make assumptions about the channel.

1.2. Project deliverables

The main aim of this project is to reproduce the results from two recent papers [1, 2] on the subject. The secondary deliverable was a potential extra objective for if the project was finished early. This deliverable was left unspecified, but would be to do something new. For instance tackling the limitation of

being limited to short block lengths, or investigating potential opportunities such as learning an optimal communication system for a channel which could not be mathematically modelled, or could be modelled but couldn't be solved.

The two papers originally contributed five deliverables, four from the first and one from the second. However the project supervisor advised that only the first deliverable from each paper were important and so only these deliverables were attempted.

The first paper, by O'Shea *et al*. [1], was only the second paper on the subject, following a paper 8 months earlier by one of the same authors [5]. The O'Shea paper first demonstrated that an end-to-end communication system could be learned in an autoencoder like method, the structure of which can be seen in Figure 2.

It trained a range of autoencoders across a range of (n,k) combinations for which the constellation diagrams and performance over different signal to noise ratios (SNRs). These autoencoders had several different types of regularisations which affected their constellation diagrams. Their performance was compared to non-learned methods of encoding, such as BPSK and QAM for channel encoding and Hamming maximum likelihood and hard decision for source encoding.

1.2.1 Side note on autoencoders

Note that while this would be better suited to the background section, it is necessary for understanding the introduction, so it has been placed here.

Autoencoders are a technique used in deep learning (DL) for encoding and compressing data. There are two NNs placed end-to-end, as in Figure 2. The input to the first network is $s \in \mathcal{M} = \{1, 2, \ldots, M\}$, where each s can be represented by k bits. The encoder (first network, analogous to the transmitter), compresses s to $x \in \mathcal{N} = \{1, 2, \ldots, N\}$, represented by n bits. The decoder (second network, analogous to the receiver) decodes x back to $\hat{s}$.

Normally with autoencoders $n < k$, because the autoencoder is finding a compressed form of $\mathcal{M}$ which maintains the information content, while reducing the number of bits used. However in this paper because there is noise added in middle layer, as shown in Figure 2, $n \geq k$ to increase robustness of the transmitted signal to noise (minimise information loss).

1.2.2 Project deliverables continued

Moving back to the O'Shea paper, it should be noted that all the following deliverables were written off by the project supervisor and so were not attempted, they are just described to give a better understanding of the O'Shea paper.

The paper then went on to investigate training two adversarial networks, but then showed that this was the same as optimising one multiple input multiple output (MIMO) NN with one common or multiple individual performance metrics. Thirdly the authors introduced radio transformer networks (RTNs)

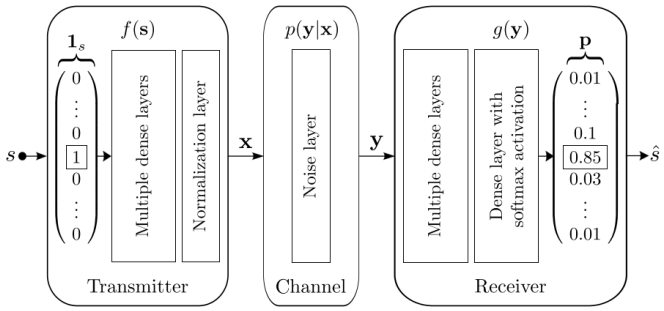


Figure 2: A diagram of the structure of the autoencoder model. Taken from [1]

which improved performance of the learned communication system in all cases. Lastly the authors investigated using NNs for modulation classification. All of these sections were competitive with or outperformed current state of the art.

The details of what the paper achieved will be covered in more detail in the background section, however a high level description was needed as reproducing these results were the main deliverables. Each of these sections would require writing some python code and then using this to reproduce the graphs and constellation diagrams. The authors stated that Keras "has become our primary tool used to generate the numerical results for this manuscript" [1]. Consequently Keras has also been used here to reproduce the results.

The second paper by Aoudia and Hoydis improves on the first by removing the need for a differentiable model of the channel. This helps access the technologies' main potential, which is learning general and optimal communication systems for unmodelable or analytically insoluble channels.

This paper exclusively solved the problem of implementing reinforcement learning to avoid the need for a differentiable channel model. However it should be noted that supervisor advice says this method's implementation is much more difficult and should not be underestimated.

The main twp deliverables are summarised below.

1. Make an autoencoder model and compare it's performance to differential binary phase shift key (DBPSK) and Hamming encoding across a range of signal to noise ratios (SNRs)

    - Produce complex channel symbolled (2,2) and (8,8) autoencoders, then obtain their constellation diagrams.
    - Produce non-complex channel symbolled (7,4) autoencoder and obtain a t-SNE dimensionally reduced constellation diagram.
    - Write encoding and decoding functions for Hamming HD and MLD
    - Write encoding and decoding functions for BPSK
    - Compare the performance of various encoding methods and debug until Figures 3a,b from [1] are reproduced.

2. Reproduce the unsupervised reinforcement learning model from [2] which uses the alternate training method. The compare the performance of this model against current state of the art and the autoencoder method across a range of SNRs for an AWGN and an RBF channel.

    - Produce a custom layer to implement RBF.
    - Replicate the paper's supervised model for an RBF channel.
    - Compare AWGN model from previous deliverable's performance across range of SNRs to the results from the paper. Next do the same with new supervised RBF model.
    - Produce unsupervised model for AWGN channel
    - Compare performance of AWGN unsupervised model across range of SNRs.
    - Produce unsupervised model for RBF channel
    - Compare performance of RBF unsupervised model across range of SNRs.
    - Once satisfied all four models are performing as they are reported to have in the paper, assess speed of convergence to correct model for all four AWGN, RBF, supervised and unsupervised models.

Any additional work is likely to fall into one of two categories. Either continueing work on focussed solely on information transfer as in the two main papers [1, 2]. This would focus on a goal like improving on the work done with reinforcement learning in [2], either tackle the limitations on block lengths, try to improve the convergence time, investigate the systems performance difficult channels or integrating RTNs with the reinforcement learning, as this always improved the results in [1].

The other option would be to be working on simultaneous wireless information and power transfer (SWIPT) systems. Either investigating the effect of block length on rate-power (RP), obtaining a model that features the practical limitations of the rectenna (rectifier-antenna) used in the EH part, or the dependency between delivered power and channel input average input power.

From assessment of the literature it is thought that it would be better to look into extending the work of reinforcement learning paper, because it better unlocks the potential of this technology. However the supervisor of the project informed the author that the he would decide based on which area was developing fastest at the time.

From a high level, the project is a software project and so the end deliverables will be a series of python scripts and functions which can learn a full end-to-end communication system. Along side these will be documentation of how to use these functions and scripts as well as comparison of the results produced with the results in the paper's the author is reproducing.

There may be customisations which allow it to use optimal formats for different channels, however this detracts slightly from the idea of an adaptable communication system that can learn an optimal communication system for any long term stationary channel.

Because of the nature of the project there will not be any hardware produced, although there will be some analytical work in analysis of channels.

1.3. Motivation

Two reasons for looking into this subject are, the recentness of it's discovery, and the exceedingly promising results, both within the field of communications and of the underlying technology. These within field results are particularly impressive in the long established field of communications where the improvements have become more marginal in modern times [1].

Other than the intellectual elegance of a fully learned communication system, there are significant advantages to a fully learned system.

The first reason is that although for known, ideal, mathematically model-able channels there are often mathematically proven optimal methods, (particularly for linear, stationary Gaussian channels), in reality there are many imperfections and non-linearities [6] both in the channel and also in the hardware such as non-linear power amplifiers and finite resolution quantisation. These often cannot be fully accounted for in an analytical solution, leading to sub-optimal performance. Whereas, a fully learned communication system could self-optimise for different hardware configurations and channel environments, with no assumptions about those factors other than stationarity.

Secondly communication systems are normally split into blocks (eg quantisation, modulation, source and channel encoding as shown in Figure 2). It has been shown that optimising these blocks individually leads to sub-optimal performance, as is shown for the cases of short block lengths and in use on practical channels in [3, 7] respectively.

A third reason to use NNs is that there have been impressive demonstrations of the learning ability of both feedforward and recurrent NNs (RNNs), suggesting these potentialy promising techniques should be applied to the field of communications. Recurrent NNs have been shown to be Turing-complete [8] and more recently there has been some very impressive work done on learning algorithms with RNNs [9].

More relevantly but less recently, feedforward NNs have been shown to be universal function approximaters [10]. This means that in the limits of width and depth they can approximate any function, even the long and complicated functions of transmitters and receivers.

The combination of these three demonstrations means that a NN should be able to learn any encoding function or algorithm which is optimal, in the limits of width, depth and time. Therefore if the current state of the art methods are really optimal for a given channel, then the NN should converge to these methods, when trained on this channel.

Fourthly due to their inherently parallel nature, NNs are more efficient than most applications at utilising parallel architectures such as GPUs and FPGAs [11]. NNs have also been shown to be very energy efficient with a very high throughput. This higher computations per watt is why Apple are trying to use their neural engine for more tasks in the iPhone [12], and is also why it could be suitable to internet of things (IoT) applications.

Additionally it has been shown that NNs maintain high performance at lower precision [13], which when combined with their parallel nature can lead to impressively high throughput and low power consumption. This could make this method useful for mobile phone communications because of their limited power budgets and high data needs.

This advantage could be further leveraged by using a chip designed specifically for NNs. This gave [14] impressively low power consumption. Other specialised chips such as Graphcore's the intelligence processing unit (IPU) claim significant performance increases on graphical processing units (GPUs), often as large as one or two factors of ten higher. A more moderate and already proved example is [15] using a GPU to train a network 70 times faster than a single core CPU.

2. Background

Due to the novelty of this subject there are a limited number of papers, with the first being from July 2017. Consequently most initial effort was focused on reading and understanding the following three papers [1, 16, 2], as well as learning background communications theory from a textbook [17]. However further but less thorough analysis has been done on a large number of other papers, which are referenced throughout this report.

A general background will be given on the technology area, before specific sections going into the required deeper detail for the papers which are being reproduced and the elements of communication theory which must be understood.

2.1. General background

The first end-to-end learned communications system was from O'Shea *et al*. [1, 5]. However machine learning had been applied to many jobs within receivers prior to this. Many of these applications such as "channel identification and equalization, coding and decoding, vector quantization, image processing, nonlinear filtering, spread spectrum applications" and cognitive radios have been summarised in [18, 19]. However research supports [1]'s claim that none of these methods had significant commercial success or adoption.

There has been a sharp rise in the number of applications of NNs to communications, probably largely due to the massive increase in access to open source machine learning libraries such as Tensorflow, Keras and Theano. This has been aided by the impressive and well publicised DL based improvements in computer vision. From this there has naturally been increased research into whether these benefits could be applied to the field of communications.

Applying DL to communications has been approached in two distinct ways. One way is that researchers have tried to apply DL to communications is by improving current algorithms and methods using DL the other is directly replacing the systems.

Some examples of the first group are improving belief propagation of code words with both feed forward [20] and recurrent neural networks [21]. With the RNN version giving significantly lower

complexity whilst maintaining performance. Another example was that applying DL to MIMO detection gave "state of the art accuracy with significantly lower complexity" in [22]. The previous three examples all built on the novel work of [23] in applying expert knowledge to NNs.

The second approach, of incorporating DL into communications by replacing the existing methods is the more relevant segment to this report. Initial research into the area was kick-started by a group of papers published during 2016-2017. One example is [24], where a sequence of pilot bits were sent across the unknown channel during training to build a non-linear channel model. [25] applied DNNs to molecular communication where the channel model cannot be modelled mathematically and [26] used a DNN to model resource allocation for wireless communication as an unknown linear function, giving "orders of magnitude" speed up compared to state of the art optimisation algorithms.

There has also been some initial research into end-to-end learned communication systems in [1, 5], with further development of a novel algorithm which uses reinforcement learning to avoid needing a channel model in [2]. Deep learning has also been applied to channel decoding [27], radio basis functions [28] and improving efficiency in radio transmissions [29].

2.2. Paper 1 - An introduction to deep learning for the physical layer

The first paper released on this subject was by O'Shea and Hoydis [1], in July of 2017. The paper cites that as communications is quite an established field, so improvements have become more marginal and semi-optimal solutions have been found for most problems. Consequently, a deep learning model (DLM) would have to do very well in order to be of any use.

DLMs have historically had a lot of success in fields such as image processing, where it is very difficult to write and algorithm which recognises or does something, but easy for a DLM. Whereas in communications rigid, mathematically backed algorithms for encoding and decoding signals can often be written, so DLMs have never been needed.

First O'Shea and Hoydis demonstrated it was possible to learn a full end-to-end communication system (transmitter, channel and reciever), for a given channel model which is optimised for a chosen loss function, in this case BLER (Block Error Rate). The structure of this model was similar to an autoencoder, however with a noise layer in the middle. The encoder acted as the transmitter, the decoder as the receiver and the noise layer as the channel.

The autoencoder based model was competitive with or outperformed state of the art, in this case Hamming MLD (maximum likelihood decoding), DBPSK, TS-QAM (time sharing quadrature amplitude modulation) and boosted trees. It also offered the potential of being applied to channels which were differentiable, but where the optimal solutions were not known.

O'Shea and Hoydis then extended this to an adversarial network of many Rx Tx pairs competing for capacity. Next showing this was the same as a single larger MIMO NN which could be optimised in one go with a common or individual performance metric. The jointly learned metric outperformed a TS-QAM system across a range of SNRs as can be seen in 4.

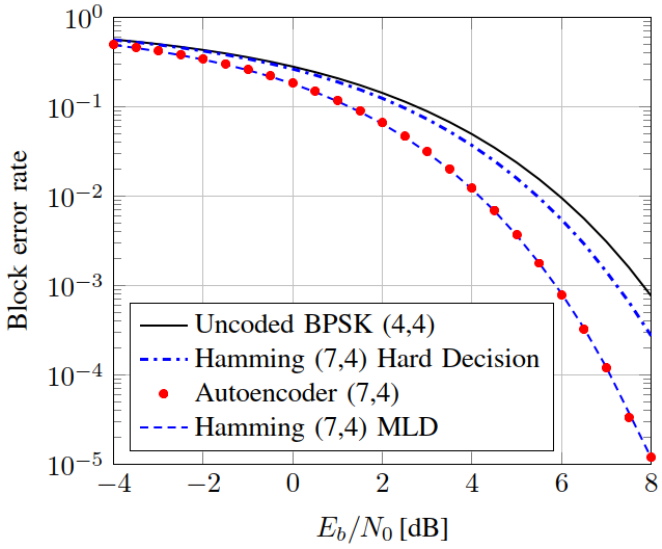


Figure 3: Comparing block error rate for different SNRs of a learned autoencoder model with current state of the art, Hamming MLD. Taken from [1]

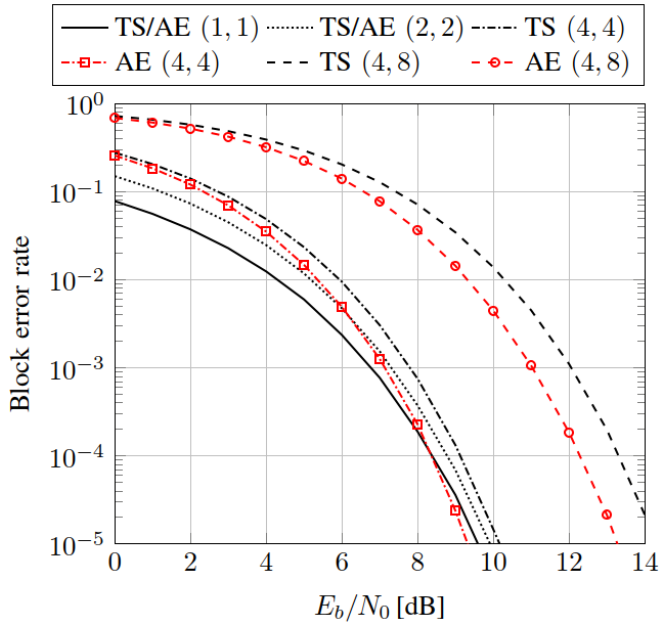


Figure 4: Comparing block error rate for different SNRs of a learned autoencoder model with a TS-QAM system for a multi-agent communication system. From [1]

It should be noted though that autoencoders perform notoriously poorly with messages they haven't seen before, consequently they normally have to be trained using all the messages from the set of symbols $M = 1, ..., M$. This stops the method scaling to longer block lengths as there are 2^k possible messages in a block of lengths k bits. This scaling is particularly poor as the models in this paper used one-hot

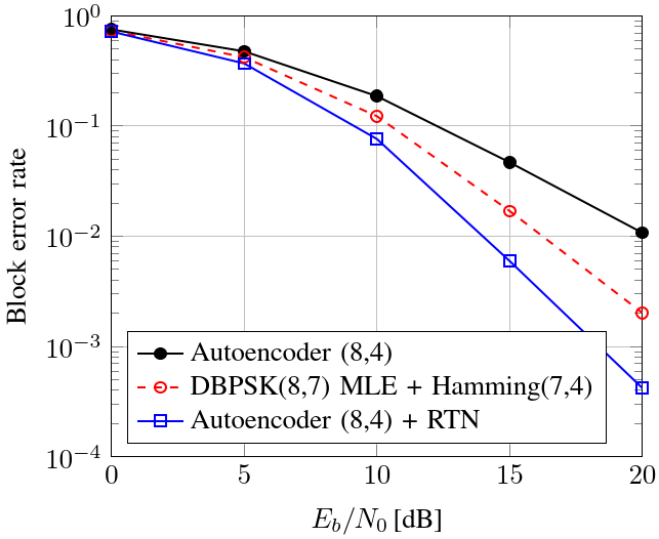


Figure 5: BLER versus Eb/N0 for various communication schemes over a channel with L = 3 Rayleigh fading taps. Taken from [1]

encoded inputs and outputs, meaning there must also be 2^k nodes and the input and output of the system.

After this the authors added an RTN (radio transfer network). RTNs can perform a predefined correction algorithm, for example multiply by a complex number or convolute with a vector. This can be used as a way in incorporate some expert features into a learned system. This was integrated into the DLM for the end-to-end training process and consistently outperformed the normal NN.

The learned system with added RTN's performance over a Rayleigh channel, with 3 ($L = 3$) fading taps, can be seen in Figure 5. It should also be noted that not only did the RTN consistently outperform current state of the art, it also converged significantly faster than the normal autoencoder. This can be seen in Figure 10 of [1].

Lastly O'Shea and Hoydis used a convolutional neural network (CNN) on raw coplex valued in-phase and quaternary (IQ) samples and found that it outperformed traditional classification techniques using expert features. This is consistent with the current trend that learned features are better than expert ones.

