School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

BiCGAN

BiDirectional Classifying Generative Adversarial Networks

By

Elliott Vercoe

Project submitted as a requirement for the degree of
Masters of Information Technology

Submitted: December 2019 Supervisor: Dr. Alan Blair

Student ID: z3411256

Abstract

BiCGAN consists of a modification to the existing BiGAN (or Adversarially Learned Inference) which involves the application of a Generator, Encoder and Discriminator network to generate synthetic data samples, by optimizing a minimax game. A novel Classifier component has been introduced, allowing the generation of data samples satisfying the properties of some given class.

In the process of implementing the new Classifier, several extensions were made to the BiGAN network structure by implementing new loss functions. Additionally, novel strategies for data generation policies were explored and reported on.

The BiCGAN was applied to the UCI Sonar Dataset. When tested with 10-fold cross validation, the BiCGAN was able to generate data samples that allowed a fresh neural network to be trained to achieve 73 % accuracy, compared to an 83 % control test that was achieved when using the original, unmodified data samples.

The BiCGAN was applied to the Credit Card Dataset, a heavily imbalanced dataset with a class ratio of approximately 500:1. The BiCGAN was able to generate data samples that allowed a fresh neural network to be trained to achieve a performance of 0.744 precision and 0.592 recall, compared to 0.869 precision and 0.612 recall that was achieved using the original, unmodified data samples as a control test.

Contents

Chapter 1 Project Goals and Motivations

1.1 Overview of Generative Adversarial Networks (GANs)

A generative adversarial network (GAN) is a class of machine learning systems invented by Ian Goodfellow and his colleagues in 2014. Two neural networks contest with each other in a game (in the sense of game theory, often but not always in the form of a zero-sum game). Given a training set, this technique learns to generate new data with the same statistics as the training set. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics. Though originally proposed as a form of generative model for unsupervised learning, GANs have also proven useful for semi-supervised learning, fully supervised learning, and reinforcement learning.

Some areas that GANs have found practical application include:

- Image generation, for creating new unseen portraits, landscapes, and artistic works
- Science, for image processing and handling computationally intensive processes
- Videogames, for texture downsampling

The wide spectrum of GAN applications means that improvements in the basic GAN structure could be applied to a variety of fields, and that improving the GAN procedure has implications well beyond their theoretical implementations.

1.2 Generating Synthetic Data

The goal of the Generator component of a GAN is to produce a new data sample which is not present in the training data, but is superficially indistinguishable from the original data set. This generated sample can be considered as a piece of synthetic data.

Synthetic data are generated to meet specific needs or certain conditions that may not be found in the original, real data. This can be useful when designing any type of system because the synthetic data are used as a simulation or as a theoretical value, situation, etc.

Synthetic data is increasingly being used for machine learning applications: a model is trained on a synthetically generated dataset with the intention of transfer learning to real data. In general, synthetic data has several natural advantages:

- Once the synthetic environment is ready, it is fast and cheap to produce as much data as needed;
- The synthetic environment can be modified to improve the model and training;
- Synthetic data can be used as a substitute for certain real data segments that contain, e.g., sensitive information.

1.3 Extending GAN, BiGAN and Existing GAN Methodologies

Existing methodologies such as BiGAN (Donahue J, 2016) and Adversarially Learned Inference (Dumoulin, Belghazi et al, 2016) have introduced some novel methods for generating synthetic data based on some selected conditions. These methodologies are explained in further detail in Chapter 2.

This project aims to build upon the basis provided by these methodologies, as well as to mix in new ideas and concepts. This project hypothesises that by implementing a Classifier into the existing BiGAN system, the functionality of the BiGAN can be greatly extended, both as a system for classification, and as a system for generating synthetic data. The project will analyse and report on the considerations that were required to produce this implementation.

1.4 Using Classification in BiGAN Inference

The major motivation behind this project is to introduce a data sample classification to the BiGAN procedure (outlined in detail in Section 2.3 and 2.4). Existing methodologies (Belghazi et al. 2016) ascribe semantic meaning to latent vectors by fixing parameters of the encoded latent space, using an encoder from outside their trained BiGAN. Belghazi et al. 2016 states ‘we conjecture that the latent representation learned by ALI is better untangled with respect to the classification task and that it generalizes better.’

This project attempts to validate their conjecture, as by implementing a separate classifier to backpropagate class information, the latent space can remain unfixed. This allows the semantic latent representation to be determined automatically, and ideally an optimal representation is found using unknown latent variables.

Further to implementing the classifier, we have provided restrictions that the latent vectors must be approximately evenly distributed (by using the Plum Pudding Loss Function), the latent vector must be conserved through a $Z \rightarrow G(Z) \rightarrow E(G(Z))$ transformation, and the binary classification must be derived through a linear operation on this vector. These techniques are elaborated upon within Chapter 3.

The intention is to explore and report on techniques and methodologies which have been examined in order to implement this classifier in the BiGAN system. This report will explain ideas that worked, ideas that did not work, and ideas that can be further elaborated upon in the future.

1.5 Generating Synthetic Data Samples of a Given Class

Implementing a binary classifier allows the latent space to be considered as 2 distinct regions: a region of latent vectors which map to a positive classification, and a region of latent vectors which map to a negative classification. An implication of implementing this mapping is that latent vectors can be sampled from these regions, and transformed into fake data samples, with a class corresponding to the classification of their seed latent vector.

This report will examine different policies and procedures for sampling synthetic data, and provide analysis on effective and novel methods.

1.6 Evaluating the Effectiveness of Synthetic Data

The existing literature surrounding synthetic data typically focuses on the generation of images. These images provide easy and entertaining qualitative feedback to the researcher, as one can simply look at the created sample and determine if their generation system is effective: if the picture looks real, then the system is probably working. Quantitative evaluation is typically performed by evaluating the classification performance of the system on a test set.

It is not possible to qualitatively evaluate the Sonar Dataset (60 variables) and Credit Card Dataset (30 variables) in the same manner, as looking at them does not give away much information. In addition, evaluating the performance of the classifier on a test set did not appear to give any noticeable improvement over the performance of an ordinary classifier.

Instead, a novel methodology has been introduced utilising the generated fake data samples. First, a fresh neural network classifier is trained using a training set of the original data. Then, a second fresh neural network classifier of the same structure is trained using a set of generated fake data. The performance of these 2 new classifiers is then evaluated on a hold out training set. Further detail of this process is outlined in Chapter 5.

If the generated data is indistinguishable from the original data, then both these fresh classifiers should achieve the same performance. Several experiments have been performed and analysed to evaluate the effectiveness of this technique.

1.7 Using Synthetic Data Generation as an Oversampling Technique

Some classifiers (such as SVMs and K-nearest neighbours) can perform poorly on imbalanced datasets (without additional parameter tuning). Existing oversampling techniques are designed to provide balance to imbalanced datasets, in order to improve the performance of these classifiers. Naïve oversampling methods simply duplicate members of the minority class to provide balance. Additional novel methodologies have been established in recent years to provide more interesting oversampling strategies.

SMOTE is the main oversampling technique which involves generating new samples. In simple terms, SMOTE interpolates between nearby minority class samples, and generates a new sample at a random point between them. This produces a set of new samples, with additional variation when compared to naïve oversampling. Section 2.6.2 provides a more detailed outline of SMOTE.

BiCGAN could provide a more diverse set of oversampled data when compared to SMOTE, and may be able to be used in its stead.

Chapter 2 Background

2.1 Introduction to Latent and Observable Variables

The concept of a data distribution (X), and a latent variable distribution (Z) will be familiar to readers with backgrounds in multivariable statistical analysis, however these concepts may prove unwieldy to readers without this background. A brief explanation of these concepts is provided for these unfamiliar readers. This section can be skipped by those with more experience.

The observed (or measured) data, conventionally referred to as X, are variables that are actually measured by the researcher. They are data that actually exists in your data files—data that has been measured and recorded. They can be discrete variables or continuous variables. In the first experiment performed in this report, these are the 60 measurements of a sonar transmitter by some sensor, each represented as a variable with a value between 0 and 1. In the second experiment, the X distribution is the set of 30 Principal Components of a Credit Card transaction. This measurement is more abstract and is outlined in more detail later, however we still treat this data sample as 30 unique variables.

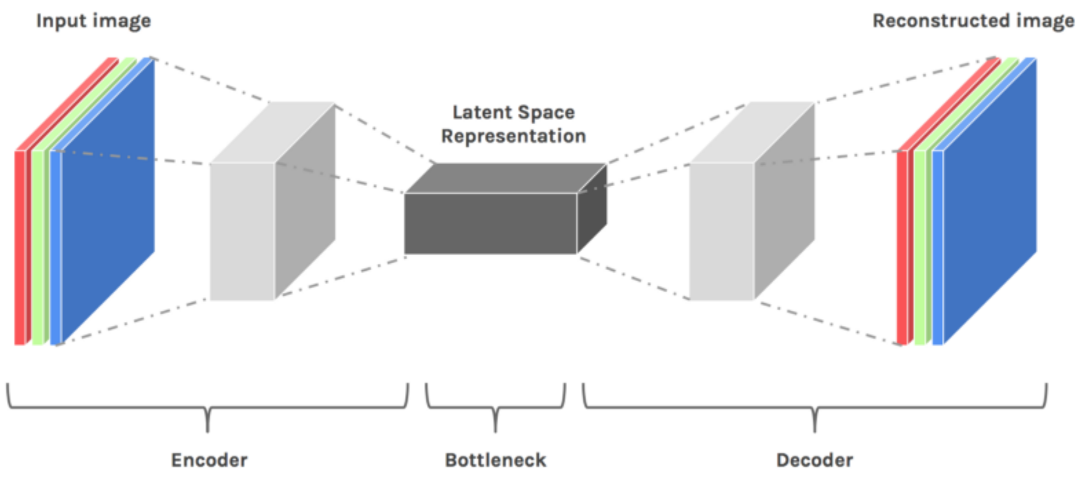

Figure 1 - Encoding to a Latent Space, then Decoding to produce a Reconstruction (Despois, 2019)

Latent variables are variables that are not directly observed, but are rather inferred (through a mathematical model) from other variables that are observed (X, as outlined above). Sometimes latent variables correspond to aspects of physical reality, which could in principle be measured, but may not be for practical reasons. In this situation, the term hidden variables is commonly used (reflecting the fact that the variables are "really there", but hidden). Other times, latent variables correspond to abstract concepts, like categories, behavioural or mental states, or data structures.

One advantage of using latent variables is that they can serve to reduce the dimensionality of data. Many observable variables can be aggregated in a model to represent an underlying concept, making it easier to understand the data. In this sense, they serve a function similar to that of scientific theories. At the same time, latent variables link observable data in the real world to symbolic data in the modelled world.

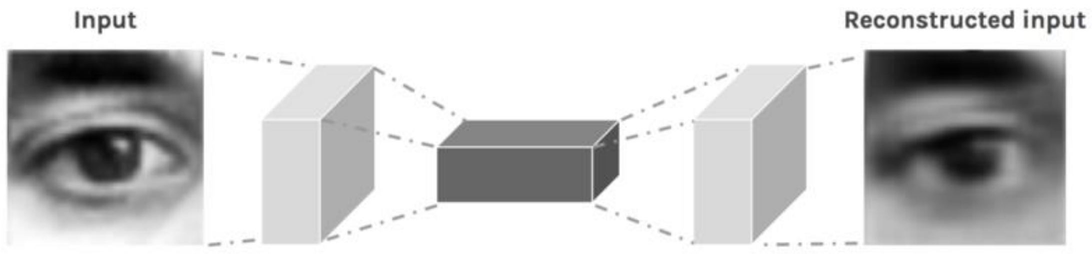

Figure 2 - Real example of Reconstructed Input (Despois, 2019)

In this research project, we attempt to build an Encoder which is able to take an observed sample X and convert it to a latent distribution Z. At the same time, we try to build a Generator (labelled as Decoder in the diagram) which builds a data sample from a latent distribution Z.