In conclusion, comparison with traditional baselines showed extremely competitive BLER, however the inherent autoencoder lack of scalability makes this method impossible for any long block length communication. This method also requires a differentiable mathematical model for the channel, which is severely limiting as a big opportunity for this technology would be to learn optimal communication systems for channels we can't model. This was still a very promising area though with potential uses of optical communications, where the channel impairments are highly non-linear and notoriously difficult to model and compensate for. Or alternatively unusual or complicated channels.

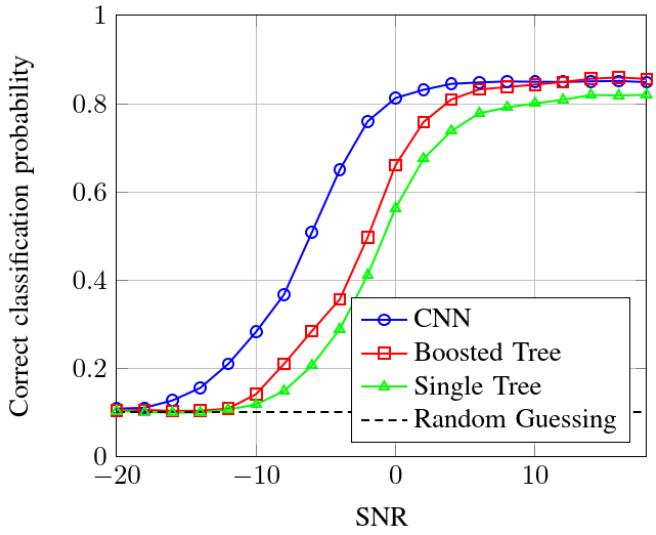


Figure 6: Classification performance comparison versus SNR. Taken from [1]

2.3. Paper 2 - End-to-end learning of communication systems without a channel model

As highlighted in the conclusion of the O'Shea paper, if an autoencoder like NN is used to learn a communication system then the channel model must be differentiable. This paper [2], written by Aoudia and Hoydis in April 2018, exclusively set out to remove the need for a channel model.

They produced a novel algorithm which solves the problem by iterating between the supervised training at the receiver and reinforcement learning (RL) at the transmitter. They showed it worked as well as supervised methods using additive white Gaussian noise (AWGN) and using Rayleigh block-fading (RBF) channels. Their method converged slower on AWGN but faster on RBF as can be seen in Figure 7. This is the first step towards learning comms systems over any type of channel without any prior assumptions.

The reason this innovation was so important is that channels are normally black boxes, where only the inputs and outputs can be observed. The authors used a method called "Policy Learning", which is a subsection of RL. RL provides a theoretical foundation for obtaining an estimate of the gradient of an arbitrary loss function with respect to actions taken by an agent. In this case the agent is the transmitter and loss is performance metric provided by the receiver. This would mean knowledge of the channel model and instantaneous channel transfer function is not needed which allows them to train an autoencoder purely from observations.

The "alternating training algorithm" trains the receiver first then the transmitter, alternating between training each side while keeping the parameters for the other side fixed. The receiver is trained using mini-batch stochastic gradient descent (SGD) as in other methods. For training the transmitter a random perturbation vector $\mathbf{w}$ is added. This is done for a whole batch of transmitted messages, these messages are then sent across the channel, and received as normal, the losses are calculated, and then transmitted

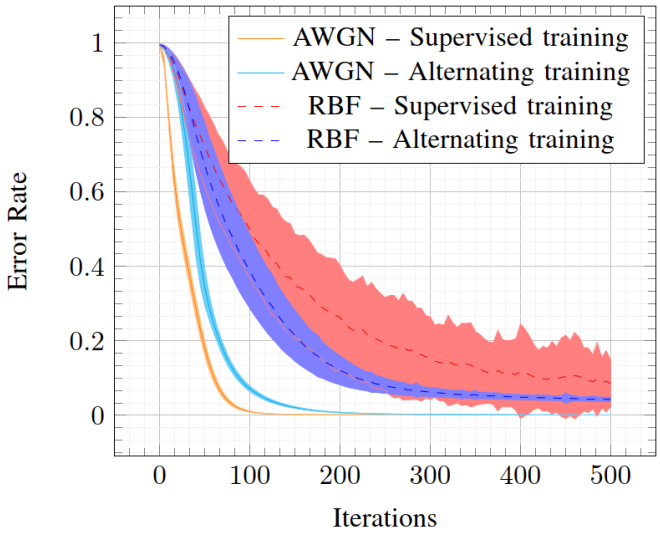


Figure 7: Evolution of the error rate over the first 500 iterations. Taken from [2]

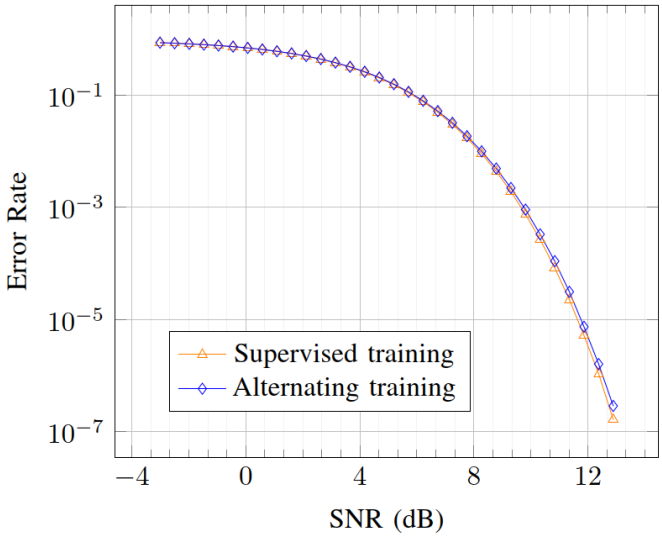


Figure 8: Alternative training vs autoencoder methods for AWGN Channel. Taken from [2]

back to the transmitter along a lossless channel. Finally a normal optimisation step is done using SGD where the loss-gradient is estimated using Equation 4 from [2].

The authors found that the alternating method took longer to converge with for the AWGN channel but less time for the RBF channel than the supervised learning method. However it had the same end performance as the supervised method in both cases. This performance can be seen in Figures 8 and 9.

In conclusion Aoudia and Hoydis showed that they could achieve the same end performance to a fully supervised approach, but without needing a mathematical model of the channel, so it could therefore be applied to any type of channel without prior analysis. Although their algorithm currently requires an additional reliable channel during training to feedback losses from the receiver to the transmitter, this is

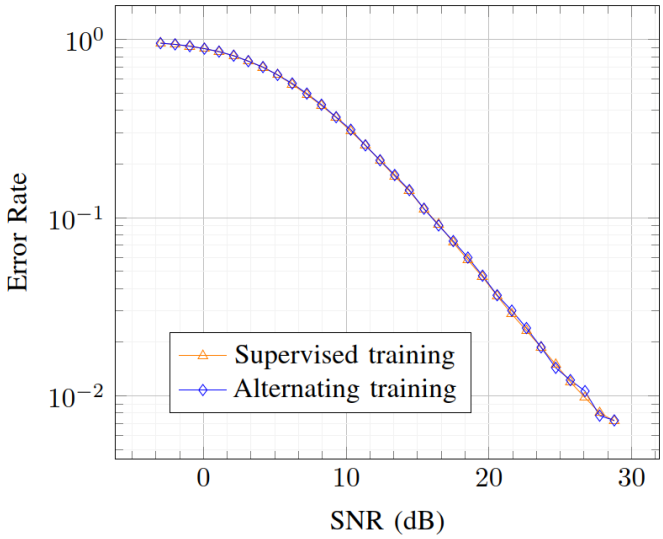


Figure 9: Alternative training vs autoencoder methods for RBF Channel. Taken from [2]

still a huge step. It also does not solve help the problem of being restricted to short block lengths, and arguably makes it worse in the case of AWGN as the method takes longer to converge even with shorter block lengths. Lastly it should also be noted that the networks for the AWGN and RBF channels are not actually the same and the RBF network's structure incorporates expert knowledge of the channel. So this method is not as general as it first seems.

However the importance of this step should not be underestimated. It allows learned communication systems to be used on channels which are either unknown, or too complex to model, which is arguably the biggest potential benefit of this technology.

2.4. Paper 3 - A learning approach to wireless information and power transfer signal and system design

This paper, written by Varasteh, Piovano and Clerkx has a slightly different focus. The previous two papers focused solely on achieving reliable communications with low error rates, this paper focuses on simultaneous wireless information and power transfer (SWIPT) using a nonlinear model, an autoencoder. This paper on learned SWIPT systems is motivated by the fact that designing SWIPT signals and systems analytically is very cumbersome.

There is a fundamental trade-off between the information rate and the power rate and so the NNs are optimised using a joint loss function. The observed results were inline with previously known theoretical results, particularly in that as you raise the demand for energy, the constellation diagram converges to one point a long way out along one of the axes, and the others closely clustering around the origin. However while this is what they would expect from the theory, it would not be possible to find these results theoretically or using numerical methods. So these found constellation, which can be found in Figure 10, diagrams are novel and significant.

To have good SWIPT it is essential to model the energy harvester (EH) with a high level of accuracy,

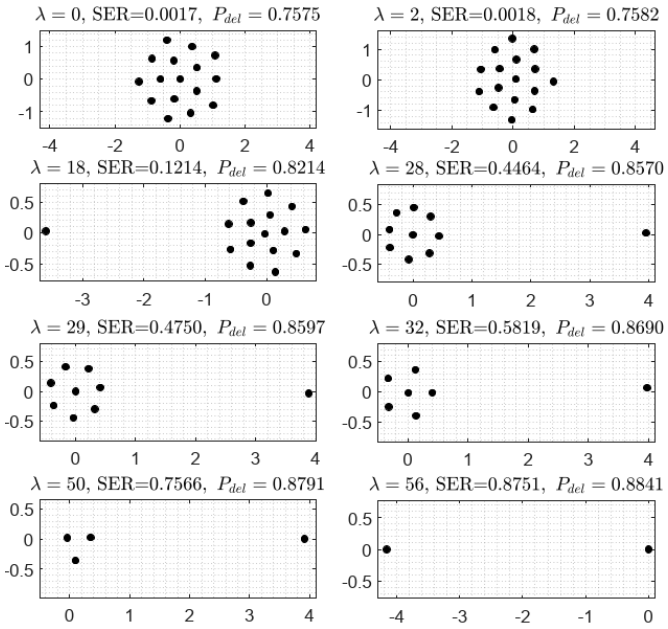


Figure 10: Representation of 16-symbols modulation for different values of (different information rate and power demand at the receiver) with SNR 20 dB. By increasing , the delivered power at the receiver increases. Taken from [16]

which is an antenna and a rectifier with fundamentally non-linear characteristics. In the literature it is usually modelled linearly, however using an NN allows it to be modelled non-linearly. Interestingly if you model the EH linearly an analytical study found it prefers single carrier transmissions and also there is no trade-off between information and power transfer. Whereas non-linear models favour multi-carrier transmission and have a trade-off between the two objectives.

The results were promising, the autoencoders and surpassed the performance of state-of-the-art algorithms. The more channel symbols, the more DC power is received by the receiver. This is because to optimally transmit lots of power the transmitter favours the high probability information symbols around zero, and the low probability information symbols a long way from zero along on of the two axes. The more symbols you have the better it can fit the ideal pdf with all but one clustered symmetrically around the origin.

The symbols near the origin are called the information symbols, and the one far away from the origin is referred to as the power symbol. Additionally the authors found that as the power demand increases, more of the channel symbols are sacrificed to increase the power delivery. Normally by mapping more of the information symbols to the origin.

Another interesting observation by the authors was that the delivered power was ”directly dependent on the channel input average power constraint”. This means that two systems trained at the same SNR but with different transmitted powers would have different designs. However they left investigation of

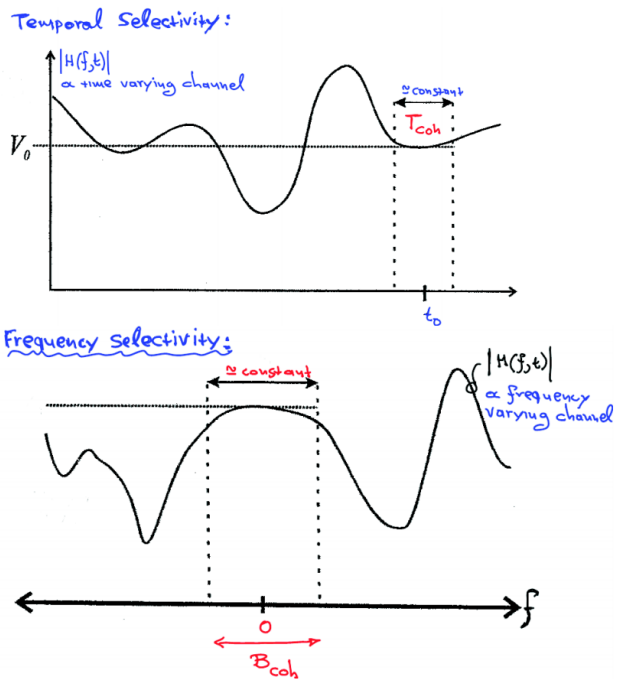


Figure 11: A figure to demonstrate the meaning of coherence time (T_{coh}) and coherence bandwidth (B_{coh}) from [4]

this aspect to another paper, along with investigation of the effect of block lengths on delivered power and making a model that features practical limitations of the rectenna non-linearity accurately.

In conclusion, this paper had some significant and completely new results, for instance the constellation diagrams would be impossible to find analytically because of the computational complexity of finding them. But in this paper they were able to find them using machine learning.

2.5. Background communications theory

2.5.1 Types of fading

Fading can be categorised into large or small scale, fast or slow and flat or frequency selective. Large scale fading is is due to path loss of signal as a function of distance and shadowing by large objects, such as buildings and hills. Because of it's large spacial size it is usually frequency independent. Small scale fading is due to constructive and destructive interference of multiple signal paths between the transmitter and the receiver. This smaller spacial scale is of the order of the carrier wavelength and is therefore a function of frequency.

In a learned communication system these factors could be taken account of, and as long as they were stationary an optimal communication system could be learned with no assumptions. Taking advantage of these factors is something a traditional communication would find more difficult.

To explain fast and slow fading we must first explain the meaning of coherence time. Coherence time

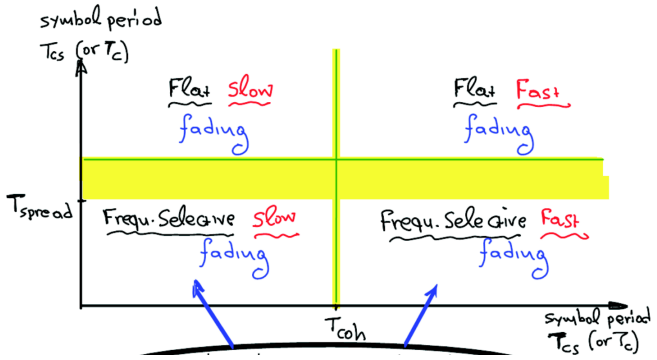


Figure 12: A figure showing how to classify a channel as fast/slow and flat/frequency selective fading taken from [4]

is the time for which a channel is roughly constant as is shown by the diagram in Figure 11. There is little consensus as to exactly what fast and slow fading means [17]. One interpretation is that a channel is fast fading if the coherence time is less than the delay requirement of the application, whereas is defined as slow fading if the coherence time is greater than the delay requirement of the application. However another definition which is shown in [4] and is demonstrated in Figure 12 is that fading is fast if the time per channel symbol (T_{cs}) is greater than the coherence time (T_{coh}).

Defined either way the main difference between the two is that in fast fading channels you can transmit channel symbols over multiple channel fades, whereas in a slow fading channel you must transmit over that particular fade. This is either to satisfy the delay requirement of the application, or because your T_{cs} is much shorter than your T_{coh}. Note that this behaviour depends on the channel characteristics (affects T_{coh}), the frequency (affects T_{cs}) and the application (sets the delay requirement). For instance voice transmission usually has quite a low delay requirement of $\sim$100ms [17], but in different channels and transmitting at different frequencies could be either slow or fast fading.

Flat and frequency selective fading are similarly defined as can be seen in Figure 11. Fading is flat if T_{cs} is greater than T_{spread}. This means that the multi-paths for consecutive channel symbols will not overlap. Frequency selective fading is where the T_{cs} is less than T_{spread} and so multi-paths for consecutive channel symbols will overlap.

Faster, flatter fading is easier to design a communication system for. In the cases of frequency selective fading, which through the multi-paths depends on the environment, a learned communication system could learn an optimal system. This is another case of something which might be much more difficult to do analytically.

Because of these characteristics it seems likely that a learned communication system would be more suited to a very complex, but stationary environment. A good example of this may be an urban environment with many multi-paths. This would be very hard to model analytically and to do so would likely require assumptions which might make the solution sub-optimal. Whereas a learned system would avoid

all these assumptions and learn the local optimum for that environment. However note that an urban environment may be too non-stationary for this method to work.

2.5.2 Capacity and block lengths

Shannon's capacity formula (Equation 1 taken from [30]) describes a theoretical limit on the capacity of an AWGN channel as a function of SNR, however it does not specify a method of achieving said bound.

$$C_{AWGN} = B.\log_2(1+\frac{P}{\sigma^2}) \text{ (bits/sec)}$$
$$C_{AWGN} = \frac{1}{2}\log_2(1+\frac{P}{\sigma^2}) \text{ (bits/symbol)} \quad (1)$$

AWGN channels are now normally decoded using iterative decoding algorithms which are far faster but have very similar performance to maximum likelihood decoders, such as "turbo" and "low density parity check" (LDPC) codes [30]. Using LDPC codes much lower error probabilities can be achieved by using longer block lengths [30] to approach maximum theoretical capacity.

Another driver towards long block lengths is that Shannon's separation theorem proves, that if you separate the source encoding and the channel encoding, then this approach becomes theoretically optimal in the limit of infinitely long block lengths [3]. This has led to most communication being done over very long block lengths. In OFDM if the block length is already very long then coding jointly across sub-carriers cannot increase the rate of reliable communication [30].

Capacity in fading channels must be thought of in a different way because whatever code is used the bit error probability (p_e) cannot be made arbitrarily small because the probability that the channel is in deep fade is non-zero [30]. Deep fade is strong destructive interference which can result in temporary failure of communications due to a massive drop in channel SNR, so will cause an error whether the signal is encoded or not. It should also be noted that Equation 2 always holds.

$$C_{Fading} \leq C_{AWGN} \quad (2)$$

Because the probability of deep fade is non zero the strict sense capacity is zero. instead we find C_ϵ where $P(deepfade) < \epsilon$. However despite the fact we think about it differently the conclusion is the same for fast fading (time varying) channels. Because the code-words span several coherence periods the outage probability improves. This is because you can treat the different coherence periods as independent identically distributed (i.i.d) variables. By the law of large numbers their total variance will be lower. Extending this principle we like the block lengths to be long both to average out the effects of the Gaussian noise and the fluctuations of the channel. Hence the same conclusion, long block lengths.

Unfortunately using long block lengths has downsides. Firstly long block lengths give significant coding delays increasing latency even though they maintain throughput. Because of this in applications

with tight delay constraints you may still get outage. Secondly for cases common in IoT applications where not much data needs to be send, long block lengths are not necessary and are impractical.

Note that the gains in longer block lengths are not as big for slower fading channels, so potentially this could be an area which a learned network of short block length communications systems could do well.

A problem with this environment though is that communication systems in slow fading channels often use feedback to the receiver to use *channel inversion* to increase their capacity. Channel inversion is where the transmitter varies the input power to keep the receiver SNR constant [30]. This can consume huge amounts of power though and so is not appropriate for IoT applications. So, if the learned communication system could be adaptable or had a mechanism for feedback from the receiver to the transmitter then it might provide an alternative to channel inversion. Making communication in slow fading channels much more possible for IoT applications.

For all of these cases above where long block lengths are not feasible, short block lengths are necessary. However the coding techniques used for long block lengths have been shown to be sub-optimal [31] when used with short block lengths. Because of this there is a motivation to learn communications techniques specifically for this area.

2.6. Additional notes

This technique has great potential for providing flexible communication across highly non-linear, unmodelable or mathematically insoluble channels. However it should be noted that in the case of non-stationary channels this method does not really have any potential without significant addition. The method could potentially be adapted to become adaptive, however this may be particularly difficult. This is because even in the most advanced form using the reinforcement learning you still require a noiseless channel between the receiver and the transmitter to communicate back what was received.

This is fine if you are pre-training the network for a stationary channel you can train it with a noiseless channel temporarily. However if you are trying to update the weights online then you may lose significant capacity or use valuable resources such as frequency trying to transmit back the received bits to train the transmitter. This weakness for non-stationary channels is very important as most wireless channels are highly non-stationary.

2.7. Analysis of competing products

The O'Shea paper [1] being replicated claimed there had not been much adoption of ML in commercial use [1], which at the time of writing was true, however this has now changed. After some research the main commercial competition found was a start-up called DeepSig. One of it's two co-founders was Tim O'Shea, the co-author of the two founding papers on the subject of end-to-end learned communication systems. It currently has two commercial products OmniSIG and OmniPHY.

OmniSIG is a software package that allows training and monitoring of a communication system on

a range of hardware devices. The package claims to often provide a 4-10 dB improvement and 10x reductions in computational complexity [32]. OmniPHY seems to be similar system however the information on DeepSig's website is not very informative about what the product does, other than it learns a communication system, much like the OmniSig [33]. DeepSig appears to be in it's infancy, and the author has been unable to find any other companies which are working in this area, suggesting it is still a very new corporate area. Additionally it is not entirely surprising that DeepSig seems to be the first, as it's founder is one of the founders of the field.

If this report was producing a commercial application then such intellectually established competition would be quite worrying. So while only producing an academic report is being produced, it makes the research arguably less valuable as there is less chance for a company to get an early monopoly on the market.

On the subject of competing papers and academics, due to the nature of the project being to reproduce the results of two existing papers, there are by definition competing papers. Alongside the two papers that have been reproduced here [1, 2] there are also several existing papers which have applied similar methods to similar problems. However few papers have learned a whole end-to-end communication system and the ones that have have been analysed in greater detail above.

2.8. Analysis of necessary software tools

One of the project specifications given to the author was to do the project in python and so this is why the python language was chosen. Although given a free choice python would still have been selected due to the large amount of high quality machine learning libraries available in python, such as Keras and Tensorflow. Another reason is that the large python community contributes to faster debugging and more support. Python 3.6.8 is being used because at the date of starting it is the latest version of python that supports Tensorflow and Keras.

The main software tool that being used is Keras with a Tensorflow backend. These are both open source machine learning libraries designed to make machine learning easier at a higher level. Allowing users to describe complicated networks in many fewer lines of code. The other reason that Keras has been chosed is that the two paper's being reproduced both specify that they used Keras [1, 2].

2.9. Analysis of necessary hardware

The main hardware specification is what device is needed to be able to train and test Keras models on. While any modern cpu core will be able to run Keras, it would be significantly faster to train and test on a graphical processing unit (GPU) as shown here [34]. But tensorflow and keras can still run on a simple cpu, and this requirement will be satisfied by any modern laptop or desktop.

The hardware which has been chosen to use is a HP Spectre with an Intel I7 6500u. This was chosen because of the convenience of it being the author's own laptop. This hardware has been used for the majority of the code development and running. However training of large models or many models has

been run on an amazon web services (AWS) g3.4xlarge instance to take advantage of the speed up associated with using a GPU.

A g3.4xlarge is equipped with 16 high frequency Intel Xeon E5-2686 v4 (Broadwell) processors and one NVIDIA Tesla M60 GPU, with 2048 parallel processing cores and 8 GiB of video memory. This significantly increased the speed of training models. This instance was selected because of it's high quality GPU, but also because of the author's familiarity with it from a fourth year course.

3. Evaluation Strategy

The two main deliverables are to reproduce the results from two papers [1, 2] as laid out in Section 1.2.2. Consequently assessing the completeness of the above deliverables should be relatively simple, in that the results produced can be compared with the results that are being reproduced, and if they are the same then the deliverable is complete.

For both papers performance of models was assessed by comparing the results of the code to the graphical results in the paper as there are no numerical results provided. Extra testing and validation was done along the way to check the the models were giving reasonable results. Strategies for this included plotting constellation diagrams, testing the system with no noise and using unit tests for BPSK and Hamming encoding methods.

To ensure written code was correct, code written for previous deliverables was used for testing new code, producing results for learned systems and current state of the art, to give ball park figures to check the code was roughly working. Then each small function was tested thoroughly to ensure the overall system was working. Where it was hard to predict whether the results made sense they were checked against theoretical estimations, and if there was still uncertainty the project supervisors was consulted.

Some extra comparisons which will be completed if there is time for evaluation of whether the developed methods are truly competitive with state of the art are comparing the learned systems to Shannon's theoretical capacity limit and current state of the art codes such as turbo and LDPC codes for an AWGN channel.

The generality of a particular architecture of learned communication system will be evaluated by comparing the learned networks and current state of the art across common fading channels such as Rayleigh and Ricean channels across a range of type of fading, large scale or small scale, fast or slow, flat or frequency selective. This would better evaluate the generality of a learned communction system than has so far been done in the literature.

This generality could be further evaluated by doing the same for some rare or hard to model channels. These being where the learned communication system should have the biggest advantage and so it would be interesting to see if they could deliver on their potential.

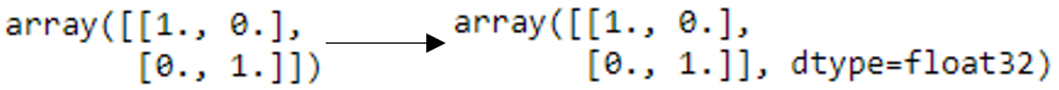


Figure 13: The input output of the initial autoencoder model during testing.

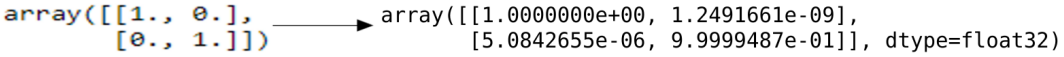


Figure 14: The input output of the autoencoder model with gaussian noise added at test time.

4. Design and Implementation

4.1. Paper 1 - Producing an Autoencoder

An autoencoder based communication system which could be trained was been produced relatively quickly, however it originally contained a bug causing it to not use all of the possible communication symbols. This was suspected to be due to the regularisation method, however was slow to fix.

4.1.1 Initial autoencoder model

The development was started by building a simple autoencoder model in using Keras with a Tensorflow back-end. This was trained on the $M = 2$ ($k = 1$) case with one hot encoded inputs and outputs.

The model had a single dense layer transmitter, L2 normalisation layer, a Gaussian channel layer and a single dense layer receiver. This meant when tested after training over 1000 epochs it gave the input output relationship shown in Figure 13.

4.1.2 Adding noise at test time

Clearly no noise has been added in the testing case. This was because the built in GaussianNoise layer in Keras is a training layer and so does not add noise if not in the training phase. To solve this problem a custom layer was added which adds noise both during training and testing as a channel model should. The code for this layer can be found by searching for "class GaussianNoise" in the *initial_dnn_comms.ipynb* notebook from the project github [35]. This changed the input output relationship to the one shown in Figure 14.

4.1.3 Adding a most likely symbol layer

The output of the receiver was a softmax layer, which gave an aposteri probability distribution for the sent symbol, given the received signal. However at test time it is desirable to know the bit error rate (BER), which requires discrete bits.

To deal with this another custom layer was added to the output of the model which would take in the aposteri probability distribution and output the most likely symbol. An example of input output of this

```
array([[0.63, 0.17, 0.21],                    array([[1., 0., 0.],
       [0.2 , 0.43, 0.37],                           [0., 1., 0.],
       [0.38, 0.44, 0.18],                  ->       [0., 1., 0.],
       [0.51, 0.18, 0.31],                           [1., 0., 0.],
       [0.3 , 0.41, 0.29]], dtype=float32)           [0., 1., 0.]])
```

Figure 15: The input output of the MostLikelySymbol layer for a randomly generated discrete pdf input.

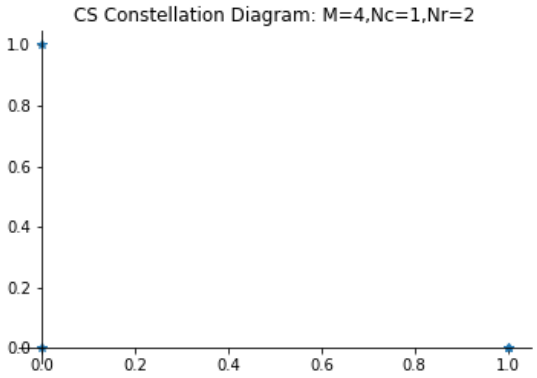


Figure 16: A constellation diagram produced by a trained (2,2) model which was clearly wrong.

layer is shown in Figure 15.

This layer was originally written using a numpy based function, however this would not compile as part of the Keras model. This was because Keras could not differentiate the layer, which is required when training NNs for back propogation, and so could not train the model. So a custom layer was written using Keras Tensorflow back-end functions which worked.

4.1.4 Making the channel symbols complex numbers

Next the channel symbols were turned into complex numbers to represent the in-phase and quaternary sections of the transmitted signal. This required modifying the channel nodes of the model to have an extra dimension, which in turn forced a change in the regularisation layer.

This allowed channel symbols to be meaningfully plotted. Some channel symbols for a (2,2) autoencoder are shown in Figure 16, but there are clearly only three constellation points, despite the fact there should be four as it should converge to a QPSK constellation diagram. This was because the model had not learnt the $(1, 1)$ channel symbol and so the first and third symbols were mapped to the same channel symbol. This was clearly an error, however it's source was not yet found until later.

Another obvious error from this constellation diagram was that the four points were only mapped to binary x and y coordinates. A normal QPSK constellation diagram would be expected to look more like what is shown in Figure 17. In Figure 17 all of the points have an equal magnitude from the origin but

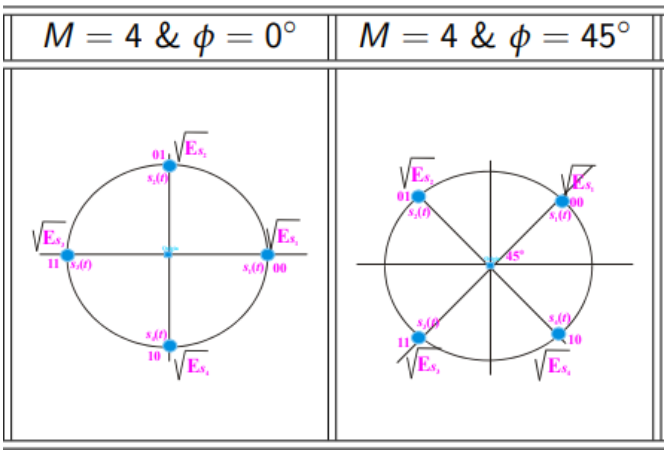


Figure 17: An example of a correct QPSK constellation diagram taken from [4].

are at phases $90°$ apart, to maximise separation for a given transmission power. This suggested that there was an error in the regularisation layer, which made this the focus for the next section.

4.2. Paper 1 - Producing an Autoencoder

4.2.1 Batch vs L2 normalisation

L2 normalisation and batch normalisation are defined below by Constraint 3a and Equation 3d respectively:

$$\left[\sum_{i=1}^{n} \|x_i\|^2\right]^{\frac{1}{2}} = 1 \tag{3a}$$

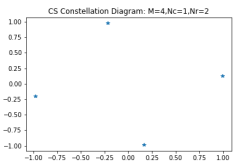
$$\mu_B = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3b}$$

$$\sigma_B^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu_B)^2 \tag{3c}$$

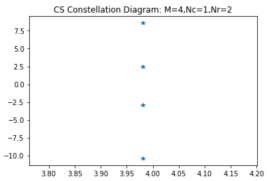
$$y = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{3d}$$

Batch normalisation was briefly investigated, and initially gave the promising looking constellation diagram shown in Figure 18a. This was a vast improvement on the previous constellation diagram shown in Figure 16. This motivated further investigation into batch normalisation to try and get the final small changes needed to reproduce QPSK. During this investigation the batch size was changed, the data size increased and axis over which BatchRegularization was performed was changed.

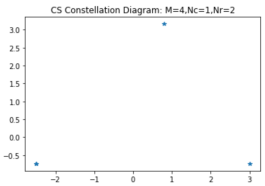
However all of these changes resulted in worse constellation diagrams than the original diagram shown in Figure 18a. Eventually it was found that this diagram was also unreproducible, and this result seemed



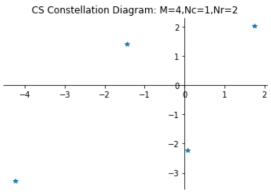
(a) The first batch normalisation model's constellation diagram.



(b) An incorrect constellation diagram where all the points only varied in one dimension, however were not placed on the origin of the other dimension.



(c) The common convergence to three instead of four channel symbols.



(d) A more promising constellation diagram, with four points varying in two dimensions, however not in the correct shape.

Figure 18: Examples of the missleading initial results obtained whilst using batch normalisation.

to have been a missleading fluke caused by the inconsistency of the model's performance from one intialisation to another.

The missleading results were also partially due to a bug, causing of a layer being bypassed. After lots of testing batch normalisation was rejected in favour of the original L2 normalisation, however over a different axis.

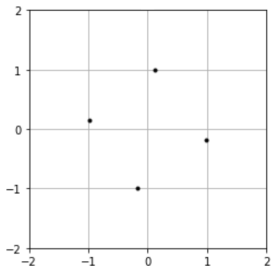
Batch normalisation simulates a unit average power constraint with zero DC component. However to obtain the QPSK constellation a constant instantaneous power constraint was needed, which was emulated by L2 normalisation in the last layer of the transmitter. This means that the L2 normalisation method was more appropriate and reintroduction of the L2 normalisation led to the contellation diagrams in Figure 19 being produced.

These constellation diagrams, while still inconsistent and not always correct, were always varying in two dimensions and had a unit instantaneous power budget so were an improvement on the constellations produced from the batch normalisation.

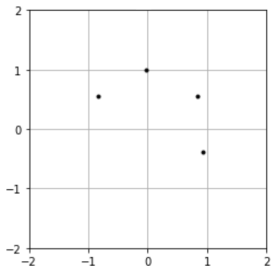
At this point the style of the constellation diagrams was changed to match the style of those in the paper for easier comparison. The square format made identifying QPSK and 16 QAM constellations simpler.

4.2.2 Leaky-Relu over ReLu

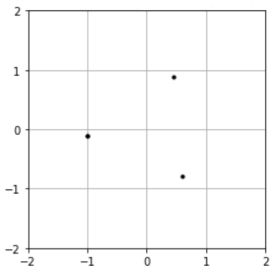
As can be seen in Figure 19 the constellation diagrams were not consistent at all. Occasionally, roughly one in ten times, the (2,2) model would converge to the correct QPSK constellation diagram. However often it would either have multiple input points converge to the same channel symbol, or would converge to a diagram which was clearly not optimal.



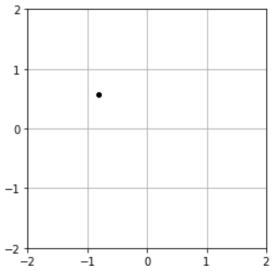
(a) A correct (2,2) constellation diagram



(b) Incorrect (2,2) constellation diagram

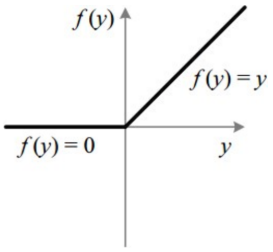


(c) (2,2) autoencoder convergence to three instead of four channel symbols.

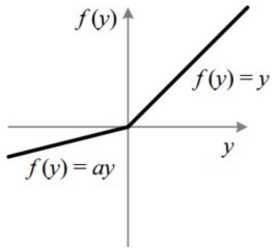


(d) (4,2) autoencoder convergence to one instead of sixteen channel symbols.

Figure 19: Four example constellation diagrams illustrating the inconsistency of the original L2-normalised (2,2) and (4,2) autoencoders with ReLu activation functions.



(a) ReLu activation function.



(b) Leaky-ReLu activation function, typically $a \approx 0.01$.

Figure 20

Meanwhile the (4,2) model was underfitting and consistently converging to one channel symbol instead of sixteen, but the exact channel symbol changed each time. This suggested that the model was underfitting, either due to insufficient layers or an inappropriate activation function.

Some research suggested that both these issues might be to do with ”neuron death” giving dependence on initialisation values. This is a problem characteristic of the ReLu activation function, shown in Figure 20a. The ReLu activation function outputs zero for all non-positive inputs, meaning that if weights are initially set so that a neuron consistently outputs a negative value pre-activation-function, it will consitently output a zero.

This means that the weights will not be updated through back-propogation as the partial derivatives of the error with respect to the weights will be zero. As a result the model will not improve for these neurons and will converge at an incorrect solution.