2.2 Generative Adversarial Networks (GAN)

The seminal work on Generative Adversarial Networks was completed by Goodfellow et al (Goodfellow et al, 2014). The team proposed a new framework for estimating generative models via an adversarial process. Two models are trained: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data, rather than G. The training procedure for G is to maximise the probability of D making a mistake. This framework corresponds to a minimax two-player game. Initial exploration of this concept defined G and D as multilayer perceptrons, allowing the entire system to be trained with backpropagation.

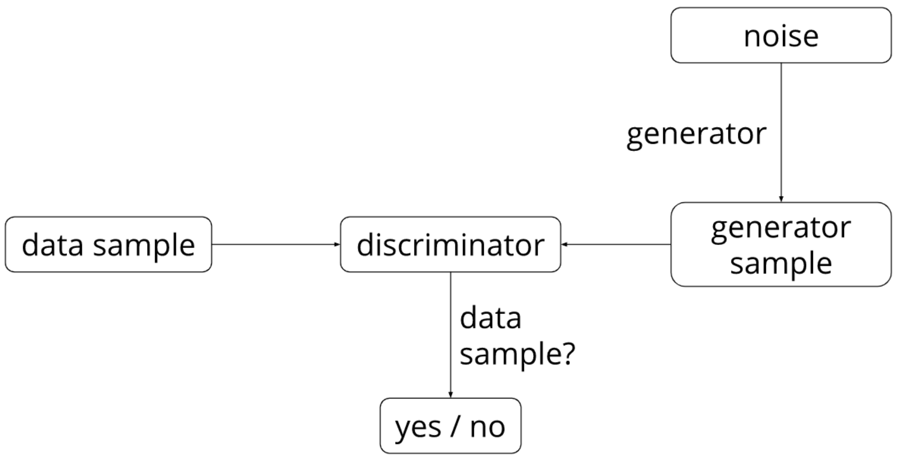

Figure 3 - General concept of GANs as proposed by Goodfellow et all (Goodfellow et al, 2014)

The Generator $G_\theta: z \mapsto x$, with parameters θ generates an image x from latent variables z, sampled from a standard Normal Distribution.

The Discriminator $D_\Psi: x \mapsto D_\Psi(x) \in (0, 1)$ with parameters Ψ, takes an image x and estimates the probability of the image being real.

The Generator and Discriminator play a 2-player zero-sum game to compute:

$$\min_\theta \max_\Psi (\boldsymbol{E}_{x\sim p_{data}}[log D_\Psi(x)] + E_{z\sim p_{model}}\left[\log\left(1 - D_\Psi\left(G_\theta(z)\right)\right)\right]$$

The discriminator tries to maximize the bracketed expression, while the Generator tries to minimize it.

2.3 Adversarially Learned Inference

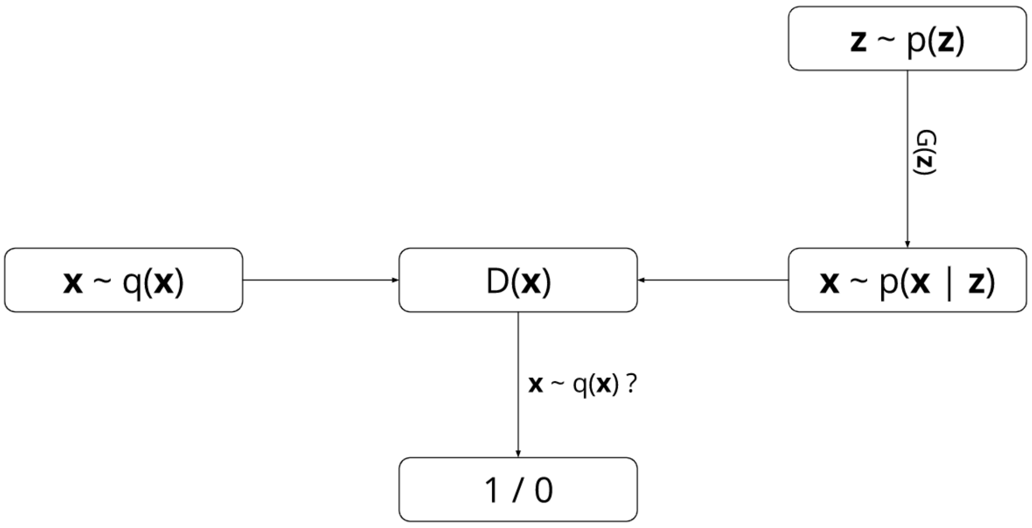

Figure 4 - A view of GANs with respect to distribution sampling (Dumoulin, Belghazi et al, 2016)

Later work by Dumoulin, Belghazi et al introduced the Adversarially Learned Inference model (ALI, which we refer to as BiDirectional GAN (BiGAN)) which jointly learns a generation network and an inference network using an adversarial process. ALI defines 2 joint distributions:

The encoder joint $q(x,z) = q(x)q(z|x)$

The decoder joint $p(x,z) = p(z)p(x|z)$

ALI also modifies the discriminator's goal; rather than examining x samples marginally, it now receives joint pairs (x,z) and must predict whether they come from the encoder joint or the decoder joint.

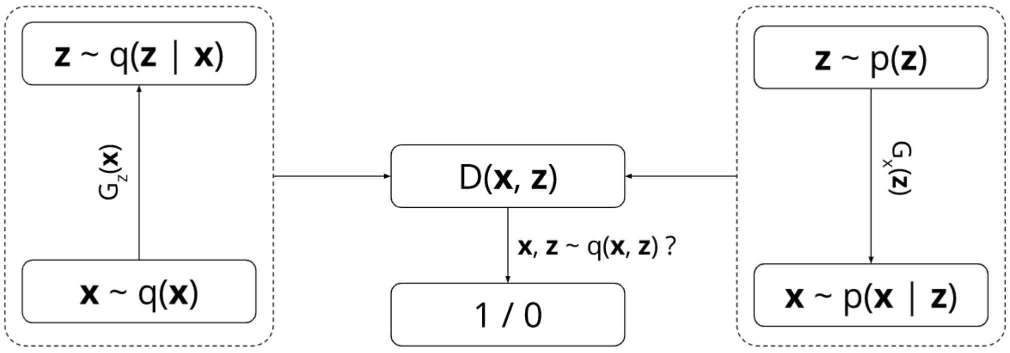

Figure 5 - ALI / BiGAN replaces the Generator and Discriminator with Joint Pairs, as extended by (Dumoulin, Belghazi et al, 2016)

This adversarial game function is therefore extended to become the following:

$$\min_{\theta} \max_{D} V(D,G) = \iint q(x)q(z|x)\log\big(D(x,z)\big)\,dxdz$$
$$+ \iint p(z)p(x|z)\log\big(1 - D(x,z)\big)\,dxdz$$

2.4 BiDirectional GAN (BiGAN)

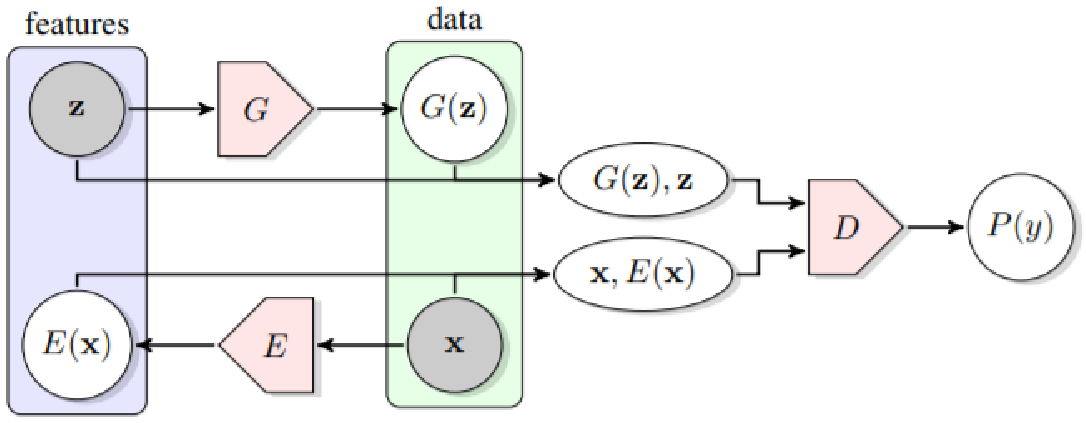

Figure 6 - BiGAN structure (Zhang J, 2018)

BiGAN (Donahue J, 2016) represents the same concept as Adversarially Learned Inference. The original papers for both ideas were published at a similar time, such that neither concept legitimately precedes the other. Reference to the BiGAN paper is provided to credit the authors of both papers, and to explain the origin of the name.

2.5 Conditional GAN (CGAN)

Work by Mirza, M., & Osindero, S. (2014) proposed Conditional GAN, in which the Discriminator and Generator are both conditioned on some extra information y. This is similar in concept to BiGAN, except that BiGAN is using an encoder to generate the extra information in the latent space.

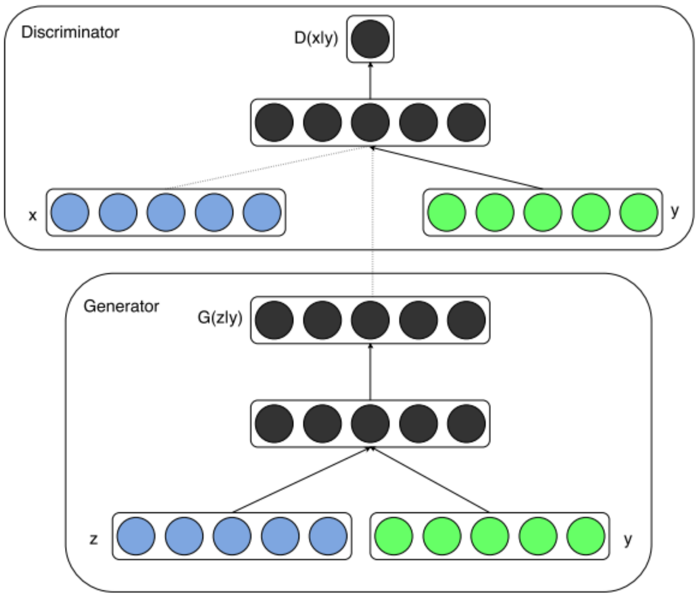

Figure 7 - Conditional GAN as proposed by Mirza, M., & Osindero, S. (2014)

2.6 Imbalanced Class Methods

2.6.1 Motivations

Imbalanced class (or minority class) methodologies attempt to achieve the same goal as BiGANs: generate synthetic data samples. However, imbalanced class methodologies are generally much simpler, and typically used with lower dimensional data. A brief outline of these methodologies is provided, to provide context to the field of synthetic data generation.

2.6.2 SMOTE (Synthetic Minority Over-sampling Technique)

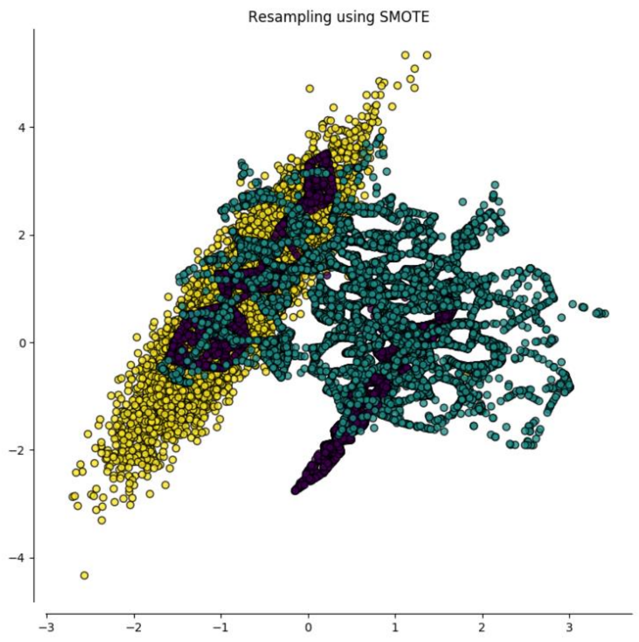

Figure 8 - Example implementation of SMOTE ("2. Over-sampling — imbalanced-learn 0.5.0 documentation", 2019)

SMOTE is the most common technique for generating synthetic data for oversampling problems. To illustrate how this technique works consider some training data which has s samples, and f features in the feature space of the data. Note that these features, for simplicity, are continuous. As an example, consider a dataset of birds for classification. The feature space for the minority class for which we want to oversample could be beak length, wingspan, and weight (all continuous). To then oversample, take a sample from the dataset, and consider its k nearest neighbors (in feature space). To create a synthetic data point, take the vector between one of those k neighbors, and the current data point. Multiply this vector by a random number x which lies between 0, and 1. Add this to the current data point to create the new, synthetic data point.