A suggested solution to this problem was using a ”leaky-relu” activation function, shown in 20b instead. A brief trial of this showed much improved performance, prompting further investigation into

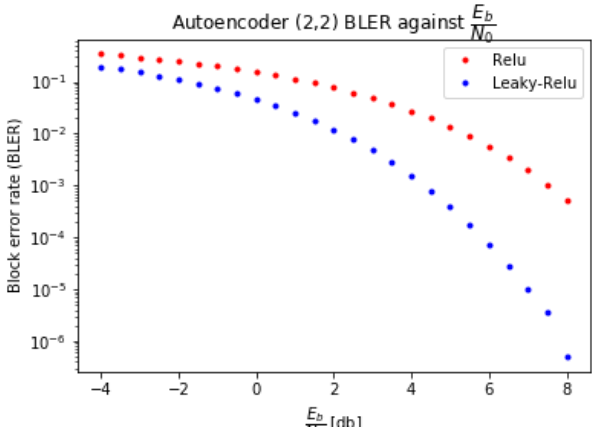


Figure 21: A comparison of the BLER performance of two leaky-ReLu and ReLu based (2,2) autoencoder models.

alternative activation functions in the section below.

The improved (2,2) constellation diagram seen in Figure 22a was consistently produced with each channel symbol being equally spaced, centred around the origin, having an equal magnitude and a phase difference of 90° from adjacent points. The only parameter which changed phase of the point in the first quadrant. The (4,2) constellation diagram shown in Figure 22b was also produced for the first time and was then consistently reproduced.

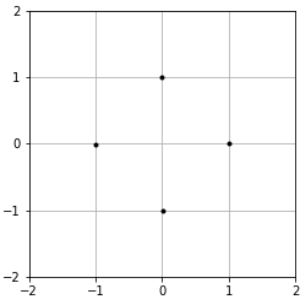
The BLER of two ReLu and Leaky-ReLu based models were also found over a range of SNRs. This was done to ensure that the metric of test error at training SNR was valid for evaluating model quality. It could be possible that some activation functions performed better at some SNRs and worse at others. The results of this are shown in Figure 21. The leaky-ReLu model consistently outperforms the ReLu model across all $\frac{E_b}{N_0}$ ratios. This both confirms that leaky-ReLu is a better activation function for this problem and that test error at training SNR is a valid metric for assessing model quality.

Now that successful models had been reproduced all of the layers in the models were named so that the weights could be saved and loaded into new models to avoid the time overhead of retraining larger models. The model weights were saved in the *.h5* format because neither the JSON nor YAML formats worked with the custom *GaussianNoise* and *MostLikelySymbol* layers.

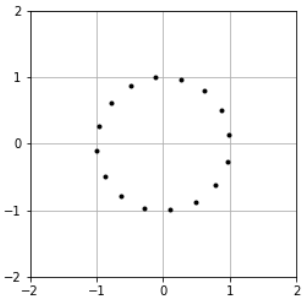
4.2.3 Comparing different activation functions

Despite having visually reached the correct constellations for the (2,2) and (4,2) models, due to the vast improvements gained in the previous section from changing the activation functions, six different activation functions were compared in this section.

The six different activation functions were *ReLu*, *tanh*, *sigmoid*, *linear*, *softmax* and *leaky-ReLu*. Because the largest issue faced so far had been lack of consistency, ten (2,2) models were trained for



(a) QPSK constellation diagram produced by the (2,2) model with a leaky-relu activation function.



(b) 16 QAM constellation diagram produced by the (4,2) model with a leaky-relu activation function.

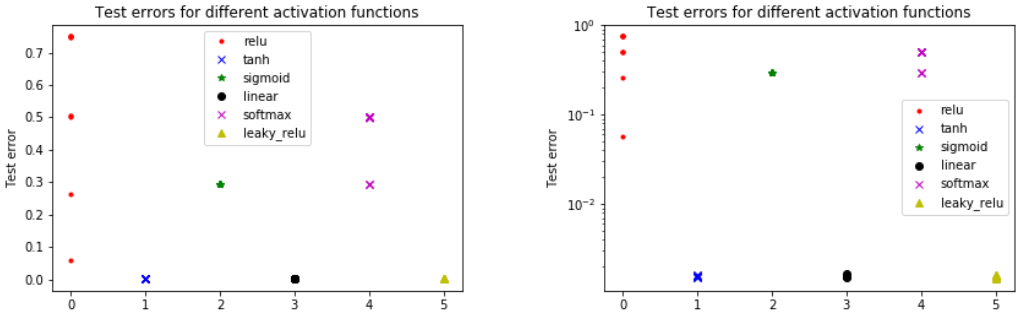
Figure 22

each activation function to assess the consistency of the activation function's performance. Each model was trained for five epochs, over a dataset of 10^7 shuffled copies of the four possible one-hot-encoded inputs, with a batch size of 4000. At the end of the training, each model was tested on a seperate test set. It's block error rate, constellation diagram and model weights were then all saved before moving onto the next model.

The results can be seen in Figure 23 and Table 1. It was decided that *leaky-ReLu* had marginally outperformed *tanh*. This was because, although they had the same average error and same variance of error, leaky-ReLu had a slightly lower minimum value. As the models were only trained for five epochs and as producing the final model would involve selecting the best model over a number of initialisations, it was decided that the minimum error was the most important statistic. Hence *leaky-ReLu* was selected as the best activation function. Because of this it was decided to use *leaky-ReLu* for all hidden layers, and use *tanh* for the output layer to capture non-linearity.

Activation Function	Mean	Min	σ
ReLu	0.55835	0.05784	$2.30e-1$
Tanh	0.00156	0.00151	$3.00e-5$
Sigmoid	0.29459	0.29413	$3.20e-4$
Linear	0.00157	0.00153	$4.00e-5$
Softmax	0.43870	0.29450	$9.42e-2$
Leaky ReLu	0.00156	0.00147	$3.00e-5$

Table 1: Activation function performance statistics from ten initialisations



(a) Linear axes

(b) Linear x-axis, logarithmic y-axis

Figure 23: Comparison of the performance of five different activation functions over ten instances of a (2,2) autoencoder model.

4.2.4 Model training methods

A number of methods were used during the training of the models to improve the performance of the end model. Firstly it was found that increasing the batch size during training improved the end validation loss. This is supported by the findings of [36], which suggest that increasing the batch size is in practice equivalent to decaying the learning rate. The batch size was started at the relatively large batch size of 1000. This lowest point is still relatively large as the noise is randomly generated for each message which is transmitted. Hence the batch size of 1000 is chosen to reduce the dependence on individual noise. This is by the law of large numbers illustrated in Equations 4 as a reminder.

$$\begin{aligned} \text{if: } y &= \frac{x_1 + x_2 + \cdots + x_n}{n} \\ \text{then } \mathbb{E}[y] &= \mathbb{E}[x] \\ \text{and } Var(y) &= \frac{\sigma_x}{\sqrt{n}} \end{aligned} \tag{4}$$

Hence as the batch size is increased the variance of the batch's noise decreases with $\sqrt{\text{Batch Size}}$. It is theorised that the model should converge faster if trained at a higher noise variance with a larger batch size, due to the higher noise variance giving larger loss gradients, and the larger batch size smoothing out the gradients to give an equally stationary loss surface. However this has not been tested empirically.

In practice the batch size was varied in three steps, 1000, $\sqrt{M} * 1000$, $M * 1000$. It was found that initial progress was made more quickly at a batch size of 1000, because there were more weight updates per epoch, then after the model performance converged at this batch size the batch size was increased and this process repeated twice more. After the batch size was increased each time the performance would at first improve quickly for a few epochs, before slowing and eventually congerging again.

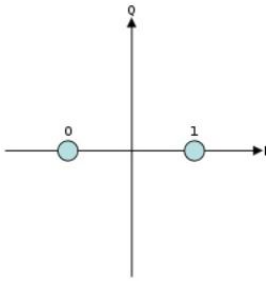


Figure 24: BPSK constellation diagram.

Another method used during the training of the models was to use the EarlyStopping callback in Keras. This was used to try and bring a constant process to training method. The EarlyStopping callback monitored the validation loss and would stop training if the validation loss did not improve after 20 epochs. This method works if the model is being trained over a large number of epochs like 1000, howver not if the model is being trained over 30 epochs. It also has the dissadvantage that it is not stopping at the best model, it is stopping 20 epochs later, so if the validation loss significantly increases due to overfitting in these final 20 epochs then the quality of the end model will be significantly affected. However having a patience parameter of 20 also has the advantage that it can cope with longer flat sections of the loss surface. This has proved useful quite a few times during the training of the larger models, when the validation loss has hardly improved for 20 epochs, before dropping off again more steeply.

After moving on to reproducing [2] the ModelCheckpoint callback was introduced. This would save the model after every epoch, if and only if the validation loss had improved from the current best. This was a better way of ensuring the best model was saved and allowed the patience parameter of the EarlyStopping callback to be increased to improve it's ability to deal with small gradients and flat sections of the loss function.

Training multiple many models with different initialisations should have been used for final models, however due to the consistency of the *leaky-ReLu* and *tanh* activation functions shown in Table 1 this was not done for the majority of models, however may be used for getting the last few $\%$ of performance out of the final models.

4.3. Adding non learned encoding

4.3.1 Adding BPSK encoding

Binary phase shift keying (BPSK) is the $m = 2$ case of phase shift keying (PSK), also known as 2-PSK. It has two points, seperated by $180°$. It's constellation diagram is shown in Figure 24, with two points

placed symetrically around the origin on the real axis. It's encoding is given by equation 5a.

$$s_i(t) = A_c \cos(2\pi f t + \pi(1-i)),\ \text{i = 0,1} \tag{5a}$$

$$\therefore s_0(t) = -A_c \tag{5b}$$

$$\text{and } s_1(t) = A_c \tag{5c}$$

Compared to the other m-PSK methods BPSK's channel symbols are the furthest separated and therefore it has the highest tolerance to noise, making it the most robust PSK. This is because it only encodes one bit, so it sacrifices data rate. BPSK's theoretical BER is given by Equation 6 from [37].

$$P_b = \frac{1}{2} erfc\left(\sqrt{\frac{E_b}{N_0}}\right) \tag{6}$$

Where the erfc function is defined by [38] as:

$$erfc = \frac{2}{\sqrt{\pi}} \int_{x=0}^{\infty} e^{-t^2} dt \tag{7}$$

The $erfc$ function gives the probability that a $|Y| \geq x$ where $Y \sim \mathcal{N}(0, \frac{1}{2})$. So $\frac{1}{2} * erfc(x)$ gives the tail probability of a normal distribution (Y).

This relationship means that the theoretical block error rate (BLER) can be calculated to ensure that the experimental graphs found later are correct.

BLER is defined by the European Telecommunication Standards Institue as *"the ratio of blocks received in error to the total number of received blocks, where a block is defined as received in error if the error detection functions in the receiver indicate an error as the result of the Block Check Sequence (BCS)"* [39].

This is taken to mean that if any of the bits in a block are incorrect, then the block is taken as wrong. The BLER is then the percentage of blocks which have any bits which are incorrect. As the noise for each block is assumed to be i.i.d, the above definition gives the following relationship between P_{be} and P_e in Equation 9. Note that this assumes stationarity, which is often not a valid assumption in wireless channels.

$$P_{be} = 1 - (1 - p_e)^n \tag{8}$$

$$P_{be} = 1 - \left(1 - \frac{1}{2} erfc\left(\sqrt{\frac{E_b}{N_0}}\right)\right)^n \tag{9}$$

This theoretical check was actually very useful as it unearthed the second of two bugs in the BPSK encoding and decoding. The graph that made these bugs obvious is shown in Figure 25. Stating that

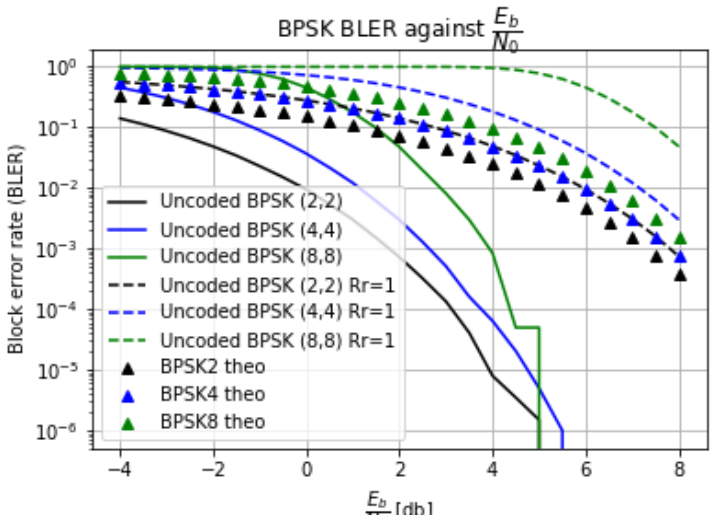


Figure 25: BPSK BLER performance over a range of SNRs.

results are wrong based on a other results requires a large amount of confidence in the correcting set of results. The theoretical results have this confidence due to matching the values displayed in the paper and the equations producing them being widely agreed upon.

For reference the simulation of the BPSK encoding in the AWGN channel was run by the process outlined in sudo code below. The exact code (which is almost identical) can be found at [35].

```
# Encode into bpsk
bpsk_encoded = bpsk_encode_vec(test_data)
# Add AWGN noise
noise = std * np.random.randn(bpsk_encoded.shape)
received = bpsk_encoded + noise
# Decode from bpsk back to bits
bpsk_decoded = bpsk_decode_vec(received)
# Get Block error rate
return get_block_error_rate(test_data, bpsk_decoded)
```

Note that *bpsk_encode* and *bpsk_decode* were vectorised because it was found that this reduced the latency of the simulation process by approximately 60%. This was important, as to give smooth curves the BPSK BLER was tested over a very large test data set and also for 25 different $\frac{E_b}{N_0}$

The first bug was that the noise standard deviation was being calculated incorrectly from the $\frac{E_b}{N_0}$. According to the O'Shea paper the variance is calculated from $\frac{E_b}{N_0}$ using the equation below [1].

$$\sigma^2 = \left(2R\frac{E_b}{N_0}\right)^{-1} \tag{10}$$

The paper is not entirely clear with it's definition of n, as to whether n is the number of real channel

uses or the number of complex channel uses. Originally it was decided the later was correct, meaning that $R = \frac{k}{n_c}$. This was found to be incorrect as it gave incorrect σs and this led to if being found that n must be the number of real bits. This gave the change from the solid lines ro the dashed lines in Figure 25. The n_c being lower than n_r was making the R two times larger, which was decreasing the variance of the channel noise by a factor of two. This gave the lower error rates shown by the solid lines. The jaggedness of the lines at the lowest error rates was due to insufficiently sized data sets to smoothly represent a error rate in the region of 10^-7.

At this point the second bug was found.

After the solving of this second issue the simulated BLER of the BPSK was indistinguishable from the theoretical BLER. Note that all of the code for the BPSK encoding and decoding can be found in the *Implementing Uncoded BPSK* section of *initial_dnn_comms.ipynb* at [35].

4.3.2 Adding hamming hard decision encoding

Hamming codes are a family of linear block code that allow error detection and correction. They were invented by Richard W. Hamming in 1950, where he laid out the general theory and used the (7,4) as the main example [40].

Mathematically there are an infinite number of potential Hamming codes with the following characteristics.

$$r \geq 2 \tag{11}$$

$$\text{block length } n = 2^r - 1 \tag{12}$$

$$\text{message length } k = 2^r - r - 1 \tag{13}$$

In the r = 3 case we get the most well known Hamming code, and the one used in this paper, the (7,4) code.

$$r = 3 \tag{14}$$

$$\therefore n = 2^3 - 1 = 7 \tag{15}$$

$$\therefore k = 2^3 - 3 - 1 = 4 \tag{16}$$

In the (7,4) Hamming code, for every four data bits transmitted three parity bits are added. These three parity bits allow the detection and correction of single errors. The combinations of the parity bits are given by Figure 26 and Table 2. Note that the (7,4) Hamming code cannot detect or correct multiple errors.

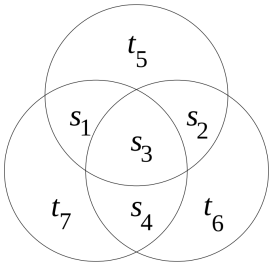


Figure 26: Venn diagram showing parity and data bits for a Hamming (7,4) code. Data bits are denoted s_i and parity bits are denoted t_i taken from [41].

However given that the probabilities of one, two and k errors are given by Equation 17a, b and d respectively, if p_e is small then the higher order terms will be negligible. For each extra number of errors you become able to correct you will reduce your BER by a factor of p_e.

$$p_{e1} = 1 - \left[(1-p_e)^4 + 4p_e(1-p_e)^3\right] \tag{17a}$$

$$p_{e2} = 1 - \left[(1-p_e)^4 + 4p_e(1-p_e)^3 + 6p_e^2(1-p_e)^2\right] \tag{17b}$$

$$p_{ek} = 1 - \left[\sum_{i=1}^{k} \binom{n}{i} p_e^i (1-p_e)^{n-i}\right] \tag{17c}$$

$$\therefore = \sum_{i=k+1}^{n} \binom{n}{i} p_e^i (1-p_e)^{n-i} \tag{17d}$$

$$\text{where: } 0 \leq k \leq n \tag{17e}$$

Hence, despite the fact that the (7,4) Hamming code can only remove single errors this should remove the vast majority of errors and will reduce the BER by approximately a factor of p_e, which is likely to be $\approx 10^{-3}$.

Bit #	1	2	3	4	5	6	7
Transmitter Bit	$\mathbf{p_1}$	$\mathbf{p_2}$	$\mathbf{d_1}$	$\mathbf{p_3}$	$\mathbf{d_2}$	$\mathbf{d_3}$	$\mathbf{d_4}$
$\mathbf{p_1}$	Yes	No	Yes	No	Yes	No	Yes
$\mathbf{p_2}$	No	Yes	Yes	No	No	Yes	Yes
$\mathbf{p_3}$	No	No	No	Yes	Yes	Yes	Yes

Table 2: A table showing which parity bits interact with which data bits in a (7,4) Hamming code.