Many modifications and extensions have been made to the SMOTE method ever since its proposal.

2.6.3 SMOTE Extensions (ADASYN)

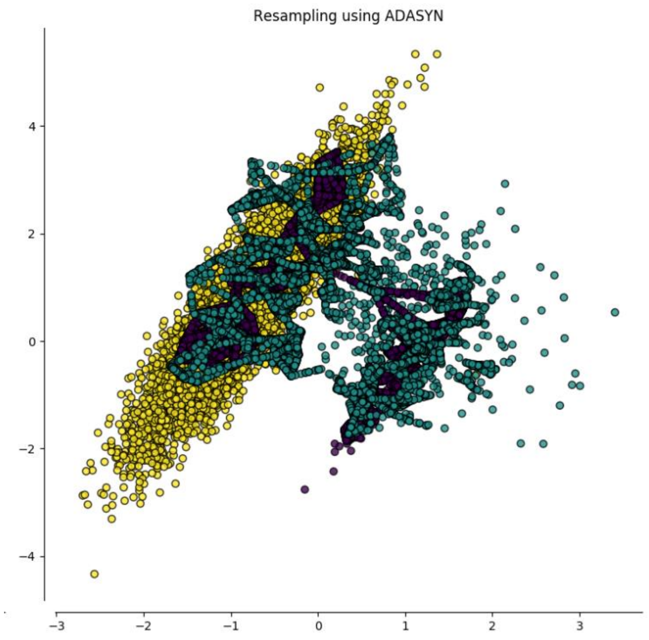

Figure 9 - Example Implementation of ADASYN ("2. Over-sampling — imbalanced-learn 0.5.0 documentation", 2019)

The adaptive synthetic sampling approach, or ADASYN algorithm, builds on the methodology of SMOTE, by shifting the importance of the classification boundary to those minority classes which are difficult. ADASYN uses a weighted distribution for different minority class examples according to their level of difficulty in learning, where more synthetic data is generated for minority class examples that are harder to learn.

2.6.4 Undersampling Techniques (Random Sampling, Clustering, Tomek Links)

Randomly remove samples from the majority class, with or without replacement. This is one of the earliest techniques used to alleviate imbalance in the dataset, however, it may increase the variance of the classifier and may potentially discard useful or important samples.

Cluster centroids is a method that replaces cluster of samples by the cluster centroid of a K-means algorithm, where the number of clusters is set by the level of undersampling.

Tomek links remove unwanted overlap between classes where majority class links are removed until all minimally distanced nearest neighbour pairs are of the same class.

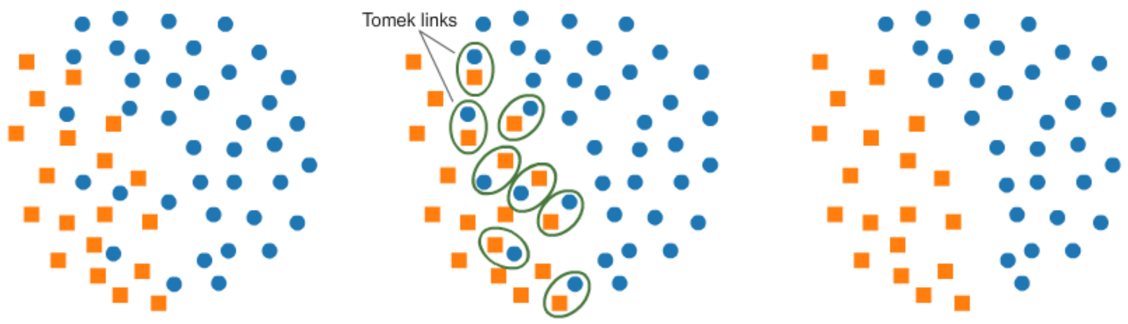

Figure 10 Tomek Links remove unwanted overlap to clearly separate classes ("Resampling strategies for imbalanced datasets | Kaggle", 2019)

Chapter 3 Training Strategies

3.1 Training Introduction

This chapter will first outline the novel principles and features which have been implemented in the BiCGAN system. The chapter will then conclude by outlining the final implemented design, with reference to the preceding features. The reader should carefully examine the system architecture, and refer back to the related feature subheading for context.

3.2 Plum Pudding Loss

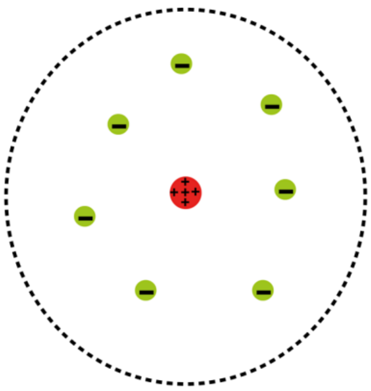

Figure 11 - Plum pudding model of an atom - each electron repels one another, however is still constrained within the substrate

The Plum Pudding Model is a model of atomic structure proposed by J.J. Thomson in the late 19th century. Thomson had discovered that atoms are composite objects, made of pieces with positive and negative charge, and that the negatively charged electrons within the atom were very small compared to the entire atom. He therefore proposed that atoms have structure similar to a plum pudding, with tiny, negatively charged electrons embedded in a positively charged substrate. The electrons would push away from each other, spreading themselves out throughout the substrate. This theory was later shown to be incorrect, however this theory forms the inspiration for the plum pudding latent vector encoding.

The theory of the Plum Pudding has been applied to the latent vector encoding. We introduce a loss function that attempts to provide a negative charge to each latent vector. The goal of this loss function is to spread the vectors evenly throughout the latent space. This will result in a latent space mapping which attempts to separate the differences between each data sample. It means that each region of the Z space will be mapped to a valid X, rather than potential regions producing strange and unseen X mappings.

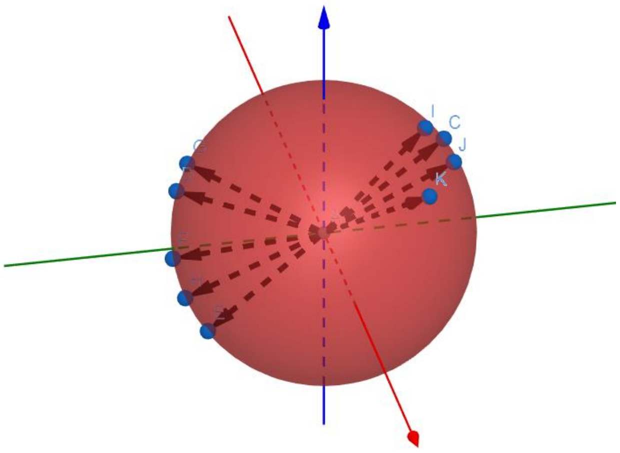

Figure 12 - Without the Plum Pudding Loss: All encodings get mapped to random regions of Z space, but are bunched together, leaving large areas of unseen Z space.

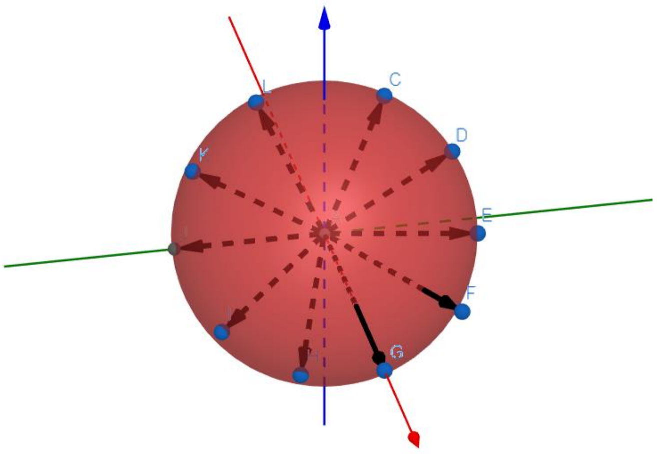

Figure 13 - With Plum Pudding Loss: The encodings have been evenly mapped throughout the Z space

The following equation specifies the exact implementation of the plum pudding loss. This was implemented efficiently with the use of Pytorch and Numpy's fast matrix operation, allowing the loss to be easily calculated between each sample in the batch.

$$Loss = \sum_i \sum_j \frac{1}{2\sqrt{N}}(z_i^2 \; + \; z_j^2) \; - \sqrt{N} log(1 + \left|z_i - z_j\right|^2)$$

Note that the first term in the plum pudding loss attempts to reduce the size of the vectors, and the second term attempts to spread them apart.

The Plum Pudding Loss enables us to generate X vectors from any region of the latent space. This means if we randomly generate vectors of the same magnitude that the generator is trained on, each of these vectors should be close to a real data sample. As a result, any point in the Z space should produce a valid X.

Belghazi et al reported problems with their BiGAN generator dropping modes. The Plum Pudding Loss also means that randomly sampling from the latent space should cover all the output modes. Ideally by distributing the latent space evenly, the modes should become spread throughout the latent space, and we can therefore cover all modes by sampling randomly enough times.

3.3 Contrastive Discrimination Loss

The original GAN implementation utilised a Discriminator which accepts a data sample X, and attempted to determine whether this data sample belonged to the real set, or the generated set. This allows for the Discriminator to learn what real data samples look like, however it gains no knowledge of the relationship between the latent variables and the data samples. As BiCGAN has been extended to learn a meaningful and manipulable latent space encoding, it is a natural extension to feed this extra information to the Discriminator, allowing it to discriminate in both data space, and latent space.

The following pairs of vectors are concatenated and fed to the Discriminator:

1. Real X + Real Z. A Real X vector from the original dataset is fed to the Encoder, and the Z distribution that is generated is considered to be a Real Z vector. This is the baseline ‘truth’.
2. Fake X + Fake Z. A randomly generated Z vector is produced. This Fake Z is fed to the Generator, which produces a corresponding Fake X.
3. Fake X + Real Z. The Real Z vector outlined in part 1 is concatenated to the Fake X vector outlined in part 2. This pushes the discriminator to learn the relationship between the X and Z distributions.

Note that the fourth contrastive discrimination, Real X + Fake Z, is not relevant, as neither the Real X (ie a real data sample) or Fake Z (ie a random vector) have been generated by the BiCGAN, so it does not make sense to train on these pairings.

The feedback is provided as a Softmax vector of dimension 3, with 1 dimension corresponding to each pair of vectors. The loss is calculated according to its ability to correctly classify the pair of vectors.

3.4 Linear Classifier Loss

A Linear Classifier is produced which classifies based on the encoded Z vector. As a result, in a binary classification problem, the Z space ends up split into 2 regions, in which vectors on one side of the boundary map to a single class.

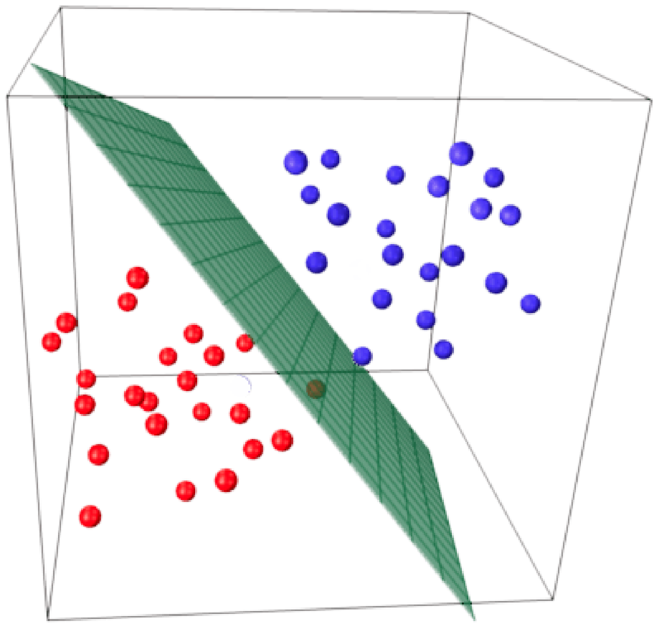

Figure 14 - Mapping Z vectors to a linearly separable 3D space
Note that the Z space is typically >6 dimensions rather than the easily visualised 3

The Linear Classifier is trained by taking a real data sample X, with its class pair y, and encoding this sample to a Z vector. This Z vector is then passed to the Linear Classifier, which is trained to correctly classify this vector using a binary cross-entropy loss function. In addition, the error is further backpropagated through to the Encoder, which is also trained using this loss function. This results in both the Classifier and Encoder simultaneously learning to map and classify the distribution.

The intention of including a Linear Classifier in the system, is to permit the generation of a specific class. To generate a specific class, the Z vector can be sampled from either side of the classification plane, to produce an X vector of a desired class.

3.5 Flipped Autoencoder Loss

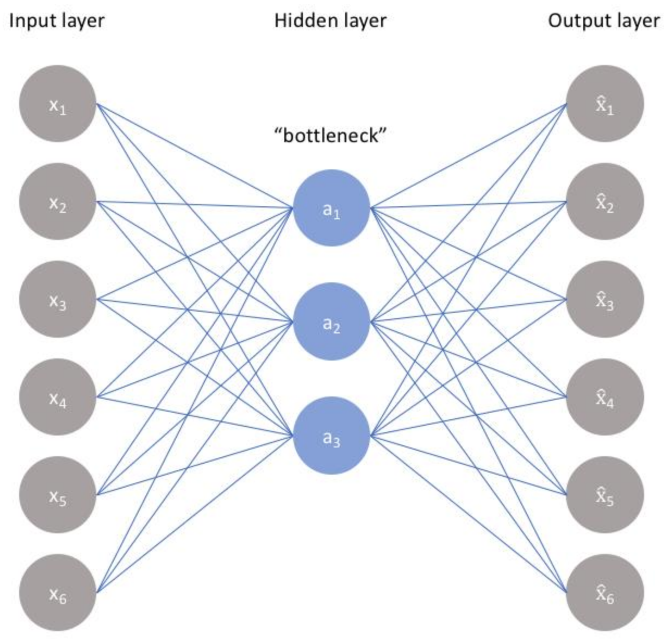

Figure 15 - Autoencoder concept

A conventional autoencoder (pictured above) transforms an input to the Z space, then reverts the transformation into an output. An autoencoder is a type of artificial neural network used to learn efficient data encodings in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal noise. Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input, hence its name. Several variants exist to the basic model, with the aim of forcing the learned representations of the input to assume useful properties.

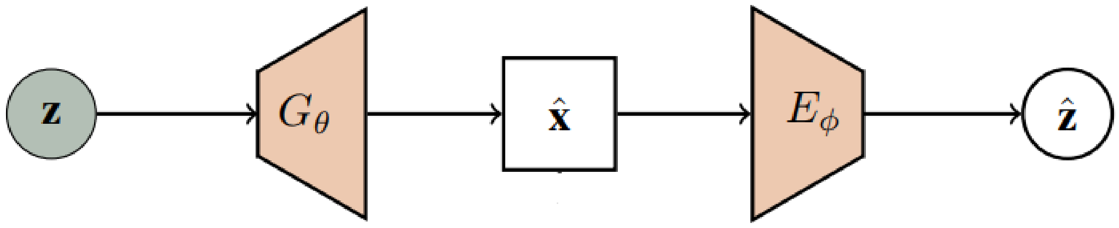

Figure 16 - Flipped Autoencoder (Zhang J, 2018)

For BiCGAN, a Flipped Autoencoder (Zhang J, 2018) is implemented to train the system to preserve the meaning of the Z encoding. The implementation takes a random noise vector Z, converts it into the observed X space as a generated data sample, then encodes this data sample back into the Z space. The mean-squared-error loss is calculated on the difference between the original noise vector, and the decoded-encoded Z vector. This error is then backpropagated through the Generator and the Encoder.

The principle behind an Autoencoder is to encode and decode a data sample, to train the system to replicate realistic samples. Instead, we have decoded then encoded a latent vector, and are training the generator to produce samples which capture the information conveyed in the latent vector.

3.6 Total Autoencoded L2 Loss Heuristic

A useful heuristic in measuring the progression of the system has been to calculate the Autoencoded L2 loss of the entire dataset. This involves taking the entire dataset, feeding it to the encoder, then feeding the encodings back to the generator, then taking the L2 loss between the original dataset and the generated encoded data. This is equivalent to using the BiCGAN as an autoencoder, as outlined in the previous section. Note that we do not train the system based on this heuristic, but rather just provide a useful plot to determine when the system has converged.

The figure below outlines a typical Autoencoded L2 loss curve on the Sonar dataset. Generally, the loss will spike, and then ultimately converge to a reasonably stable value. This provides a useful indicator of whether any experimentation with the system design has caused instability, or whether the system can still safely converge.

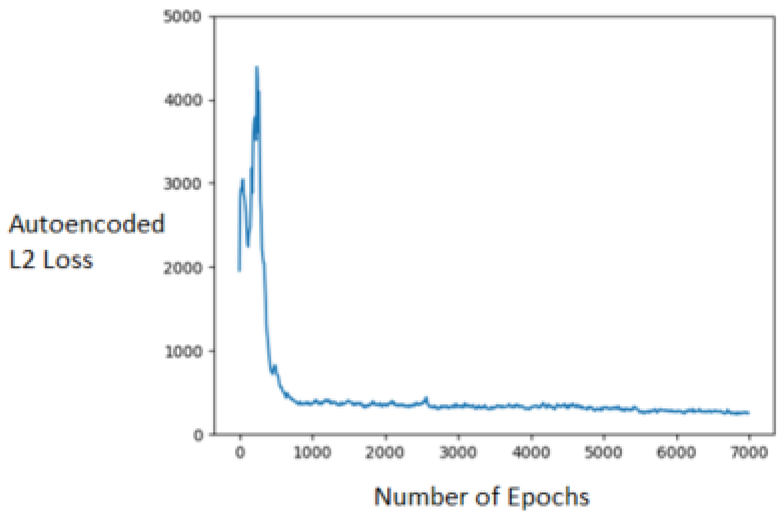

Figure 17 - Sample plot of the Autoencoded L2 Loss, this result can be seen to converge

3.7 System Architecture

The following diagram outlines the general concept behind the BiCGAN. Each system in this diagram is outlined in further detail within this chapter. Some training interactions have been excluded from this diagram for simplicity and will be later explained in complete detail.

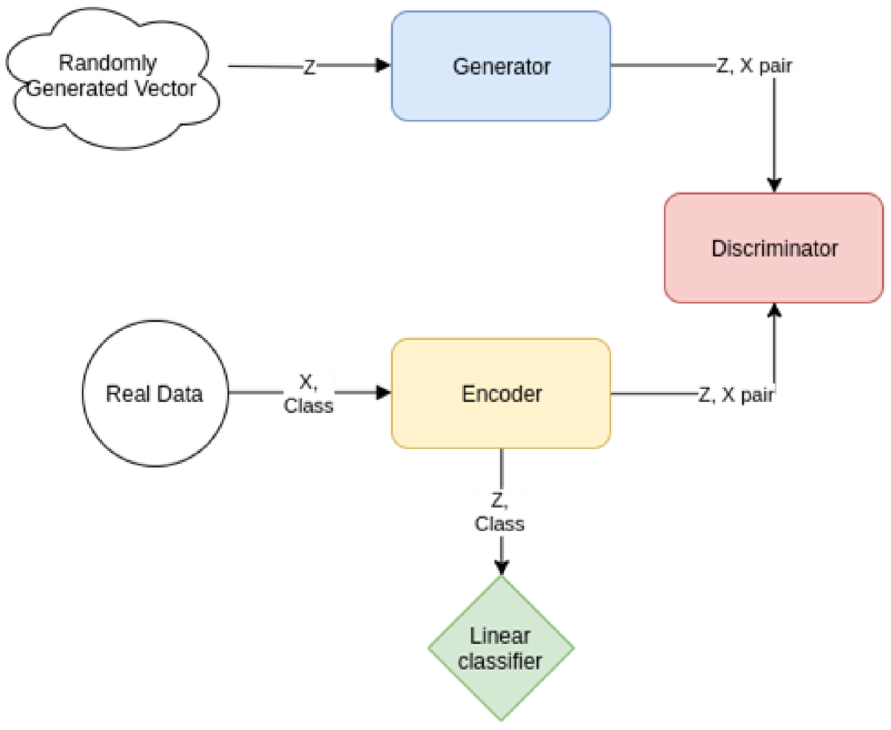

Figure 18 - System Architecture

3.7.1 Generator

The Generator maps the Z space to the X space, that is to say that it accepts latent variables, and generates an imitation data sample. The Generator has 2 objectives:

1. Deceive the Discriminator into mistaking the imitation data for real data.
2. Accurately preserve the meaning of the encoded feature space; make sure that when a Z vector is transformed into an X vector, it can then be encoded back to the original Z vector (Flipped Autoencoding).

3.7.1.1. Typical Network Parameters

- 2 hidden fully connected layers with 64 nodes each.
- Leaky ReLU activation functions on each layer.
- Batch norm applied after the second hidden layer.
- Adam optimizer (with success found with learning rate of 0.0001).

3.7.2 Encoder

The Encoder maps the X space to the Z space, that is to say it accepts a data sample, and attempts to map this sample to a set of feature parameters. The Encoder has 3 objectives:

- Satisfy the plum pudding model, distributing encoded samples approximately evenly across the latent space (Section 3.2)
- Provide an encoding which linearly separates the sample class
- Produce meaningful encodings of X vectors; make sure that when a Z vector is transformed into an X vector, it can then be encoded back to the original Z vector (Flipped Autoencoding).

3.7.2.1. Typical Network Parameters

- 1 hidden fully connected layer with 64 nodes.
- Leaky ReLU activation functions on each layer.
- Batch norm applied after the hidden layer.
- Adam optimizer (with success found with learning rate of 0.0001)

3.7.3 Discriminator

The Discriminator accepts a pair of X and Z vectors, and attempts to determine whether these vectors are real or fake. The following pairs of vectors are concatenated and fed to the Discriminator:

1. Real X + Real Z. A Real X vector from the original dataset is fed to the Encoder, and the Z distribution that is generated is considered to be a Real Z vector.
2. Fake X + Fake Z. A randomly generated Z vector is produced. This Fake Z is fed to the Generator, which produces a corresponding Fake X.
3. Fake X + Real Z. The Real Z vector outlined in part 1 is concatenated to the Fake X vector outlined in part 2. This pushes the discriminator to learn the relationship between the X and Z distributions.

These pairings are produced as a one-hot Softmax encoding, ie a vector of length 3 with values RxRz, FxFz, FxRz.

3.7.3.1. Typical Network Parameters

- 1 hidden fully connected layer with 64 nodes.
- Leaky ReLU activation functions on each layer.
- Batch norm applied after the hidden layer.
- SGD optimizer (with success found with learning rate of 0.01)

3.7.4 Linear Classifier

The Classifier simply attempts to find a linear separation in the encoded Z vector, which will permit optimal classification. This feedback is provided to the encoder, to encourage an encoding that captures the classification of a data sample.