Because the Hamming codes are linear codes, all the encoding and decoding can be performed using linear algebra. To perform the encoding and decoding three matrices are needed. The generator matrix G, the parity matrix H and R the decoding matrix [41] which are all defined below.

$$\text{Generator Matrix: } \mathbf{G} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{18}$$

$$\text{Parity Check Matrix: } \mathbf{H} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{19}$$

$$\text{Decoding Matrix: } \mathbf{R_{ham}} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{20}$$

The Hamming encoding step is simple, the transmitter message is produced from the column vector $\mathbf{p}$ by doing the matrix multiplication below in Equation 21, then taking the second modulus of each element. This is equivalent to taking the last least significant bit of the number in binary. For instance $1 \to 1, 2 \to 0, 3 \to 1$.

$$\mathbf{s} = mod_2(\mathbf{Gp}) = LSB(\mathbf{Gp}) \tag{21}$$

The sent message $\mathbf{s}$ then passes through a channel where an error may or may not be added. The received vector is denoted:

$$\mathbf{r} = \mathbf{s} + \mathbf{e} \tag{22}$$

Where $\mathbf{e}$ may be the zero vector. The syndrome vector $\mathbf{z}$ is obtained by matrix multiplying by $\mathbf{H}$. If e has less than or equal to one non-zero element then the result will be H.

$$\mathbf{H_r} = \mathbf{H}(\mathbf{x} + \mathbf{e}_i) \tag{23}$$

$$= \mathbf{Hx} + \mathbf{He}_i \tag{24}$$

$$= \mathbf{0} + \mathbf{He}_i \tag{25}$$

$$= \mathbf{He}_i \tag{26}$$

$$\mathbf{z} = mod_2\left(\mathbf{Hr}\right) \tag{27}$$

Noting that $\mathbf{r}$ has been rounded $\mathbf{z}$ is a three bit binary number. If only one bit has been changed then this number corresponds to the index (indexed from one to seven) of the bit which has an error. The value of this bit can be simply flipped to correct the error.

After the error correction stage the message must be decoded. The message is simply recovered my matrix multiplying by $\mathbf{R}$.

$$\hat{\mathbf{p}} = \mathbf{R}\mathbf{p} \tag{28}$$

It should be reiterated that the Hamming (7,4) will incorrectly diagnose multiple bit errors and so will not be able to correct them. As with the previous section the exact code for the Hamming (7,4) Hard Encoding can be found in the *Hamming Hard Decision Encoded BPSK* section of the *initial_dnn_comms.ipynb* notebook in [35], but some psuedo code for end-to-end encoding and decoding process is provided below.

```
# Encode the one_hot_encoded_vectors into bits
s = hamming_7_4_encode(p, G)
# Add noise
e = std * np.random.randn(s.shape)
r = s + e
# Do error correction and decode message
p̂ = hamming_7_4_decode_and_correct(r, H, R_ham)
# Get Block error rate
bler = get_block_error_rate(p, p̂)
```

4.3.3 Initial Hamming HD performance and debugging

After first producing the Hamming (7,4) encoding method it was tested seperately. It was tested using unit several unit test vectors with deterministic errors added. The results of these unit vectors at each stage of the process were compared with results calculated by hand. It was also tested with Gaussian distributed errors added, first with a standard deviation of zero, and then with small standard deviations.

After this seperate training the Hamming (7,4) source encoding was combined with BPSK channel encoding, then it's performance evaluated over a range of SNRs. This performance was then compared to the already produced (2,2) leaky-relu autoencoder model and the BPSK (4,4) model which gave the results shown in Figure 27

These results did not appear to be correct, despite all three of the encoding schemes having very similar performance across the whole range of SNRs. Thjs aspect was potentially compatible with Figures 3a and 3b because the (2,2) Autoencoder model would be expected to be slightly better than the (4,4) BPSK and slightly worse than the Hamming (7,4) hard decision encoding.

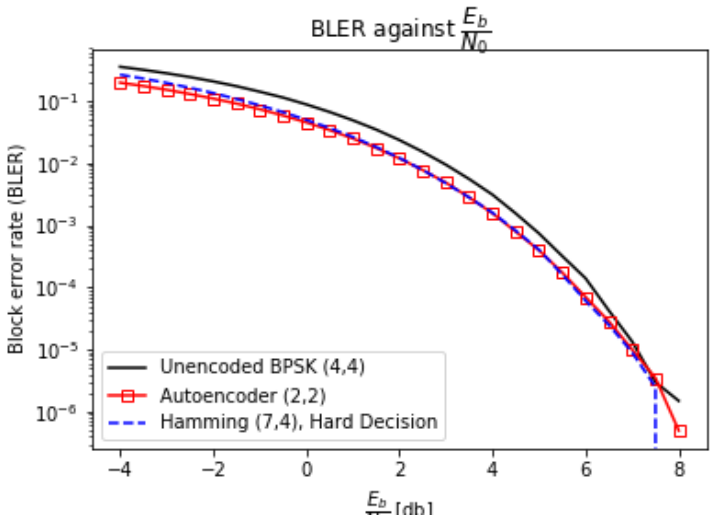


Figure 27: Original graph of BPSK (4,4), Hamming(7,4) and Autoencoder (2,2) BLER performance, before setting Rr correctly.

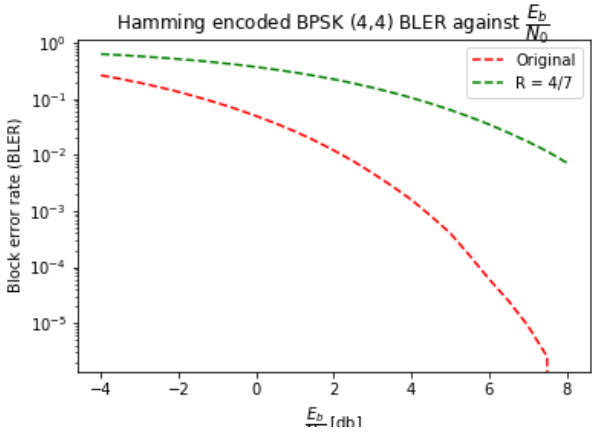


Figure 28: Comparison Hamming HD BLER performance with correct and incorrect values of R_r.

However at $\frac{E_b}{N_0} = 8$ all three of the encoding schemes were achieving BLERs of 10^{-6}, which was significantly better than the $\approx 10^{-3.5}$ results in the paper. The correct relative performance suggested that there was an error with the way the noise variance was being calculated from the $\frac{E_b}{N_0}$.

This intuition was correct, this led to the discovery n being the number of real bits instead of number of complex bits discussed in the Section 4.3.1 on adding BPSK. After this error was corrected the Hamming HD BLER performance was a lot more conservative. The difference is shown in Figure 28

4.3.4 Adding Hamming MLD + MLD explanation

Maximum likelihood decoding is a simple concept. The probability of each possible codeword s in a codebook $s \epsilon S$ having been sent, given a received signal r is calculated. Then the codeword with the maximum likelihood of having been sent is selected. This is mathematically formulated below.

$$\hat{s} = \arg\max_i \left[P\left(s_i \mid r\right)\right] \tag{29}$$

$$= \arg\max_i \left[\frac{P\left(s_i \cap r\right)}{P\left(r\right)}\right] \tag{30}$$

Now note that, as this will be computed for all of the possible s_is and the relative size is the only thing which is cared about, the $P\left(r\right)$ can be taken out as a common factor.

$$\therefore \hat{s} = \arg\max_i \left[P\left(s_i \cap r\right)\right]$$

Now given that s_i and r are each made up of n bits, and that as the channel here is AWGN, the noise applied to each bit is treated as independent and identically distributed. Therefore Equation 31 is true.

$$P\left(s_i \cap r\right) = \prod_{j=1}^{n} P\left(s_{i,j} \cap r_j\right) \tag{31}$$

Now let n be the Gaussian noise vector which is added in the channel. So Equation 32 can be found from the below equations.

$$\begin{aligned} r &= s + n \\ \therefore n_j &= r_j - s_j \\ \therefore P\left(s_i \cap r\right) &= P\left(N = n_j\right) \\ &= P\left(N = r_j - s_j\right) \\ \text{so as: } N &\sim \mathcal{N}\left(0, \sigma^2\right) \\ P\left(s_{i,j} \cap r_j\right) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{\left(r_j - s_{i,j}\right)^2}{2\sigma^2}} \\ \therefore \text{ from 31: } P\left(s_i \cap r\right) &= \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{\left(r_j - s_{i,j}\right)^2}{2\sigma^2}} \end{aligned} \tag{32}$$

Therefore the final operator for finding the maximum likelihood estimate of s is given by the two equations summarised below.

$$\hat{s} = \arg\max_i \left[P\left(s_i \cap r\right) \right] \tag{33a}$$

$$\text{where: } P\left(s_i \cap r\right) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(r_j - s_{i,j})^2}{2\sigma^2}} \tag{33b}$$

To find the maximum likelihood sent message the operator in Equation 33a is evaluated on every received codeword. Pseudo code illustrating how the performance of this method was evaluated for a set of messaged denoted p is shown below. The exact code can be found in the *Hamming Encoding with MLD* section of the *initial_dnn_comms.ipynb*.

```
# Encode the one_hot_encoded_vectors into bits
s = hamming_7_4_encode(p, G)
# Add noise
n = std * np.random.randn(s.shape)
r = s + n
# Do maximum likelihood decoding
ŝ = ml_decode(r,bpsk(hamming(S)),std)
# Convert ML message out of BPSK
p̂ = bpsk_decode_vec(ŝ)
# Get Block error rate
bler = get_block_error_rate(p, p̂)
```

Note that the process is made more efficient by entirely skipping the linear algebra of hamming decoding. The bpsk and hamming encoded codewords are used for maximum likelihood decoding instead of the codewords, then the encoded messages are mapped directly back to their original message counterparts using a look-up table. The BPSK decoding step was left in as it was very simple and had been vectorised for numpy making it very efficient already.

After the Hamming MLD had been implemented it's performance was assessed over a range of SNRs. It performed very well first time, with the only fault being that it was run over too small a data set giving jaggedly lines at low error rates. The performance can be see in Figure 29.

Part of the reason for it's superior performance is that Hamming MLD can correct some multiple bit errors which Hamming HD can not. For instance, in the case where $s = 0,0,0,0,0,0,0$ and $r = 0.5,0.5,0,0,0,0,0$. With Hamming HD the 0.5s would be rounded to 1s, at which point there would be two incorrect bits which could not be decoded by Hamming HD. But in Hamming MLD there is no rounding and this sequence is still closer by Hamming distance to the $s = 0,0,0,0,0,0,0$ codeword than to all other of the codewords.

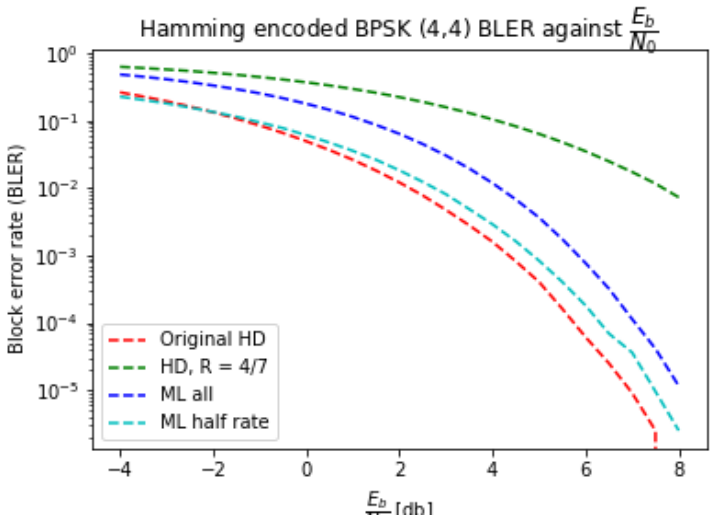


Figure 29: Comparison of all Hamming encoding BLERs.

Out of curiosity the extra performance gain from only transmitting a four bit one-hot encoded vector instead of a a normal four bit vector was investigated. This was interesting because the autoencoders receive M bit one hot encoded vectors instead of k bit normal binary vectors. It was hoped that some parallels might be able to be drawn between the performance gains of the two methods switching between the two.

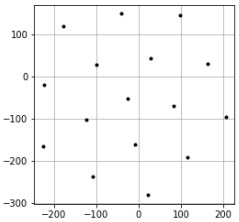
The performance of the MLD only transmitting one-hot encoded vectors is shown by Line "*ML half rate*" in Figure 29. There is a relatively marginal improvement in performance which does not make up for the halfing of the date rate. It should be noted that the autoencoder models are still transmitting the same information content per symbol as the Hamming models they are being compared against as they input a 16 bit one-hot-encoded vector instead of a 4 bit binary encoded vector.

The small improvement in performance suggests that the autoencoder models could likely, given enough training time significantly improve their data throughput by inputting a binary instead of a one-hot-encoded vector, without significantly increasing the BLER.

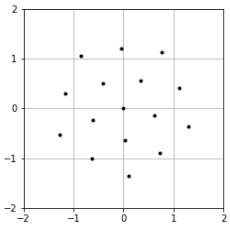
4.4. Reproducing Figure 3a and 3b from the O'Shea paper

4.4.1 Initial (7,4) autoencoder, t-SNE constellation diagrams and (8,8) autoencoder

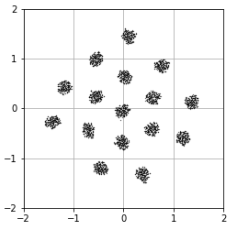
After producing the (4,4) BPSK, (7,4) Hamming HD and autoencoder (2,2) encoding schemes the (7,4) model was produced so that it could be compared against the Hamming HD encoding scheme. This required a different model making function and structure as it could not transmit complex channel symbols. The previous models had transmitted n_c complex channel symbols, however as 7 is an odd number this required a shape of transmitter signals and a new regularisation function. Despite these necessary changes an initial (7,4) model was produced without too many issues.



(a) Unprocessed t-SNE constellation diagram.



(b) t-SNE dimension reduced constellation diagram for a (7,4) autoencoder model.



(c) 200 full sets of all M received symbols with noise.

Figure 30: t-SNE constellation diagrams

The first check on that the autoencoder was performing correctly was to plot the constellation diagram. However this again required a new method. The constellation diagram for the (7,4) model has seven dimensions. So as in the O'Shea paper this was reduced to two dimension using t-SNE. This was done using the *sklearn* libraries *TSNE* function.

However some processing was required to produce the (7,4) tSNE constellation diagrams from [1]'s Figures 4c and 4d. Firstly, despite the input to the tSNE function being normalised to have a constant unit instantaneous power constraint, the output has neither a non-zero power constraint, neither a zero mean. The constellation diagram was not at all stationary but an example of a typical constellation diagram is shown in Figure 30a.

Because of the lack of power constraint the output had to be manually scaled after the t-SNE processing. First the mean was set to zero, and then the signal was scaled to have an average power unit power. At this point a constellation diagram similar to [1]'s Figure 4c was produced, an example is shown in Figure 30b. However it should also be noted that due to the t-SNE being a stochastic method, the constellation diagram changed every time. The central point is very rarely centred on zero and there are often less nice looking diagrams than Figure 30b. It is suggested that a perfect looking constellation diagram cannot be representative of a randomly selected iteration of the model.

Lastly a constellation diagram was produced of the channel symbols received at the transmitter from sending 200 of the full $M = 16$ one-hot-encoded message sets through the AWGN channel. This constellation diagram is shown in Figure 30c. It was found that the small numbers of symbols transmitted with noise were much less likely to converge to a correct looking diagram, however increasing the number of symbols improved this somewhat, although not to the level of the constellations without noise like Figure 30b.

It is interesting that a (7,4) model with constant power constraint, once fed through t-SNE produces the same constellation diagram as [1]'s Figure 4c did with a constant power constraint. This suggests that a similar process is performed by the t-SNE dimensionality reduction techinque.

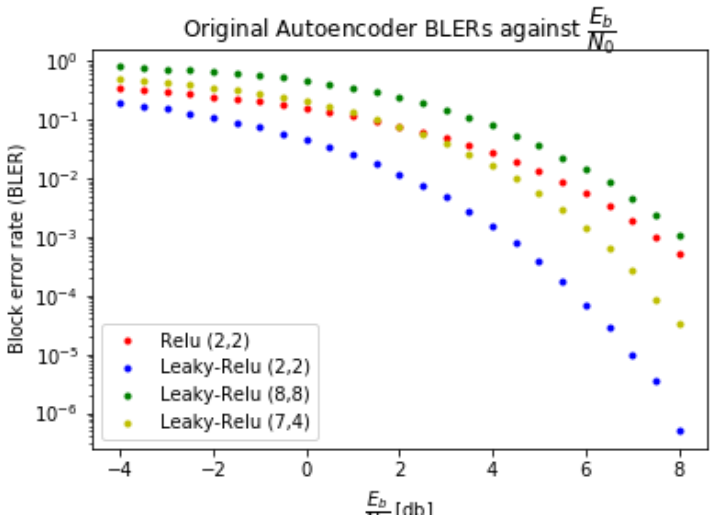


Figure 31: Initial BLER performance comparison of (2,2), (8,8) and (7,4) autoencoder models.

As the (7,4) model produced correct looking constellation diagrams, with the input to the t-SNE reduction being consistent an (8,8) model was produced and their BLER performances assessed. The initial performance of all the autoencoder models, with a relu model added for reference can be seen in Figure 31.

The results from Figure 31 don't look correct. We expect the (7,4) autoencoder to have the best performance, followed by (8,8) and then a larger gap to (2,2). However in Figure 31 the (2,2) autoencoder is far and away the best of the models. Followed by the (7,4) autoencoder and then the (8,8) autoencoder. Also the (2,2) autoencoder model has a BLER $\approx 10^2$ times lower than that achieved in [1] accross the whole range of SNRs, which is suspicious. The (8,8) and (7,4) both significantly underperform in terms of BLER compared to those achieved in the paper. The only thing that does look correct is that *leaky-ReLu* outperforms *ReLu*.

At this point the discovery that Rr was not being set to the correct value mentioned in the two previous sections was made and applied to the leaky-ReLu models. The difference that this made can be seen in Figure 32.

The results after setting the Rr correctly look just as incorrect as the results that preceded them, although differently. Before, the (2,2) autoencoder outperformed the results in the paper, now all three models clearly significantly underperform the results in the paper. The relative ordering is still wrong in the same way, although the shape of the (8,8) autoencoder's BLER looked more promising. This was because it started off with poor performance and then showed strong improvement over the high SNRs, which was what was found in [1]'s Figure 3b.

At the end of this section the (7,4) and (8,8) autoencoder models are clearly significantly underperforming. This was seen as a problem and so was addressed in the section after next (Section4.4.2). However it was noticed that, apart from in extreme cases, it was hard to tell if the (7,4) model was

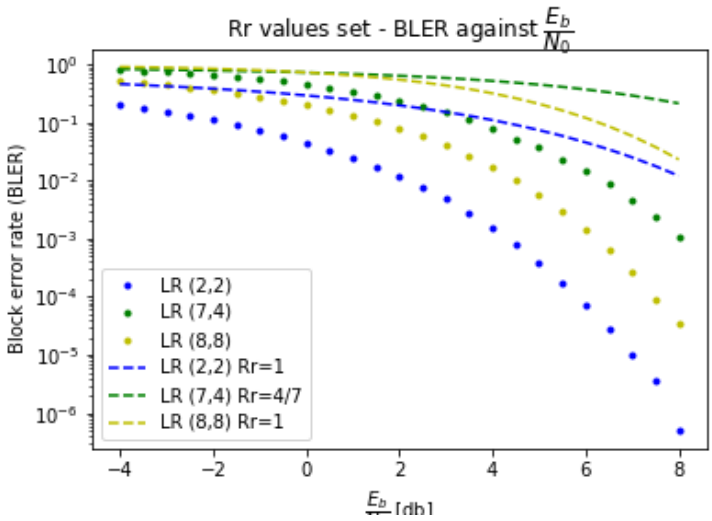


Figure 32: Comparison of BLER performance of (2,2), (8,8) and (7,4) autoencoder models before and after Rr was set to the correct value.

overperforming or underperforming. In [1] the (7,4) autoencoder produced indistinguishable performance from the Hamming (7,4) MLD method. So the next section focussed on producing Hamming (7,4) MLD, in order to be able to better assess performance of the (7,4) model.