3.7.4.1. Typical Network Parameters

- The Linear Classifier is a single Linear Layer, with input matching the dimension of Z, and a single dimensional output.
- Multiple output activation functions were experimented with, typically a Sigmoid was used.
- Adam optimizer (with success found with learning rate of 0.0001)

3.8 Feedback Pathways

3.8.1 Discriminator Accuracy + Plum Pudding Feedback [Generator + Encoder Training]

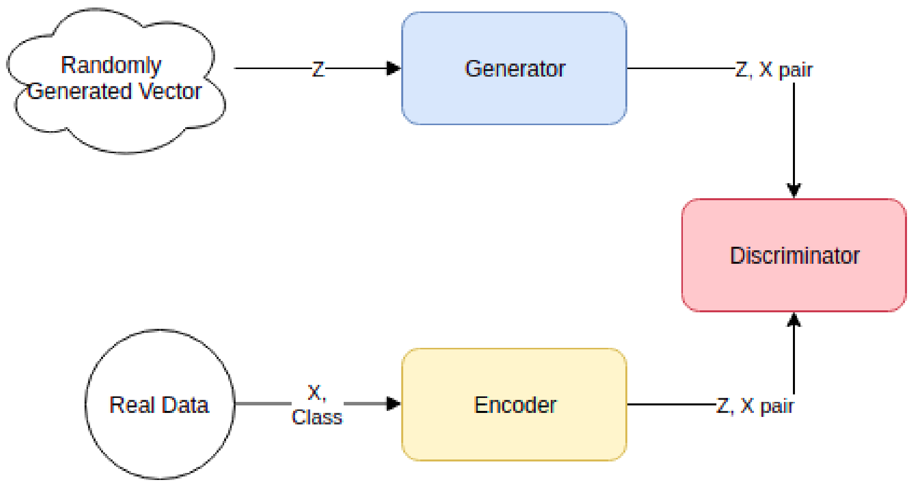

Figure 19 – Training the Generator and Encoder on their ability to fool the Discriminator

1) Take a real data sample X and encode it with the encoder producing a Z, X pair such that Z = E(X). Concatenate these 2 vectors and send to the Discriminator.
2) Generate a random Z vector and pass it to the generator to yield a Z, X pair such that X = G(Z). Concatenate these 2 vectors and send to the Discriminator.
3) The Discriminator vector contains the Discriminator's best guess as to whether the X/Z pairing has come from a real data sample, or has been generated based on random noise. Because the goal of the Generator / Encoder pairing is to

deceive the Discriminator, we backpropagate the negative of the Discriminator vector Real X / Real Z pairing for the X, E(X) pair, and the negative of the Fake X / Fake Z pairing for Z, G(Z) pair.

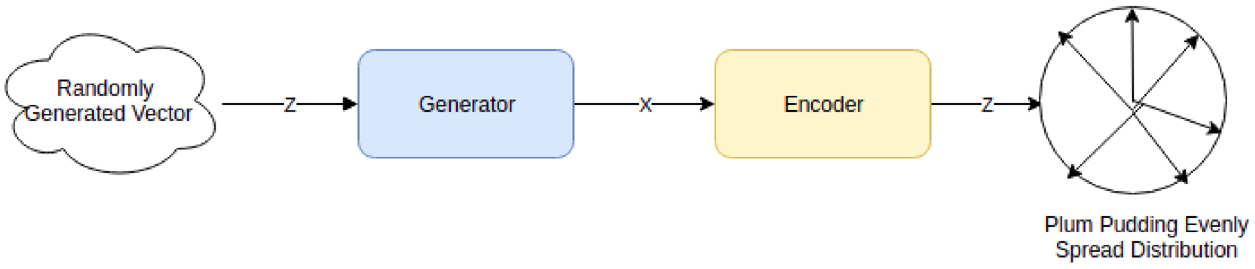
Randomly Generated Vector
Z
Generator
X
Encoder
Z
Plum Pudding Evenly Spread Distribution

Figure 20 - Plum Pudding Loss applied to spread random vectors evenly amongst the Z space

4) We apply the Plum Pudding Loss to the Generator → Encoder decomposition. We generate a random noise vector Z then feed it to the Generator to obtain an X = G(Z), then feed that X to the Encoder to generate a Z = E(G(Z)). We then apply the plum pudding loss model to try to spread these Z vectors apart to obtain an even distribution. As the input values are random, the latent variables should, on average over enough iterations, be evenly spread apart in the latent space. This step teaches the encoder to generate a healthy Z distribution.

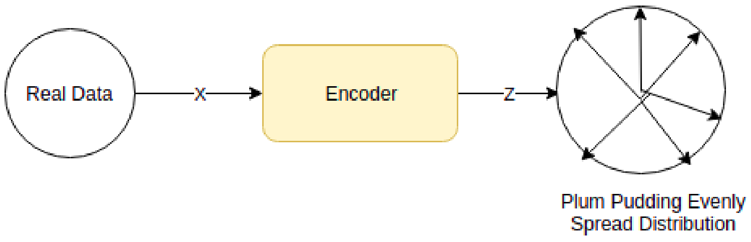

Figure 21 - Plum Pudding Loss applied to spread Real Data evenly amongst the Z space

5) We repeat the above step, except use the Z = E(X) calculated in step 1. This ensures that there is sufficient variation in the encoding of the real vectors, in an attempt to capture the minor difference in latent variables between observed data samples.

3.8.2 Discrimination Accuracy Feedback [Discriminator Training]

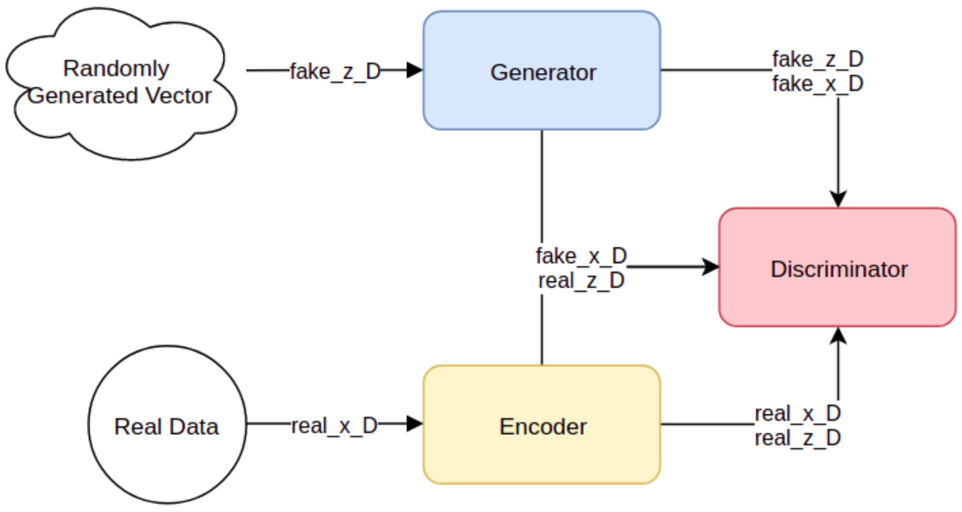

Figure 22 - Training the Discriminator on its ability to distinguish between contrastive Generated and Encoded pairings

1) Generate the following vectors:

   real_x_D: An observed data sample

   real_z_D: Take the above real_x_D and pass it to the encoder to yield a real encoded vector.

   fake_z_D: Generate a random Z vector.

   fake_x_D: Take the above fake_z_D and pass it to the generator to yield a fake data sample.

2) Concatenate these vectors into the following combinations:

   d_RxRz = real_x_D + real_z_D

   d_FxFz = fake_x_D + fake_z_D

   d_FxRz = fake_x_D + real_z_D

3) Pass each of these contrastive combinations to the Discriminator, based on the correctness of the Discriminator’s prediction. Only the Discriminator is trained.

3.8.3 Accurate Classification Feedback [Linear Classifier + Encoder Training]

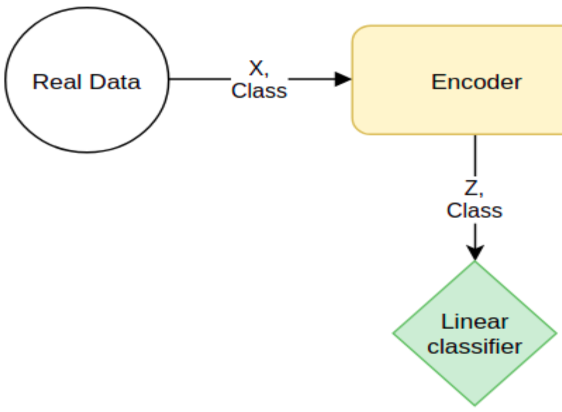

Figure 23 - Train the Classifier and Encoder based on their ability to correctly classify data

1) As in step 1) above, take an observed data sample X and pass it to the encoder to yield a real encoded vector E(X).
2) Pass this encoded vector E(X) to the classifier to produce a classification C(E(X)).
3) Compare the result of the classified encoded vector, to the actual classification of the original data sample. Backpropagate this loss through the Classifier and the Encoder utilising a binary cross-entropy loss.

3.8.4 Flipped Autoencoding Feedback [Generator + Encoder Training]

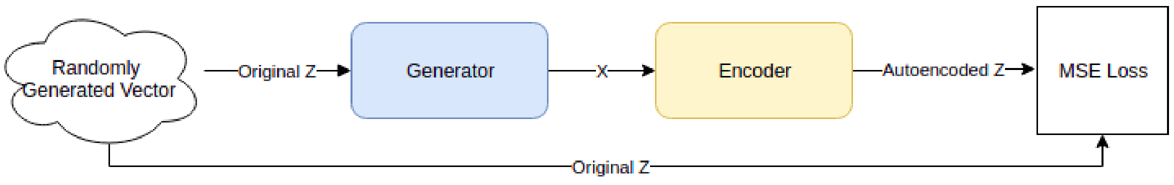

Figure 24 - Train the Generator and Encoder on their ability to conserve the Z space meaning throughout their transformations

The flipped Autoencoder is based on the properties of a conventional Autoencoder, except instead of mapping an X→Z = E(X)→X = G(E(X)) distribution, it maps a Z→ X = G(Z)→ Z = E(G(Z)) distribution. Section 3.5 outlines the principles of autoencoding in greater detail.

1) Generate a random noise vector Z
2) Pass this noise vector to the generator to produce X = G(Z)
3) Pass this generated sample back to the original encoder to generate Z = E(G(Z))

4) Backpropagate using the mean-squared error between the original noise vector, and the encoded generated vector, updating the Encoder and the Generator.

Chapter 4 Generation Strategies

4.1 Generation Concepts

The system that has been trained thus far has four components:

- Generator
- Encoder
- Discriminator
- Linear Classifier

By feeding some unseen Z vector to the Generator, we are able to generate a new fake data sample which corresponds to this specific Z vector, which has not previously been seen by the system. Through the strategies implemented in the previous section, we have attempted to map all regions of the Z space to vectors which will produce valid data samples.

The Linear Classifier allows further semantic meaning to be applied to the Z space. The weight vector of the Classifier represents a plane of separation between the two classes. Generating a data sample from a Z vector along this plane will result in a sample which obtains ~50 % classification for each class. In contrast, generating a data sample from along the vector perpendicular to the plane produces a data sample further from the classification boundary, meaning it should exhibit features which more distinctly identify it as a member of its class.

The principle is to sample vectors from the Z space and feed them to the Generator to produce fake data samples of known classification. The following chapter outlines several policies and considerations for sampling from this space, as well as optimisations and alternative strategies.

4.2 Random Z Vector Sampling

The most obvious policy would be to choose a random point in the Z space, feed this Z vector to the Classifier to produce a class, and then feed the Z vector to the Generator to produce a sample. This works, and will produce some output, however it has some issues.

Vectors on or near the classification decision boundary will produce samples whose class is not well defined. The sigmoid output for the Classifier could be in the range from 0.4 to 0.6. This suggests that the classification for these data samples will not be certain. If the system that is generating new data samples cannot be certain of their classification, then these data samples are unlikely to be very useful. A better policy is outlined in the next section.

Additionally, randomly selected points may be beyond the regions of Z space which the system has been trained. That is to say that as the system is generally provided with Z vectors of unit length, generating a Z vector of length 100 is unlikely to produce meaningful results. Using this reasoning, all generated Z vectors should have length matching the length of the vectors used to train the system, which is typically unit length.

4.3 Critical Z Vector Data Generation

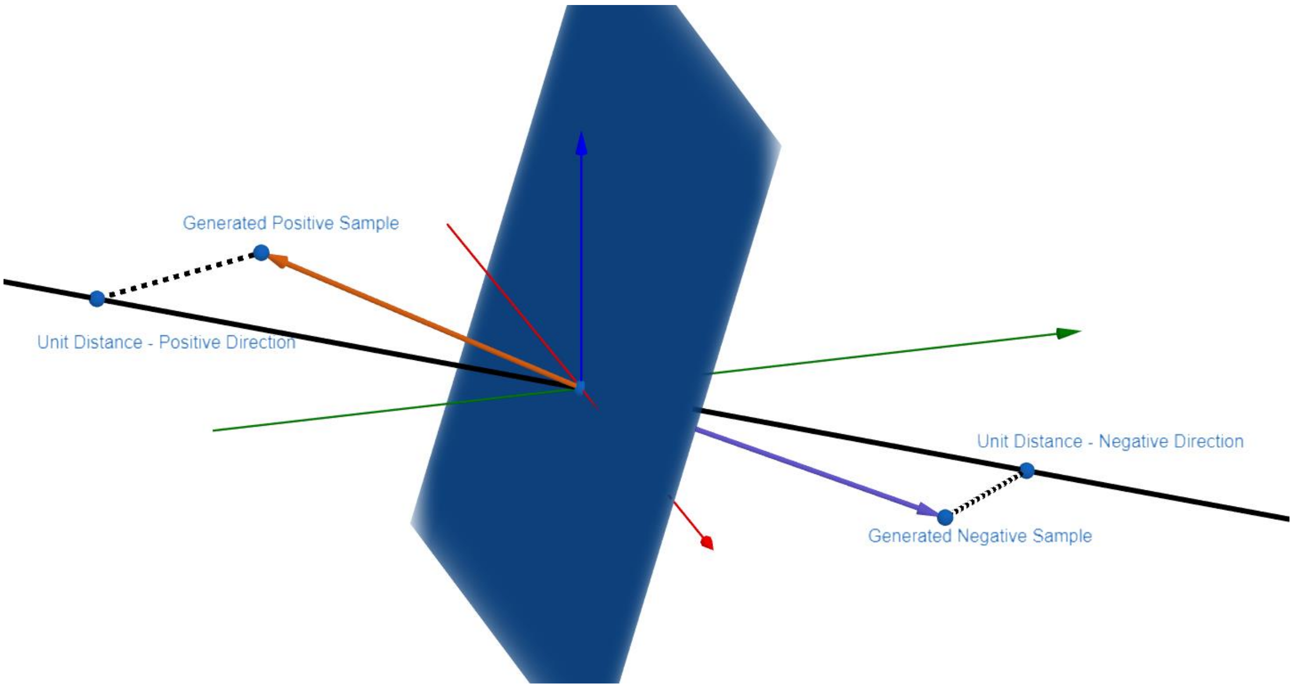
Generated Positive Sample
Unit Distance - Positive Direction
Unit Distance - Negative Direction
Generated Negative Sample

Figure 25 - Diagram of the policy for generating vectors along the direction of most change (perpendicular to the classification boundary)

The weight vector of the Linear Classifier constitutes the direction of maximal change in classification. That is to say that the sigmoid output of the Linear Classifier will change the most rapidly as the Classifier's input is moved along this vector. This vector has been represented in the above diagram by the black line. Similarly, the blue plane represents the boundary in which the classification of the Z vector changes from positive to negative.

Sampling points at unit distance in either the positive or negative direction of maximal change (black line) will result in classifications which are the furthest from the decision boundary. However, sampling from a single vector will result in only 2 outputs, 1 for a positive sample and 1 for a negative sample. We need to use a policy that respects these critical vectors, but introduces some variation.

To produce variation, vectors in the vicinity of the vector of maximal change should be sampled. A mathematical procedure has been developed to achieve this and is outlined as follows. A random vector orthogonal to the vector of maximal change is generated. This vector is subtracted from the maximal change vector, to produce a small permutation to the main vector.

$$\boldsymbol{maximalVector} = \textbf{The Linear Classifier Weight Vector, normalised to unit length}$$

$$\boldsymbol{randomVector} = \textbf{A random vector of Z dimensionality, with unit length}$$

$$\boldsymbol{orthogonalVector} = \boldsymbol{randomVector} - (\boldsymbol{maximalVector} \cdot \boldsymbol{randomVector}) * \boldsymbol{maximalVector}$$

$$\boldsymbol{offsetVector} = \boldsymbol{maximalVector} - \boldsymbol{orthogonalVector}$$

This offsetVector can then be scaled to either the positive or negative direction, corresponding to a data sample that is of class 0 or class 1. This vector is close to the direction of maximal change, and will produce a classification of high certainty. This offsetVector is represented in the above diagram as the orange and purple vectors.

This vector generation policy proved the most successful, and was generally used throughout the experiments.

4.4 Generated Sample Classification

Generated samples are paired with their corresponding classification. There are 2 procedures for which classification can be appended.

1. The initial Z vector used as the seed for the Generator can be classified, with this classification appended to the Generated Sample. That is to say that the data sample = G(Z) and data classification = C(Z)
2. The initial Z vector used as the seed for the Generator can be discarded. Instead, the generated sample can be passed to the Encoder to produce a new generated-encoded Z vector. This generated-encoded Z vector can be used for classification. That is to say that the data sample = G(Z) and data classification = C(E(G(Z)))

Ultimately, the second methodology was chosen, as the X -> Z transformation results in the compression of information, which ideally would be less lossy than the Z -> X transformation which expands information. However, an argument could be made either way, and this methodology required further exploration.

4.5 Generated Sample Certainty

The Linear Classifier provides a sigmoid transformation to its classification output, rather than a step function. As a result, the classification provides a sliding scale for its output: a sample could receive a classification of 0.99, indicating strongly that it belongs to class 1, or it could receive a classification of 0.4, indicating that it is likely to belong to class 0, however it is less likely to belong to class 0 than a sample with a classification of 0.01. The Classifier output can be considered as its ‘certainty’ (or ‘confidence’ however, this terminology overlaps with statistics terminology so is avoided). This means that not all generated samples are created equally; some samples are definitely of a given class, whereas some are less sure. There are 2 policies which were implemented to handle this situation.

One policy could be to discard any generated vectors that the encoder-classifier is unsure about. A sample in which the encoder-classifier provides an 0.55 likelihood of a binary class, is unlikely to be a good sample to train with. Some threshold range (for example, [0.3, 0.7]) could be used to discard any results that the system is unsure of that might fall within this range. This will provide only certain data samples to an external system.

This policy was initially experimented with, however was ultimately deemed not necessary when paired with the critical Z vector generation strategy. If a random sampling of Z vectors was used, this strategy should be considered and tested.

A second policy is to provide the vectors with a probabilistic class assignment, rather than a binary class assignment. For example, the encoder-classifier might determine a generated sample to be of class 1, and assign it a value of 0.99. In this case, it would be safe for that sample to be delivered to a user (for example, a separate neural network) with a classification equal to 1.

However, consider the case where the encoder-classifier assigns a sample a class value of 0.6. In this case, considering it as a member of class 1 may be inaccurate, as the system is not confident of its real class. In this case, the policy would assign this vector a class of 0.6. When this vector is then passed to a separate neural network for training, this network knows that it should not be confident in the value of this sample and will adjust its losses accordingly.

This policy was used in the final system. Further experimentation of its effectiveness is required.

4.6 Class-Proportional Generation

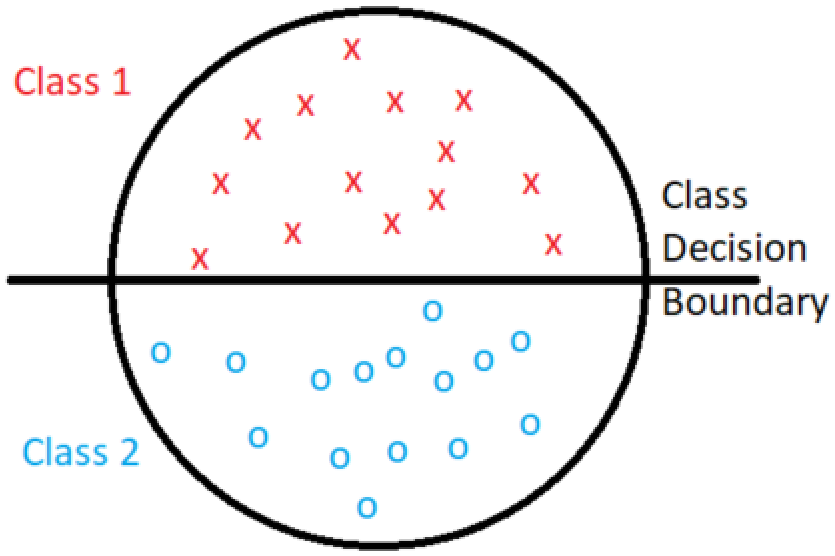

Figure 26 - Diagram of the latent space for BiCGAN trained on balanced data

The above diagram indicates a representation of the latent space for a balanced dataset consisting of approximately equal number of samples from each class (Sonar dataset).

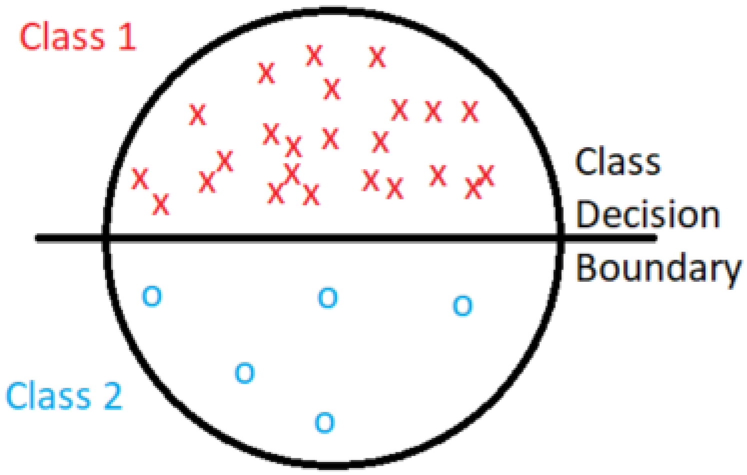

Figure 27 - Diagram of the latent space for BiCGAN trained on imbalanced data (no bias)

The above diagram indicates a representation of the latent space for an unbalanced dataset, consisting of a large proportion of a majority class. As the majority of samples are from the majority class, the latent vectors corresponding to these vectors will be significantly denser in the latent space region corresponding to the this class. Notice that this contradicts the Plum Pudding Loss function, which aims to spread the latent vectors evenly among the latent space.

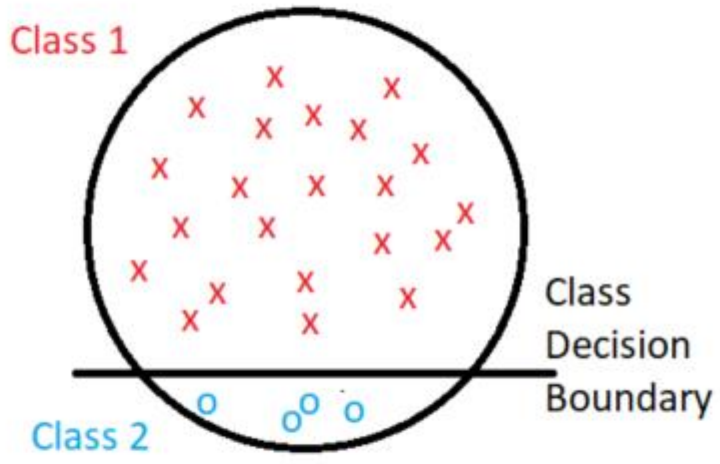

Figure 28 - Diagram of the latent space for BiCGAN trained on imbalanced data (with a Linear Classifier bias)

The above diagram indicates how a Linear Classifier bias effects the latent space when dealing with a majority class. The bias moves the class decision boundary to achieve an even density of vectors throughout the latent space.