4.4.2 Finding errors and improving performance

At this point all of the models had been implemented, so the focus of this section was to find any bugs and improve the models to give equal or better performance to those in the paper. The ordering of the models was the opposite of what would be expected. From best to worst the ordering was (2,2), (7,4) then (8,8), whereas it would be expected (7,4), (8,8) then (2,2). Additionally BPSK was significantly overperforming.

The first place to start was that the (7,4) and (8,8) models were significantly underperforming. It was thought that the more complex models might be underperforming from underfitting due to insufficient layers or an unsuitable activation function. At this point the activation function of all the layers other than the final receiver layer had a *leaky-ReLu* activation function. It was also thought that the layer widths dropping in one step from 256 to 8 for the (8,8) case may be causing the issue. So a general function was written to produce an n layer model, with the last layer having a *tanh* activation function to capture non-linearity. This model also had linearly tapered layer widths to try and improve the fitting of the function.

In the development of the generalised n layer model it was found that due to a bug in the *make_model* function the (8,8) model was actually implementing an (8,4) model. Retifying this error instantaneously improved the validation loss during training at an $\frac{E_b}{N_0} = 7db$ from 1.06 to 0.0750. The performance of

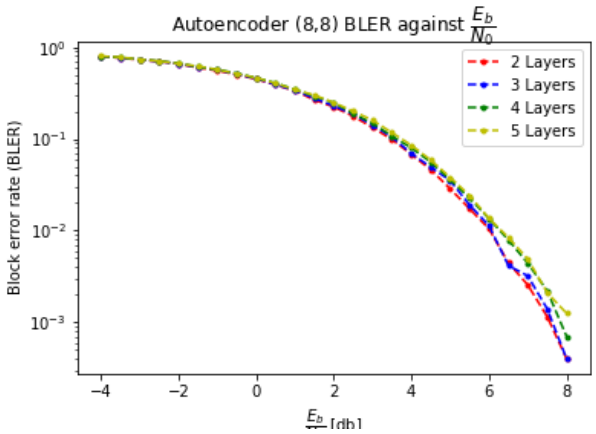


Figure 33: Comparison of different number of layers for the (8,8) autoencoder model.

the different layered models with the (8,4) bug corrected can be seen in Figure 33.

The four and five layer models were seperably worse, and the two layer model very marginally outperformed the three layer model, however the margin negligible. In machine learning it is customary to increase the complexity of the model, and pick the complexity at which the training error stopped significantly improving to avoid overfitting.

However as in this project the data is synthetically generated, exceptionally uniform, the noise is newly generated for each transmission and the data sets used for training and testing are very large there is not much worry of overfitting.

So where normally decisions on hyperparameters would only be made based on performance over the validation set, in this project decisions have consistently been made based on performance on the test set. This section is no different.

Two layers was selected as the number of layers, because it had the best (although very marginally) performance over the test set. Additionally it was the simplest model complexity wise and so trained more quickly and was less likely to overfit.

Due to the indistinguishable performance of the last layer being a *tanh* activation function it was left in as it had a good theoretical backing. The *tanh* function exhibits linear like behaviour for small values, but can also introduce non-linearity, which the *leaky-ReLu* function could not.

After the discovery of the (8,4) bug and the confirmation that the number of layers and lack of non-linearity was not the issue with the (8,8) model, the (8,8) autoencoder was performing better, but still not well enough.

4.4.3 Finding incorrect scaling

The continued lack of performance and easy find of the (8,4) bug led to the (7,4) model being investigated instead. It was realised that the models were being unfairly normalised. All the complex models were

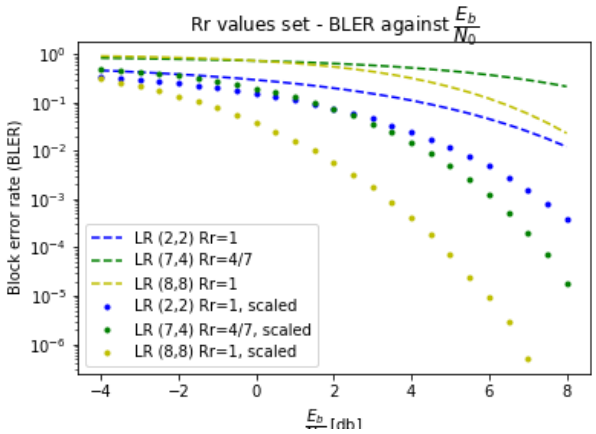


Figure 34: Comparison of performance of all autoencoder models before and after introducing correct scaling.

being normalised such that each complex channel symbol had an instantaneous power of 1. However the (7,4) model was being normalised such that each (7,1) channel encoded message had an instantaneous power of 1. This meant an average power of $\frac{1}{7}$ for each channel bit, whereas the complex (2,2) and (8,8) models had an average power of $\frac{1}{2}$ per channel bit. Going further it was realised that this was again unfair when compared to BPSK, which had a power of 1 per bit.

Due to uncertainty about what the correct average power per channel bit should be a meeting was aranged with the co-supervisor who confirmed that there should be an average power of 1 per channel bit. So a total power of n_r per channel message.

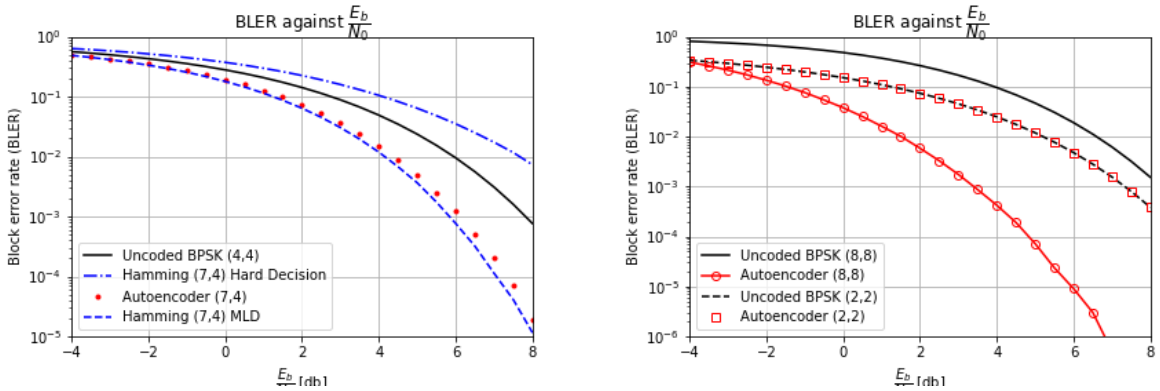
To solve this problem a scaling layer was added after the normalisation layer. The normalisation layer normalised the last axis of the transmitter tensor to have and L2-norm (power) of 1. The following layer then multiplied the channel symbols by a factor of $\sqrt{(N_r)}$ to give a message power of N_r.

In order to get the correct results for all models quickly, training of the models and running of the performance cells was switched to an AWS instance. The instance used was a g3.4xlarge, the hardware details of which are described in Section 2.9. This model has a GPU and 16 CPU cores. This resulted in training models approximately fifteen times faster and allowed the performance loops to be effectively parallelised with a speedup of approximately ten times.

The improvements of performance from introducing correct scaling of the channel symbols can be see in Figure 34. The results show that the larger models were being unfairly penalised as the (2,2) autoencoder marginally improves, whereas the (7,4) and (8,8) models both significantly improve by factors of $\approx 10^5$ at the high SNRs.

The (8,8) model worked first time, however the (7,4) took a little longer to get working as there was a bug where the scaling layer was being skipped.

After this correction the (7,4) model performs more or less inline with what it should according to



(a) Final reproduction of [1]'s Figure 3a, before adding BCH codes and redebugging Hamming HD.

(b) Penultimate reproduction of 3b, with overperforming (8,8) model due to incorrect scaling.

Figure 35: t-SNE constellation diagrams

[1], as does the (2,2) model. However the (8,8) model far outperforms it's counterpart from [1]. In fact the (8,8) model had to be rerun on the AWS instance with larger data set to give a smooth curve at such low error rates.

At this point the next iteration of [1]'s Figure 3a and 3b were plotted. The results can be seen in Figure 35.

4.4.4 Final corrections and BCH coding

At this point after submitting a draft of the first two sections of the report to the supervisors the returning advice was to compare the performance against BCH codes instead. Consequently BCH codes will be investigated later in this section.

The current differences between [1]'s Figure 3a and 3b, comared to Figure 35a and b are that the (8,8) model significantly overperforms and the Hamming (7,4) HD significantly underperforms. The first of these to be investigated was the (8,8) autoencoder.

It was noticed while writing up Section 4.4.3 that the reason for this was the scaling being incorrectly applied to the (8,8) complex model. As the L2-normalisation is applied to the last dimension, the same method cannot be applied to the (7,4) non-complex and (8,8) complex models.

The (7,4) model's channel symbols must be multiplied by a factor of $\sqrt{N_r}$ because the last dimension is the (7,) dimension. However the last dimension in the complex model is the complexity dimension of shape (2,). So the channel symbols must be scaled by a factor of $\sqrt{2}$. This means that the performance of the (2,2) model is correct by chance, however the symbols of the (8,8) model are being scaled up incorrectly by an extra factor of 2.

This was rectified giving the performance shown in Figure 36.

After correcting the incorrect scaling of the (8,8) autoencoder to a factor of $\sqrt{2}$ the results graph looks

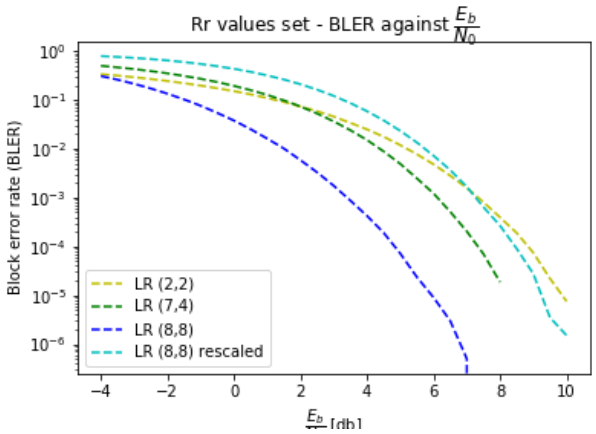
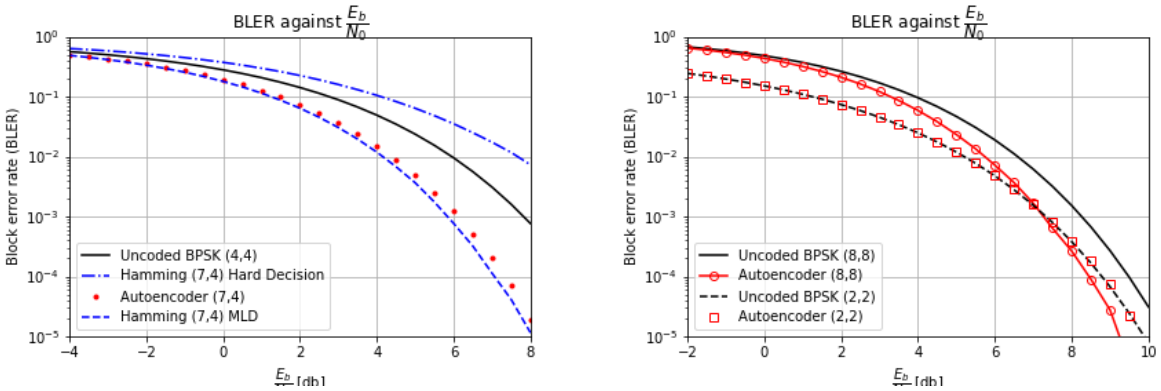


Figure 36: Comparison of performance of all autoencoder models after (8,8)s scaling was corrected.



(a) Final reproduction of [1]'s Figure 3a, before adding BCH codes and redebugging Hamming HD.

(b) Final reproduction of 3b, three lines identical, (8,8) autoencoder slightly underperforming.

Figure 37: t-SNE constellation diagrams

as it does in Figure [**?**]. This is now very similar to Figure 3a from [1]. Three of the lines are identical, the (2,2) autoencoder and both the BPSK models. The (8,8) autoencoder line is similar in that it starts in tangent with the (8,8) BPSK and finishes with the lowest error rate. However it slightly underperforms it's counterpart from [1]. It crosses the (2,2) lines at ≈ 7 as opposed to ≈ 5.5. The BLER seems to be roughly a factor of 5 out. This suggests there may be some small change to the architecture that can be made to improve performance. However the number of layers and activation functions have already been varied. One option left is dropout regularisation. This is where some fraction of the neurons are switched off randomly at training time, which is done primarily to avoid overfitting. However this hasn't been trialed so far because the data sets are very uniform, very large, and the noise is newly randomly generated every time the model transmits a message. Because of this it seems unlikely that the model could overfit.

Additionally the (7,4) autoencoder in Figure 37a is also slightly underperforming at the high SNRs. As this margin is so small it will be tackled by retraining and using the ModelCheckpoint callback to ensure that the model with the best validation loss achieved during the training process is saved. Also the model will be retrained over a large number of initialisations to remove any small dependence on initialisation values. This has not been used before due to the very large time commitment that is needed for training a large number of models fully.

The last inconsistency in Figure 37 is that the Hamming HD encoding is significantly underperforming. A reasonable amount of time has been committed to debugging this and the underlying BPSK, however it seems to perform correctly in all unit tests and have the correct amount of noise being applied so the author can not see what is being implemented incorrectly.

BCH codes were investigated on the direction of a project supervisor. BCH codes are a generalisation of Hamming codes for multiple error corrections, where the number of errors which can be corrected is specified as a variable design parameter t. For any integers $m \geq 3$ and $t < 2^{m-1}$ there exists a BCH code with the characteristics described in Equations 34 where k is the number of data bits. Note that the minimum distance is ofter referred to as the designed distance.

$$\begin{aligned} \text{block length: } & n = 2^m - 1 \\ \text{parity bits: } & n - k \leq mt \\ \text{minimum distance: } & d_{min} \geq 2t + 1 \end{aligned} \tag{34}$$

The (7,4) Hamming code examined above is a special case of the BCH code where $m = 3$ and $t = 1$. It would have been interesting to look at the next BCH code up where $k = 4$ and $t = 2$, or in the same ratio. The *python-bchlib* library [42] written by J.Kent was used for encoding and decoding. It seemed a good idea to start with the $m = 3$ and $t = 1$ case as it is the same as Hamming (7,4) so the results should be easily comparible.

The implementation code for this section can be found in the *BCH Encoding* section of *initial_dnn_comms.ipynb*. It required a seperate noise generation section function as the *bchlib* encoding functions output bytearrays instead of numpy arrays. The noise function is called *gaussian_bitflip*, It works on the basis that the noise is AWGN distributed, so has the pdf shown in Figure 38. The red highlighted area from Figure 38 represents the probability of error for a transmitter BPSK signal. The *gaussian_bitflip* function takes in normal binary encoded signals and assumes that they would have been encoded into BPSK and then passed through an AWGN channel. So p_e is given by Equation refeq:GaussianBitflip.

$$\begin{aligned} p_e =& P(X \leq -1) \\ =& CDF(-1) \end{aligned} \tag{35}$$

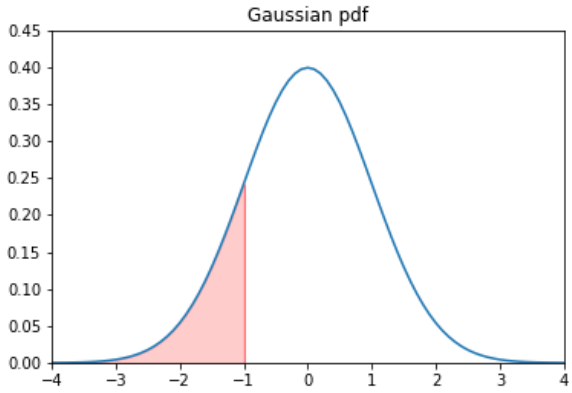
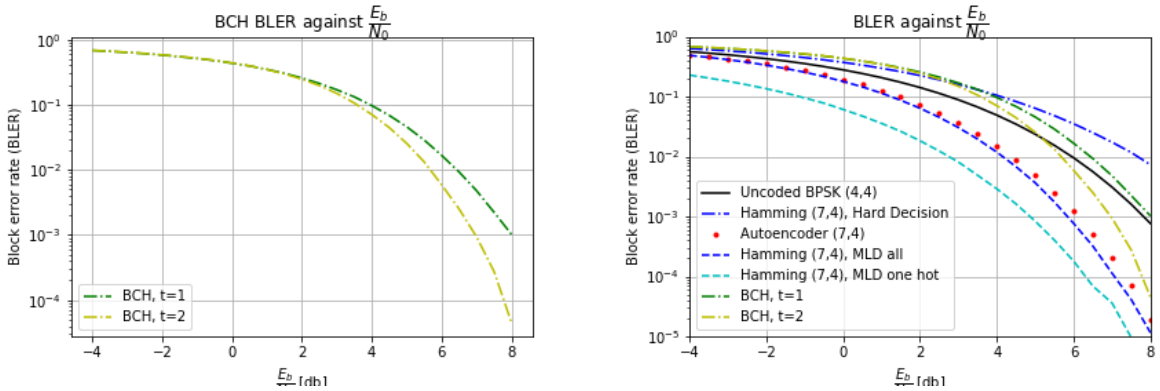


Figure 38: Gaussian pdf for illustration.



(a) A figure to show the performance of two BCH codes with $t = 1$ and $t = 2$ respectively.

(b) The reporduction of [1]'s Figure 3a with BCH encoding added for extra comparison.

Figure 39: t-SNE constellation diagrams

So the function goes through every bit sent, and with probability p_e flips the bit by XORing (exclusive or'ing) with $1 << \text{bit\_number}$, which is the number one logically shifted left by bit_number bits. This function therefore directly replicates the effect of encoding into BPSK, adding gaussian noise, picking the most likely symbol and decoding back to binary. Using this function the two lines in Figure 39a were produced to compare the BLER of a BCH code with $t = 1$ and $t = 2$.