It can be seen that there are 2 valid configurations for training the BiCGAN: either balance the data and do not include bias, or leave the data imbalanced and let the bias compensate for the uneven class distribution. The third configuration, of using imblanced data and no bias is not compatible with the goals of the Plum Pudding Loss function.

These configurations must be considered when generating new data of a given class. Results indicated that the best practice was to train the system with a balanced dataset, even if the underlying data was imbalanced. Then, when training a new system, data should be generated at a proportion approximately equal to the original imbalanced dataset. This allows a new system to learn the true ratio of the majority class to the minority class.

Chapter 5 Evaluation Strategies

5.1 Evaluation Concepts

An ongoing problem in the study of generative networks is how to quantitatively assess their performance. Many of the existing models attempt to generate new images, which allows for easy inspection of outputs by humans. Generated images can easily be checked for 'reasonableness', ie that the output is not just random noise, and other problems such as mode collapse. However, it is difficult to quantitatively compare the performance of different models – how can we measure how satisfactory the generated images are?

The following chapter outlines the methodologies which have been used to evaluate the effectiveness of the Generator.

Work by Shmelkov, K., Schmid, C., & Alahari, K. (2018) has attempted to develop a quantitative evaluation methodology similar to the procedure outlined in this report, applied to image datasets. Their paper 'How good is my GAN?' outlines similar results to the outcomes of this project.

5.2 Fresh Neural Network Training; is our generated data indistinguishable from the original data?

To quantitatively address the evaluation problem, we have developed a methodology which attempts to generate a set of data, and then use this data to train a fresh Neural Network Classifier. If the generated data is indistinguishable from the original data, then this fresh classifier will perform to the same degree of performance as if it were trained on the original data. The following diagram outlines this procedure.

5.2.1 Train the system on training data set

The data is split into a training set and a test set. The test set is held out until Step 4. The system is trained in accordance to the methodology previously outlined in Chapter 3, using the training data only.

Once the system has completed a specified number of epochs, the Generator, Encoder, Discriminator and Linear Classifier models are all saved, to be used separately with the following processes.

5.2.2 Generate a fake set of data samples

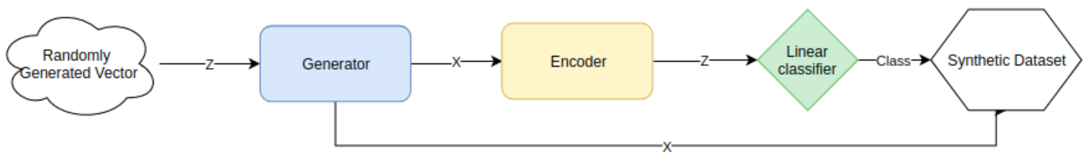

Figure 29 - Procedure to generate fake data samples

We first randomly generate a Z vector according to some policy outlined in Chapter 4.

We feed this Z vector to the Generator, to produce a fake sample X = G(Z). We store this fake X.

We then feed this fake X to the Encoder, then to the Linear Classifier to produce an estimate Class = C(E(G(Z))). This produces an X + Class pairing. This is repeated to produce a fake data set for a given size (typically 1000 to 100000 samples, depending on the dataset).

5.2.3 Train a Fresh Neural Network Classifier on the fake data samples

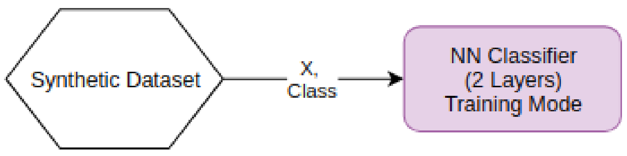

Figure 30 - Procedure to train a fresh classifier on fake data samples

The fake dataset is fed to a Fresh Neural Network Classifier as the training set. This new classifier learns only based on the generated fake samples.

The structure of the Fresh Neural Network structure is typically chosen to be a structure which achieves acceptable results on the original dataset. If the network can perform acceptably on the original dataset, then if we create indistinguishable data, it should also perform acceptably on this data.

5.2.4 Feed the hold-out test dataset to the Trained Fresh Classifier and evaluate its performance

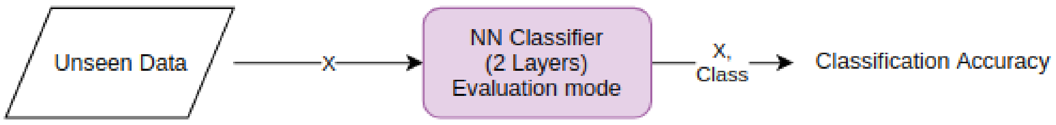

Figure 31 - Procedure to evaluate the performance of the fresh classifier on fake data samples

We now take the hold-out test set, and pass it to the Trained Fresh Neural Network Classifier. If the generated data is indistinguishable from the original data, the classifier should achieve the same classification accuracy as it achieves when trained with the original data.

5.2.5 Repeat with a 10-fold cross validation

The above process is repeated with a standard 10-fold cross validation procedure.

5.3 Comparison Graphs

One methodology to evaluate the performance of the generator is to compare the generated data samples to the original samples. However, it is difficult to effectively compare both fake and original sets, as they consist of many samples. It provides some results at a glance, and allows the user to determine whether the system is behaving, or whether it is producing noise. Several methodologies have been developed to compare these sets.

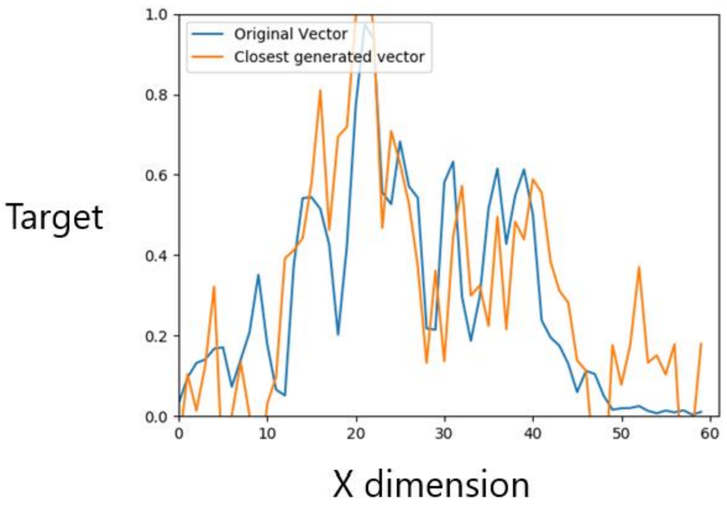

Figure 32 - Closest generated vector to an original vector

The first methodology is to compare the L2 distance from each original vector to the closest generated vector. This will generate a single comparison for each original sample. The above image is a plot of an original sample in blue, with the closest fake sample in orange.

It can be clearly seen that the generated vector is reasonably similar in shape to the original vector, with some variation. This methodology tests for complete input representation. If an input vector did not have any generated vector that is similar to it, then the system has not achieved a satisfactory representation of the input data, as not all modes of input data have been accounted for. This is conceptually similar to underfitting, but from a generation perspective.

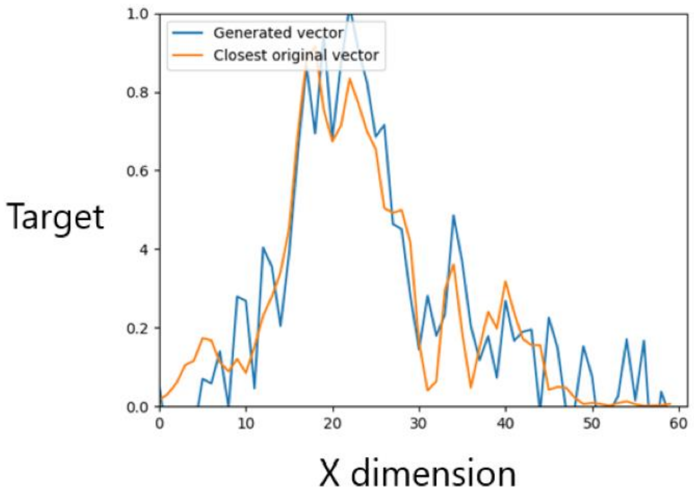

Figure 33 - The closest original vector to each generated vector

The above image shows the inverse of the first methodology: this is the closest L2 distance original vector to each generated vector. It can be seen that this generated sample matches an original sample very closely.

This tests whether the system is overfitting: if the closest original vector was exactly the same as the generated vector, then the system has simply memorised the input data. The system should not be able to exactly replicate the input set, but should rather be attempting to denoise it, by producing its best interpretation of the original sample.

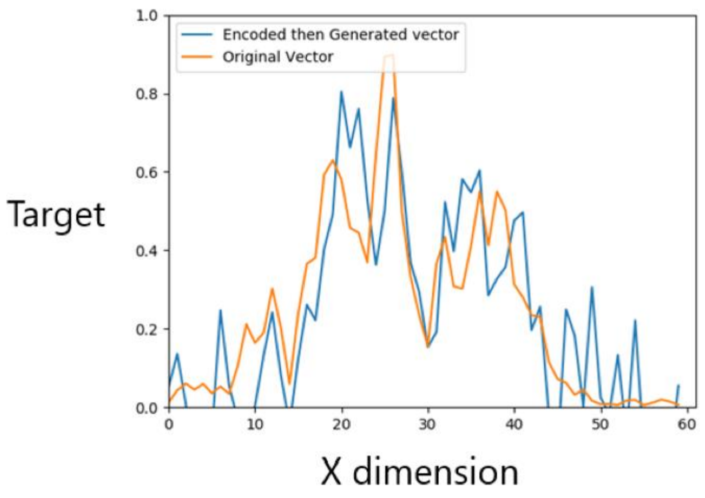

Figure 34 – Autoencoded Vector (ie encoded then decoded vector)

The above image represents the autoencoded vector. The original vector is passed to the encoder, then through the decoder, and compared with the original vector. The original vector is shown in orange, and the autoencoded vector in blue. It can be seen in this example that the results are reasonably similar, although some noise has been added towards the right of the graph.

This diagnostic shows the effectiveness of the semantic latent representation. The latent representation must be capturing some important elements of the original vector to be able to produce a similar reconstruction.

5.4 Latent Vector Autoencoding Misclassification

When a random latent vector is sampled from the Z space, it can be classified according to its position with respect to the Linear Classification boundary. It is then used to generate a fake data sample X, which has features matching the seed latent vector. If it is accurately generating features according to the seed, then this fake vector could be fed back to the encoder, and it should ideally be transformed back to the original latent Z vector. This is the principle of the flipped autoencoder loss as outlined in Section 3.5.

Some generated-encoded fake vectors result in a latent representation that has changed so much that the classification changes. That is to say that we could take a Z vector representing a classification of Class 1, then when we generate-encode this vector, we get a new vector E(G(Z)). However, when we go to classify E(G(Z)), we find that the Classifier decides it is now a member of Class 0. In this case, what label should be given to the fake data sample?

This loss of classification certainty was a major problem in initial versions of the system design. In initial designs using the Sonar dataset, up to 35 % of generated vectors were having their classification flipped during the generate-encode process. This drove the decision to include the Flipped Autoencoder Loss, which reduced the misclassification to 0 – 3 % in most cases.