As expected the $t = 2$ line outperforms the $t = 1$ line. Although as mentioned above the $t = 1$ line should give identical performance to the Hamming (7,4) line. As mentioned earlier the Hamming (7,4) line in Figure 37 underperforms. From Figure 39b it can be seen that the $t = 1$ line outperforms the Hamming (7,4) line on the whole, however it still underperforms what the Hamming (7,4) line should produce according to both [1]'s Figure 3a and also the common sense that adding Hamming encoding should not decrease the performance from normal BPSK.

Despite outperforming the experimental Hamming (7,4) encoding's BLER the BCH codes still do not pass this common sense test of improving the BLER of the uncoded BPSK. This shows that there must be an error in this section. A suspected causes of this error was that the BCH polynomial prime used (8129) was unsuitable. Sadly as this was investigated in response to feedback and so the implementation was done just a couple of days before the report was due there was not enough time to properly debug this section.

It was suggested in a second meeting the day before the deadline that BCH encoding could be easily implemented in Matlab, however as the rest of the project was already implemented in python, the author wasn't recently familiar with Matlab and the deadline was the following day there was not enough time to investigate this option.

4.5. Paper 2 - Reinforcement learning, RBF and RF

4.5.1 Rayleigh Fading

Rayleigh fading is a type of fading which models a channel in which there is not a clearl line of sight between the receiver and the transmitter, this is a good approximation for signal propogation through the troposhpere and ionoshpere as well as built up areas like cites [43]. The fading implemented in this report is assumed to be flat and slow. The large and small scale fading are assumed to be stationary. Rayleigh fading is implemented by the equation below.

$$\begin{aligned} \mathbf{r} &= \mathbf{s} * \mathbf{h} + \mathbf{n} \\ \text{where:}\mathbf{h} &= h_r + jh_i \\ \mathbf{n} &= n_r + jn_i \\ n_r, n_i &\sim \mathcal{N}(0, \sigma_a^2) \\ h_r, h_i &\sim \mathcal{N}(0, \sigma_m^2) \end{aligned} \tag{36}$$

$\mathbf{s}$ and $\mathbf{r}$ are the complex sent and received signals respectively. In all types of rayleigh fading a different realisation of n is used for every sent signal, but the same does not apply to h. In RBF (rayleigh block fading) the multiplicative factor h stays the same for the whole block of sent messages. In rayleigh slow fading (RSF) h changes every message that is sent but remains the same for each bit of the message, whereas rayleigh fast fading (RFF) would have both h and n changing during the sending of each symbol, so each symbol would be transmitter over multiple fades. The Aoudia paper examined RBF.

Rayleigh fading channels are a good type of channel to examine the performance of the learned communication system. The reason for this is that, as described above they model well the effect of a signal propogating through a built up area like a city. A city with a very complex, but stationary large scale fading function would be an environment where a learned communication system could perform

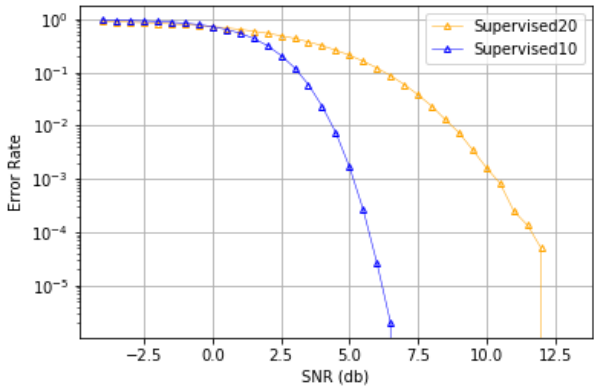


Figure 40: Reproduction of [2]'s Figure 6a, only supervised line currently present.

particularly well. This is because it could learn an optimal communication system for taking advantage of these large scale fading effects in a way that a normal communication system could not.

4.5.2 Investigation into RSF

The first objective was to reproduce the supervised AWGN graph from [2]. This was very easily done by simply using the (8,8) model from the previous section to produce Figure 40.

COMMENT ON WHETHER THIS LINE LOOKS LIKE THE ONE IN THE PAPER.

After producing the supervised AWGN model it was decided to investigate RSF before RBF. This was because they required very similar noise layers, but the RSF model would have a significantly simpler architecture. This architecture can be seen in Figure 41. The architectures of the individual blocks can be seen in Figure 42

These architectures show the addition of an expert feature as an advancement from the AWGN supervised models. This is the *HEstimator* section, which makes an estimate of the $\mathbf{h}$ parameter from Equation 36 for that message. There is then a *y_over_h* layer which implements complex division of the received signal by the output of the *HEstimator*. Note that the received signal has shape (batch_size,N_c,2) and estimated h has shape (batch_size,2). So a $\hat{h}$ is found for each overall message which is sent, and then all four of the transmitter complex channel symbols are divided by this factor.

The transmitter architecture has also changed, the new architecture can be seen in Figure 42a. The previous architecture had a single dense *leaky-ReLu* layer and a single dense *tanh* layer. The new arechitecture starts with an MxM embedding layer for feature extraction. This allows each one hot encoded input to activate M different nodes which can then all be used as inputs to the function of the transmitter network. This embedded layer has an exponential linear unit (ELU) activation function. The ELU activation function is defined below in Equation 37.

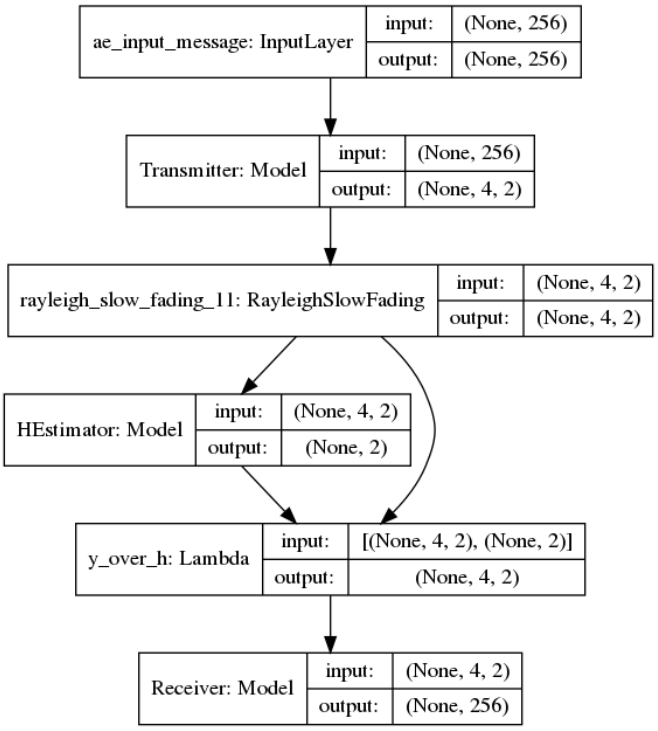


Figure 41: The architecture of the whole end-to-end system of

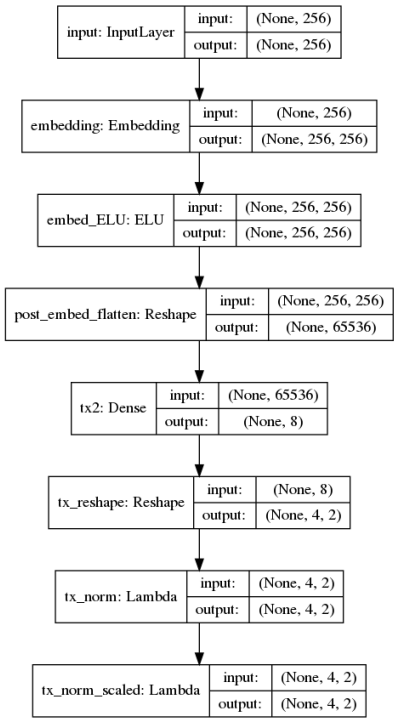
$$ELU(x) = \begin{cases} x, & \text{if } x \geq 1 \\ \alpha(e^x - 1), & x < 0 \end{cases} \tag{37}$$

This activation function is good because it can both learn linearity and non-linearity depending on if the weights are trained to make the input greater or less than zero. Next there is a single dense layer with a *linear* activation function. There are also a couple of reshape layers in the transmitter to make flatten the input before the dense layer and to make the channel symbols complex.

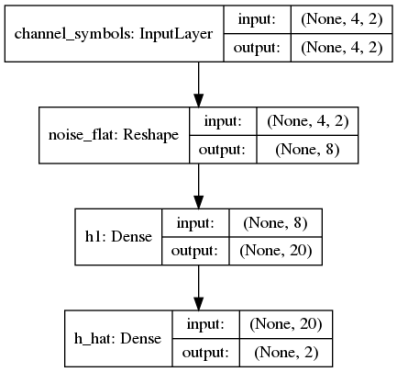
The transmitter had a different normalisation scheme as well. The previous normalisaiont scheme was a constant power constraint (L2-Normalisations) of one per channel bit described in Section 4.2.1. The new architecture had an average power constraint defined below in Equation 38.

$$\mathbb{E}\left[\frac{1}{N}\left\|x\right\|_2^2\right] = 1 \tag{38}$$

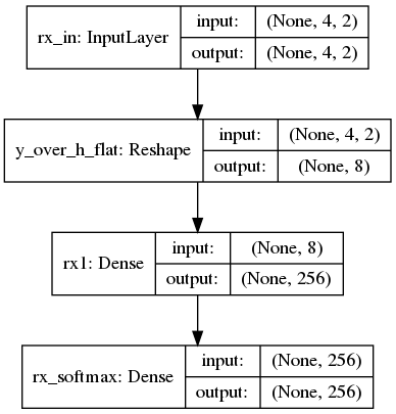
This allows the transmitter more flexibility, as if a constant power constraint is optimal it can converge to that from this power constraint. However it also has the options of giving certain channel symbols more power. If the input symbols were unequally frequent then more common symbols might be expected to have more power and less symetric constellation diagrams would be allowed by this power constraint. However in this case each of the messages within M is equally likely.



(a) A block diagram of the RSF transmitter architecture.



(b) A block diagram of the RSF $\hat{h}$ estimator architecture.



(c) A block diagram of the RSF receiver architecture.

Figure 42: Block diagrams showing the architectures of all the submodels of the RSF autoencoder model shown in Figure 41.

The receiver architecture has changed as well. Before the receiver had three dense layers with *tanh*, *leaky-ReLu* and *softmax* activation functions respectively. Now the receiver has only two dense layers, with *relu* and *softmax* activations respectively. This new architecture is again identitical to the architecture used in [2].

The h-estimator layer structure is also taken from the architecture in the paper. It has two dense layers, one with a *tanh* activation function and a second with a *linear* activation function.

Note that throughout this section the SNR will define the standard deviation of the additional noise n, the multiplicative noise was left constant with a $\sigma_m = \frac{1}{\sqrt{2}}$ as follows convention. This is the conventional value because it gives an average unit channel gain from the multiplicative noise. This value was used because it is convention, however it should be noted that it is unrealistic as in a real channel it would be expected that the received power be almost always less than the transmitter power. In accordance with this idea $sigma_m = 0.33$ was also trialed, by the theory that the channel gain being less than one 99.7% of the time seemed more realistic. However it was found that actually the exact σ_m made very little difference as the weights just scaled to a be larger for a lower σ_m giving no noticable difference in performance. Also note that as the power constraint is Equation 38 the SNR is simplified according to

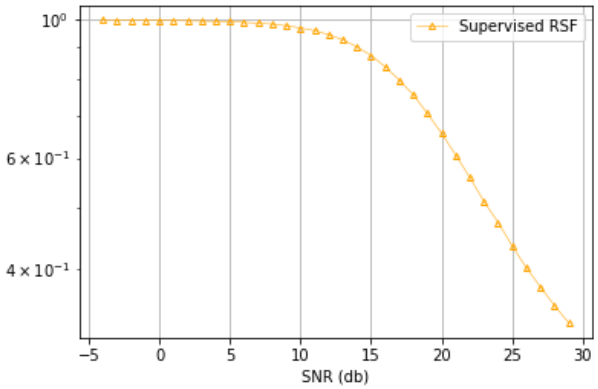


Figure 43: The poor BLER performance of the supervised RSF autoencoder.

the equations below.

$$\begin{aligned} SNR &= \frac{\mathbb{E}\left[\frac{1}{N}\left\|x\right\|_2^2\right]}{\sigma_a^2} \\ &= \frac{1}{\sigma_a^2} \end{aligned} \tag{39}$$

This architecture describes above, despite being identical to the architecture of the (8,8) supervised autoencoder in [2] performed extremely poorly. The results are show in Figure 43.

The BLERs in this graph range from 99.55% at the low SNRs to 31.78% at the higest SNRs which is clearly extremely poor and nowhere near the performance shown in [2]'s Figure 6b which ranges from ≈ 1 to $8 * 10^{-3}$. To try and combat this more dense layers were tried with many combinations of activation functions. Two, three and four dense layers were tried with all *tanh*, all *leaky-relu* and all *relu* activation functions. However none of these had improved performance. Several combinations of *leaky-relu* and *tanh* activation functions were tried. Training at the higher SNRs of 40db, 80db and no noise were all tried as well. Training until convergence ($\approx 10 - 20$ epochs) at each SNR and then reducing the SNR was trialed, despite the fact this could not be done with a real channel. However none of these methods reduced the validation loss by more than a couple of $\%$.

An example of four of the combinations of different activation functions can be seen in Figure 44. Here all three of the combinations different to that used in the Aoudia paper all underperform it. The *2l-lin-lin-relu* describes a model, with two layers in the *transmitter* and *receiver* blocks. The first *lin* refers to the activation function in the hidden layers of the receiver and the non-first layers in transmitter. The second *lin* refers to the activation functions of the non-first layers in the *h_estimator* block, the *relu* refers to the activation function of the final layer of the receiver prior to the softmax layer.

About four days was spent exclusively trying to improve the performance of the supervised RSF autoencoder, with no success. This suggests that learned communication systems perform poorly in as

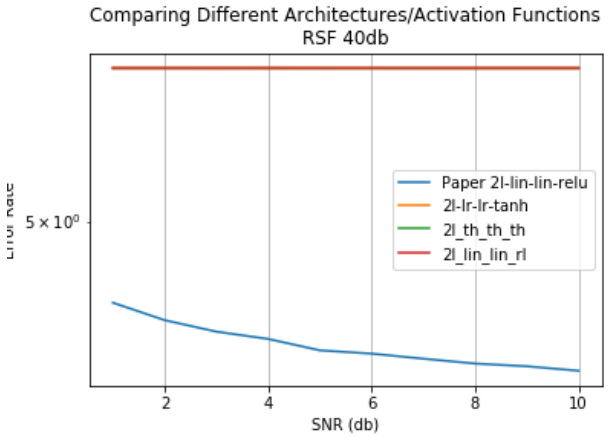


Figure 44: Examples of some different tried activation function combinations when trialing two layers.

non-stationary an environment as RSF. It makes sense that the communication system should perform worse in a RSF environment than an RBF environment. This is because in the RBF environment the h factor from Equation 36 stays the same for the whole block, which could be up to 1000 messages. This means that h can be estimated a lot more accurately as the variance of the estimate $\hat{h}$ decreases with $\sqrt{n}$. After the factor h is removed the noise n can then be removed a lot more effectively. Whereas in RSF h only stays the same for the channel bits of a given message, which is too few to estimate once the additive noise n has been added.

The inability to estimate the h factor of the channel noise effectively makes removing the RSF very difficult, as the multiplicative noise is much more damaging than the additive noise.

This in itself is a novel result, as far the author can see no other paper has investigated the performance of an end-to-end learned communication system in the presence of slow or fast rayleigh fading. The very poor performance of the learned communication system in RSF compared to RBF suggests that the learned communications system would perform even more poorly in a RFF channel and highlights an previously unproven weakness in learned communication systems.

4.5.3 Investigation into RBF

RBF can be seen as "*very slow*" rayleigh fading. After the implementation and testing of the RSF fading communication system above the implementation of a supervised RBF communication system was started. To reproduce the results in the paper the BLER would have to improve by a factor of $\approx 10^2$, however given the reduction of the variance of the $\hat{h}$ estimator should improve by a factor of $10^{\frac{3}{2}}$ and that the h factor was the main damaging factor of the rayleigh noise there was reasoable confidence that this should be achievable relatively easily.

As locally-optimal receivers and transmitters had been learnt in the RSF section the main task was to implement an architecture which could encode and decode many messages identically in parallel, but

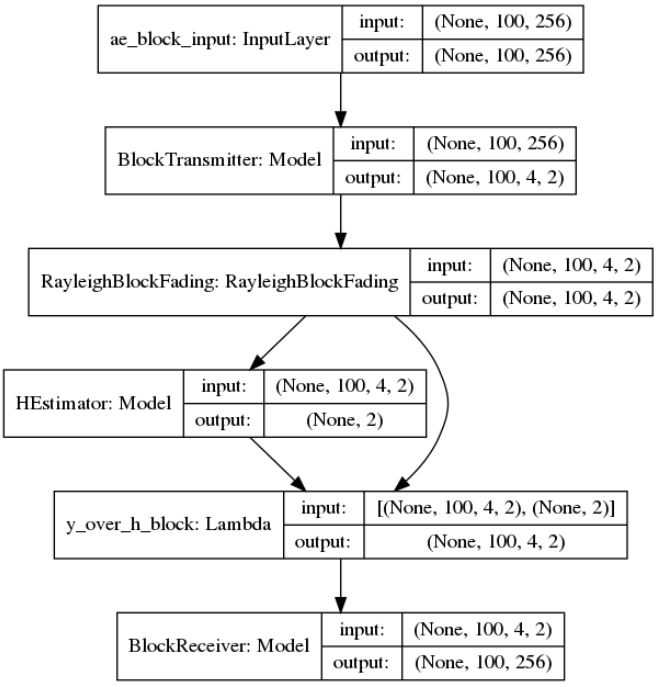


Figure 45: RBF autoencoder overall architecture with an example block size of 100.

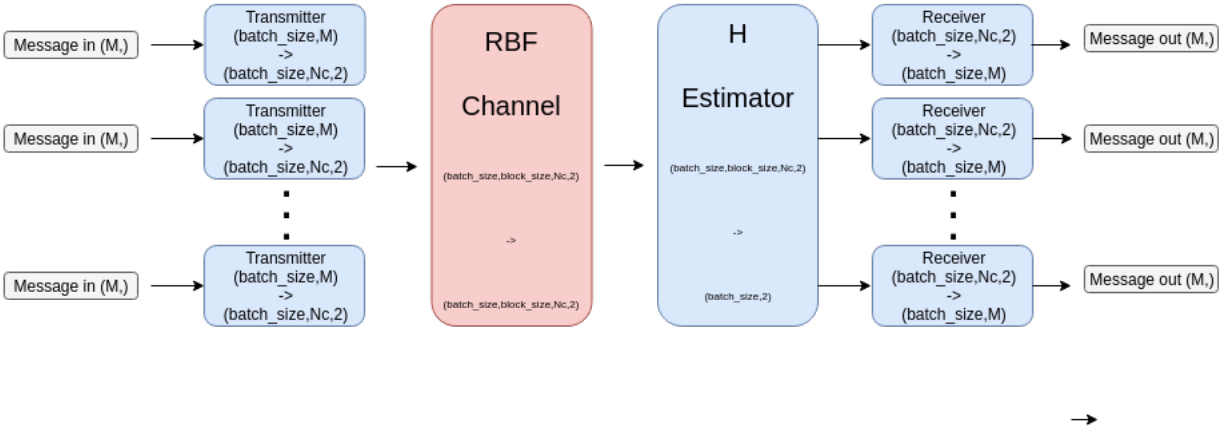


Figure 46: Alternative representation of the RBF autoencoder overall architecture, showing the make up of the BlockTransmitter and BlockReceiver sections.

add noise to all of them and estimate h to all of them in one go.

The architecture of the proposed model can be seen in Figures 45 and 46. Figure 46 gives a more detailed description, showing how the *BlockTransmitter* and *BlockReceiver* blocks from Figure 45 are made up of *block_size* copies of the single transmitter or receiver in parallel. It should be stressed that these are the same transmitter with the same weights which should be updated in the same way. Note that the number of parameters of the *HEstimator* block depends on the *block_size*.

This architecture is very awkward to implement, because it requires blocks to be made up of many repeated subblocks in parallel, with the same weights shared for all the parallel sub-blocks. It must also be able to update the weights of all three learned modules.

It took a very long time to be able to form a model which would compile and could fit at all with a non-*None* gradient. The *transmitter* and *receiver* had to be put into seperate models and the receiver had to be split into three parts. It had seperate parts for the h estimating model, the complex division lambda and the decoding part of the receiver. Then a Lambda layer was used to implement the *BlockTransmitter* and *BlockReceiver* functions. The lambda function was implemented four different ways before it gave a non-*None* gradient. The lambda ended up in the form of a list of the encoded channel symbols being made using a for loop, followed by a Keras Concatenate layer.

The debugging was particularly confusing, as each of the individual models worked and gave non-*None* gradients, it was only when the *BlockReceiver* submodel was combined with the *y_over_h* lambda layer that the *None* gradient would appear. Also as the model could not be fit, the gradients with respect to given weights could not be found, which meant it could not be debugged line by line.

The final implementation with the Concatenate layer compiled, gave a non-*None* gradient, however the only parameters which the overall model could see were those of the HEstimator section, meaning the model did not work. It could not see the weights of the transmitter and receiver blocks, so did not update them by back propogation. This meant that despite the model being able to estimate h significantly more accurately due to the larger block sizes, it did not significantly improve from the baseline validation loss as the receiver and transmitter blocks were not being trained.

Another issue with the RBF model was that with large block sizes it became extremely slow to initialise and train, meaning that large block sizes would not be able to be effectively investigated. Table 3 shows the characteristics of the RBF models made with different block sizes. This table illustrates the prohibitive effect on latency of increasing the block size. The ? given for the training s/step and block size of 1000 is there because the model was sat for 20 minutes and it never starting to fit and it never showed it had started the first epoch, so the training s/step could not be found. However the ball park of prohibitively slow as found.

Block Size	Compile Time (s)	h_est parameters	training s/step
10	4.75	$1,662$	$173\mu s$
100	33.45	$16,062$	?
1000	$1,241.67$	$160,062$	?
n	n/a	$8 * 20n + 62$	n/a

Table 3: Activation function performance statistics from ten initialisations

Because of the issues listed above the investigation of an supervised autoencdoer model for an RBF

channel was a failure. It is hoped that were the issue with the *transmitter* and *receiver* weights not updating to be resolved the performance would be competitive with that shown in the Aoudia [2] paper, however it is hard to know in advance of producing it.

5. Results and Discussion

- I mainly produced the same results as them.
- The final two graphs from the first paper
- The results from the RSF section

This section will go over the end results of the first and second set of deliverables.

6. Conclusions

6.1. AWGN autoencoder and unlearned communication systems comparison

- I couldn't replicate their performance with their architectures. I found that they underperformed their results.
- I reproduced their results mainly.
- I got different results for Hamming (7,4) and BCH.
- I confirmed that a learned communication system can replicate the performance of Hammign (7,4) MLD, which is an efficient coding scheme.
- I confirmed that a learned communication system could learn QPSK, 16-QAM and an unusual pentagon like constellation diagram, with no prior knowledge of encoding schemes.
-

The main conclusion of the first section of deliverables from the report is that

6.2. Novel investigation of learned communication system over an RSF channel

One novel area investigated in this report was the performance of supervised learned communication systems in the presence of Rayleigh slow fading. The performance of autoencoder-like learned communication systems had been investaged over Rayleigh block fading channels, but as far as the author can find, no paper has investaged Rayleigh slow fading. Contrary to the name, RSF is actually faster fading that RBF. RBF can be thought of as "*really slow*" rayleigh fading.

The meaning of the speed of the fading is how long the h parameter from Equation 36 remains constant. In slow fading there is a new realisation of h for each message sent, in block fading h remains the

same for the whole block of messages, and in fast fading each channel symbol would be transmitted over multiple fades. It was found in this paper that learned communication systems perform very poorly over an RSF channel, as the N_r bits sent for each message are not enough to effectively estimate h. As the multiplicative noise is far more distruptive than the additive noise not being able to estimate $\hat{h}$ accurately prevents the communication system from removing the additive noise effectively and so communicate with a reasonable bit error rate. What can be learnt from this is that not only do learned communication systems perform poorly over channels as non-stationary as an RSF channel, but they would also perform even worse over an fast fading rayleigh channel. Consequently this area can be ruled out of further research.

The normal way of dealing with fast fading is called waterfilling, which involves transmitting only when the channel is not in deep fade. (See Section 2.5.1 for more explanation). It is theorised that a better way of combatting a fast fading channel would be to use an RNN, which can learn any algorithm, as then if waterfilling is optimal, it could learn it. However this is a subject for another project as it would require substantial work.

7. Future Work

There is a large amount of interesting future work which could be carried out, some of which is completing the original deliverables which were not covered in this report. However a large amount of the potential future work is examining the performance of this new technology in promising, currently untested areas.

Starting with the incomplete deliverables, it would be desirable to get the RBF autoencoder working. This would likely give results immediately with very little hyperparameter tweaking. It would also be useful to produce the reinforcement learning models for the AWGN and RBF channels. This is especially important as this is the promising part of the technology. Without the reinforcement learning a differentiable channel model is needed, which significantly detracts from the usefulness of the technology.

Moving onto non-deliverable future work, it would be interesting to examine how the BLER of the RBF learned communcation system deteriorates as the block size is decreased. This could give a practical guide for what kind of ratio is needed between the T_{cs} and the T_{coh} of a channel (how slow the fading needs to be) for a learned communication system to be a viable option.

Another idea which could be investigated is, in the RBF model, assessing the accuracy of the estimation of h with respect to the length of the sequence used to estimate it. If it was found that near comparible accuracy of estimation could be achieved with a small fraction of the block size, then this could be sent as a pilot, in which the h parameter was estimated. Then after this the rest of the block would not have to pass through the h estimator section. This would significantly reduce the complexity of the model. Also as requiring the h estimator section take all the sent bits as an input produces a significant latency bottleneck this could massively reduce the latency of the process.

Some extra comparisons which could be done are to check that the developed methods are truly competitive with state of the art are comparing the learned systems to Shannon's theoretical capacity limit. The performance could also be checked against and current long block length state of the art codes, such as turbo and LDPC codes for an AWGN channel. This would show the performance which is being sacrificed by not using long block lengths.

To better assess the generality of a learned communction system learned networks and current state of the art could be compared over common fading environments such as Rayleigh and Ricean channels across a range of type of fading, large scale or small scale, fast or slow, flat or frequency selective. A wide scale assessment of the performance of learned networks over many different types of channels is currently missing from the literature. This is quite important to do as learned communication systems are likely to be have impressive performance in environments with complex stationary large scale fading as they can adapt to give the locally optimal communication system. After which, more interestingly the same could be done for some rare or hard to model channels. These being where the learned communication system should have the biggest advantage and so it would be interesting to see if they could deliver on their potential.

A final idea potentially worth investigating would be to add an RTN layer to the unsupervised reinforcement learning models. In [1] the RTN autoencoders consistently outperformed the normal autoencoders for all SNRs. The unsupervised models are very promising, so it would be interesting to see if there performance could be improved by adding the RTN networks.

A. Appendix

B. Ethical, Legal and Safety Considerations

B.1. Ethical considerations

One interesting ethical consequence of this project is that, if this technology were to become very successful and so very competitive with or superior to current state of the art communications systems, it could reduce the number of jobs for communications engineers. If it was possible to just set up a generic transmitter and receiver, then let them train for their environment and then just have them run at optimal performance, there there would not be as much of a need for communications engineers to design systems.

While this would be bad for current communications engineers, it would lead to greater productivity and efficiency by companies which would contribute to greater economic growth, which would then trickle back down. So while it may be bad for a minority, it should be better for the majority, therefore overall it is ethically alright. However other than this there are no significant ethical implications of this project.

B.2.1 Infringing patents

A search was performed on Espacenet, which was found by direction from the UK government website. This seach of Espacenet was to see if there were any parents which this project might be infringing. Patents were searched for using related key words, however no patents of similar ideas were found. Consequently the project is likely safe from infringing any patents. If there were any significant commercial consequences of this report then a more thorough search could be carried out through a patent attorney, however this would be expensive and is unlikely to be needed.

The legal alternative to patents for protection of intellectual property is referred to as a ”trade secret”. However these are only legally binding for company employees who have signed a non disclosure agreement (NDA), or if someone is accessing the information either when they do not have the permission to access it, or they are receiving it from someone who doesn't have the permission to access it. Trade secrets become non-binding once the information is publicly available, so as the papers that this paper is reproducing are publicly available it cannot be infringing any trade secrets that the author is unaware of.

B.2.2 Use of data

The other potential legal concern for a project of this type is potentially using someone's data inappropriately while training a machine learning model. For the first part of this project this isn't possible as the model involves training an autoencoder-like model. In this method the desired output is the initial input, and that input is any one of the potential combinations of n binary bits. This is where n is the block length. Hence there is no involvement of non-self-generated data and so can be no illegal use of someone else's data.

In the last of the main deliverables where would be developing the reinforcement learning model which could be trained on any samples from any channel there is marginally more of a risk of this. To counter this risk it will be made sure that these samples are sourced from publicly available sources, so people who have contributed to it will have signed that they consented to having their data collected.

However it should be made clear that it should be more or less impossible to decode the information that people were sending across a channel from the channel interference samples. To ensure there can be no accusations of this the data and model will be kept private, results will not be shared publicly and there will be no inspection of the channel interference.

B.2.3 Licences of used libraries

The Keras library has an MIT lisence, which means that users have the right to ”use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software” as long as a copyright notice is place in the code. To ensure that this is abided by, if the software from this project is published then

Keras will be acknowledged and referenced. Additionally the copyright stamp will be included in the code if it is published.

B.3. Safety Plan

Firstly it should be stated that compared to hardware projects there are no really pressing safety issues. However the relevant safety issues are assessed in Table 4.

References

[1] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Trans. on Cogn. Commun. Netw.*, vol. 3, pp. 563–575, July 2017.

[2] F. A. Aoudia and J. Hoydis, "End-to-end learning of communications systems without a channel model," *Nokia Bell Labs*, Apr. 2018.

[3] A. Goldsmith, ""joint source/channel coding for wireless channels."," *Proc. IEEE Vehicular Technol. Conf.*, vol. 2, pp. 614–618, 1995.

[4] A. Manikas, "Lecture 4: An overview of fundamentals: Wireless channels," in *Imperial College EE303 Communications Systems Course*, Nov 2017.

[5] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," in *2016 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pp. 223–228, Dec 2016.

[6] T. Schenk, ""rf imperfections in high-rate wireless systems: Impact and digital compensation."," *Springer Science & Business Media*, 2008.

[7] E. Zehavi, ""8-psk trellis codes for a rayleigh channel,"," *IEEE Trans. Commun.*, vol. 40, no. 5, pp. 873–884, 1992.

[8] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," *J. Comput. Syst. Sci.*, vol. 50, no. 1, pp. 132–150, 1995.

[9] S. Reed and N. de Freitas, "Neural programmer-interpreters," in *International Conference on Learning Representations (ICLR)*, 2016.

[10] Stinchcombe and White, "Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions," in *International 1989 Joint Conference on Neural Networks*, pp. 613–617 vol.1, 1989.

[11] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic," in *IEEE journal*, pp. 77–84, 12 2016.

[12] T. Simonite, "Apple's latest iphones are packed with ai smarts," *Wired Magazine*, Dec 2018.

Risk	**Potential Cause**	**Likelihood (/5)**	**Potential Harm (/5)**	**Mitigation Plan**
Repetitive strain injury (RSI)	Excessive use of laptop	4	1	I will take regular breaks every 20-60 minutes. If I get RSI then I will identify what movement is causing it and stop this movement. I will also treat it with hot and cold treatment, rest and anti-inflammatories like ibuprofen. []
Eye Strain	Excessive use of laptop	4	2	To avoid straining my eyes I will try to work with adequate, but not too bright, lighting at all times, and will rest my eyes every twenty minutes. I will also try to keep the screen approximately 20-24 inches away from my eyes and 10-15 inches below my eye line. []
Back Pain	User's poor posture	2	2	I have set my chair at my work station up according to this NHS guide [] and a will make more effort to sit with correct posture to avoid back pain which might prevent me from working.
Trip Hazards	Often work with laptop plugged in	1	2	I will make sure that my charger lead is as short as possible and does not stretch across any areas where people regularly walk.
Electric shocks	Often work with laptop plugged in	1	5	I will be careful when plugging and unplugging my laptop to avoid shocking myself.
Fires	Often work with laptop plugged	1	5	I will not leave my laptop plugged in unsupervised. I have also checked that my laptop charger has been PAT tested and so I think I have taken reasonable steps.

[13] I. Hubara, M. Courbariaux, D. Soudry, R. El- Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *arXiv e-prints*, p. arXiv:1609.07061, Sept. 2016.

[14] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, pp. 127–138, Jan 2017.

[15] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, (New York, NY, USA), pp. 873–880, ACM, 2009.

[16] E. P. M. Varasteh and B. Clerckx, "A learning approach to wireless information and power transfer signal and system design," *IEEE*, Oct. 2018.

[17] D. Tse and P. Vishwanath, "Fundamentals of wireless communication," in *Fundamentals of Wireless Communication*, ch. 2, pp. 10–48, Cambridge University Press, 2005.

[18] M. Ibnkahla, "Applications of neural networks to digital communications a survey," *Signal Processing*, vol. 80, no. 7, pp. 1185 – 1215, 2000.

[19] M. Bkassiny, Y. Li, and S. K. Jayaweera, "A survey on machine-learning techniques in cognitive radios," *IEEE Communications Surveys Tutorials*, vol. 15, pp. 1136–1159, Third 2013.

[20] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," *CoRR*, vol. abs/1607.04793, 2016.

[21] E. Nachmani, E. Marciano, D. Burshtein, and Y. Be'ery, "RNN decoding of linear block codes," *CoRR*, vol. abs/1702.07560, 2017.

[22] N. Samuel, T. Diskin, and A. Wiesel, "Deep mimo detection," in *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 1–5, July 2017.

[23] J. R. Hershey, J. L. Roux, and F. Weninger, "Deep unfolding: Model-based inspiration of novel deep architectures," *CoRR*, vol. abs/1409.2574, 2014.

[24] Y. Jeon, S. Hong, and N. Lee, "Blind detection for MIMO systems with low-resolution adcs using supervised learning," *CoRR*, vol. abs/1610.07693, 2016.

[25] N. Farsad and A. J. Goldsmith, "Detection algorithms for communication systems using deep learning," *CoRR*, vol. abs/1705.08044, 2017.

[26] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for wireless resource management," *CoRR*, vol. abs/1705.09412, 2017.

[27] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," *CoRR*, vol. abs/1701.07738, 2017.

[28] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Unsupervised representation learning of structured radio communication signals," *CoRR*, vol. abs/1604.07078, 2016.

[29] T. J. O'Shea and J. Corgan, "Convolutional radio modulation recognition networks," *CoRR*, vol. abs/1602.04105, 2016.

[30] D. Tse and P. Vishwanath, "Fundamentals of wireless communication," in *Fundamentals of Wireless Communication*, ch. 5, pp. 166–227, Cambridge University Press, 2005.

[31] Y. Polyanskiy, H. V. Poor, and S. Verdu, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, pp. 2307–2359, May 2010.

[32] T. O'Shea, "Deepsig: Omnisig." `https://www.deepsig.io/omnisig/`. Accessed: 21-01-2019.

[33] T. O'Shea, "Deepsig: Omniphy." `https://www.deepsig.io/omniphy/`. Accessed: 21-01-2019.

[34] A. Lazorenko, "A medium corporation: Tensorflow performance testing, cpu vs gpu." `https://medium.com/@andriylazorenko/tensorflow-performance-test-cpu-vs-gpu-79fcd39170c`. Accessed: 01-11-2019.

[35] "Alistair Wallace project github." `https://github.com/alistairwallace97/DNN_short_block_comms`. Last Updated: 07-06-2019.

[36] S. L. Smith, P.-J. Kindermans, and Q. V. Le, "Don't decay the learning rate, increase the batch size," in *International Conference on Learning Representations*, 2018.

[37] M. Divya, "Bit error rate performance of bpsk modulation and ofdm-bpsk with rayleigh multipath channel," 2013.

[38] L. Andrews, *Special Functions of Mathematics for Engineers*. Oxford science publications, SPIE Optical Engineering Press, 1998.

[39] E. T. S. Institute, "Final draft etsi en 301 908-8," *Candidate Harmonized European Standard (Telecommunications series)*, vol. V1.1.1, 2016.

[40] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, pp. 147–160, April 1950.

[41] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. New York, NY, USA: Wiley-Interscience, 2005.

[42] "J.Kent's python-bchlib bch encoding github." `https://github.com/jkent/python-bchlib`. Accessed: 07-06-2019.

[43] J. G. Proakis, *Digital Communications*. McGraw-Hill, 1995.