The rate of autoencoding misclassification appears to be a useful methodology for evaluating the success of a model. To achieve a low autoencoding misclassification, the model must be accurately capturing meaning in the latent space, or else the Z vector would deteriorate through the generate-encode process.

Chapter 6 Experimental Results

6.1 Sonar Dataset

6.1.1 Dataset Context

The data set was contributed to the benchmark collection by Terry Sejnowski, now at the Salk Institute and the University of California at San Deigo. The data set was developed in collaboration with R. Paul Gorman of Allied-Signal Aerospace Technology Center. ("UCI Machine Learning Repository: Data Sets", 2019)

The dataset contains 111 patterns obtained by bouncing sonar signals off a metal cylinder at various angles and under various conditions, and 97 patterns obtained from rocks under similar conditions. The transmitted sonar signal is a frequency-modulated chirp, rising in frequency. The dataset contains signals obtained from a variety of different aspect angles, spanning 90 degrees for the cylinder and 180 degrees for the rock.

Each pattern is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency band, integrated over a certain period of time. The label associated with each record contains the letter "R" if the object is a rock and "M" if it is a mine (metal cylinder). The numbers in the labels are in increasing order of aspect angle, but they do not encode the angle directly.

6.1.2 Training Strategy

The BiCGAN was trained with a test / train split of 0.8. Data points were sampled in accordance with their original class labels (ie the training set was neither oversampled nor undersampled).

The BiCGAN was run for 20000 epochs. The following graph shows the convergence of the autoencoded loss over time.

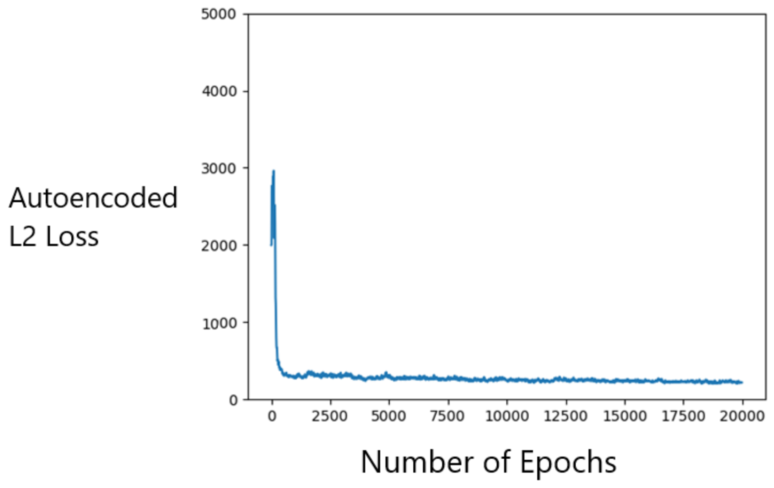

Figure 35 - Autoencoded Loss of Sonar BiCGAN Results. Converged at a loss of ~250 units

6.1.3 Image Comparisons

6.1.3.1. Context

Once training had completed, the following image comparisons were generated using the completed elements of the BiCGAN, ie a combination of the final Generator, Encoder, Classifier and Discriminator.

6.1.3.2. Closest Generated Sample to each Original Sample

The below figures display the closest generated vector to a random selection of original vectors. Qualitatively, the generated vectors appear to be close in representation, with additional noise introduced at the tails of the data, where the signal is typically smaller. Each original vector appears to have a representative generated vector, indicating that the generator is able to capture the entire data sample space. These images are consistent with results throughout the entire output set.

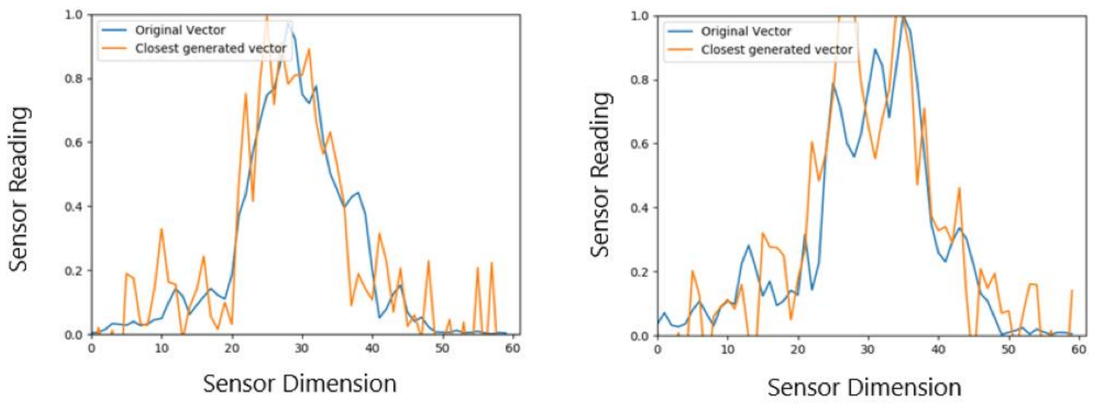

Figure 36 - 2 randomly chosen samples of class 0, and the closest generated sample

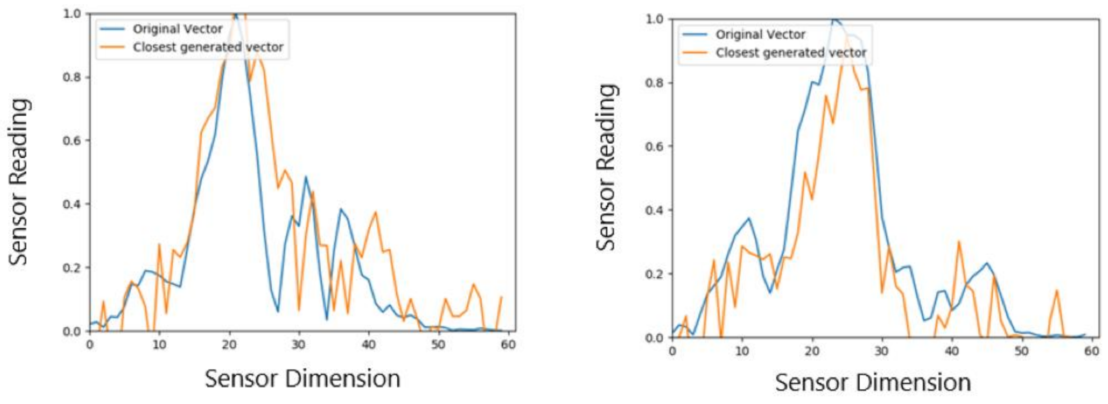

Figure 37 - 2 randomly chosen samples of class 1, and the closest generated sample

6.1.3.3. Closest Original Sample to each Generated Sample

The below figures represent 4 randomly generated samples in blue and show the closest original sample to each. Again, these vectors are qualitatively similar, but still include some noisy differences. This indicates that the generator is not simply memorising and replicating the original dataset.

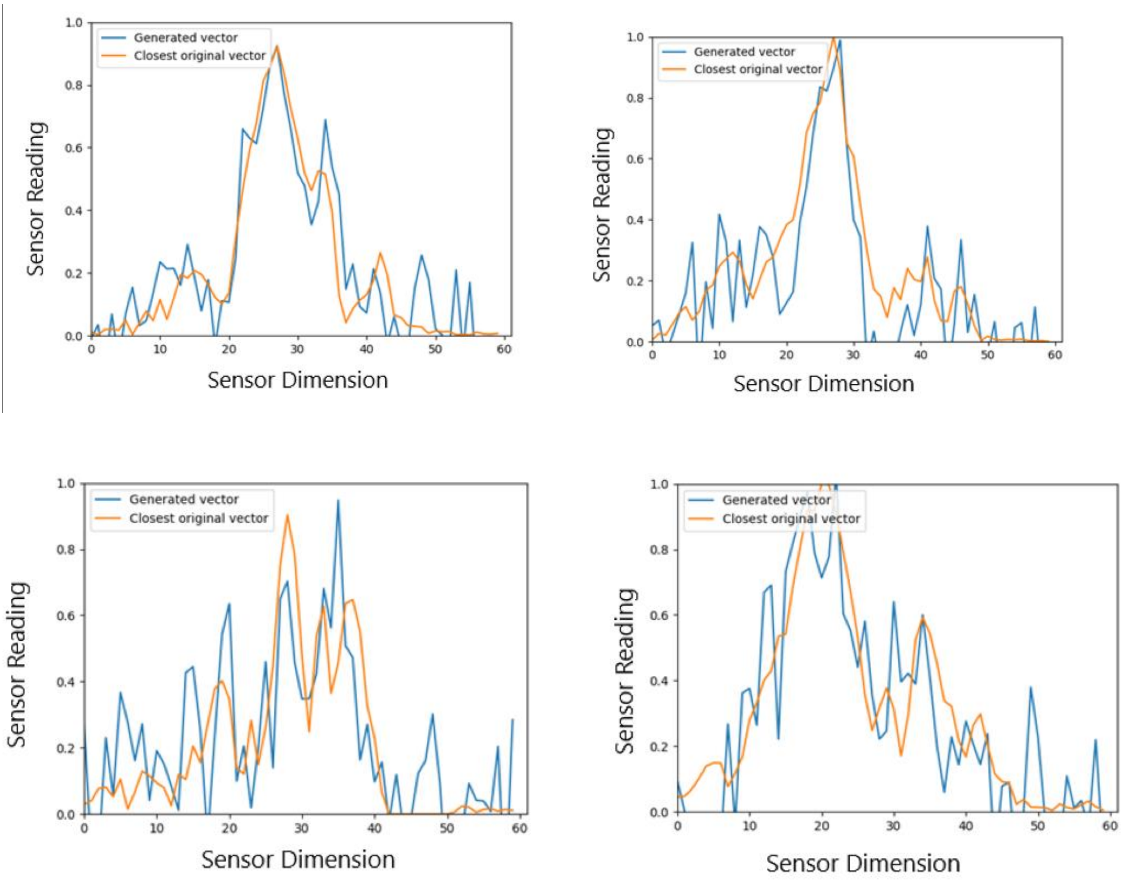

Figure 38 - 4 randomly selected Generated Samples, and the closest original sample to each

6.1.3.4. Encoded – decoded vectors

The below figures represent 4 randomly chosen encoded-decoded vectors. The original vector is seen in orange. This vector is fed to the encoder, then to the generator, and the resulting output is plotted in blue. The encoded-decoded representation can be seen to be clearly capturing the peaks of the original data. In some cases it appears to denoise the original data, but in others it seems to introduce noise.

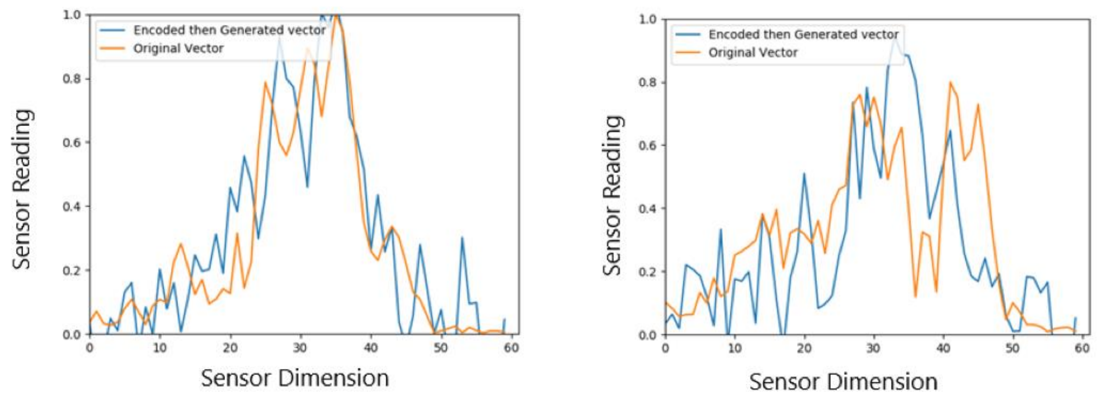

Figure 39 - Encoded-decoded vectors of class 0

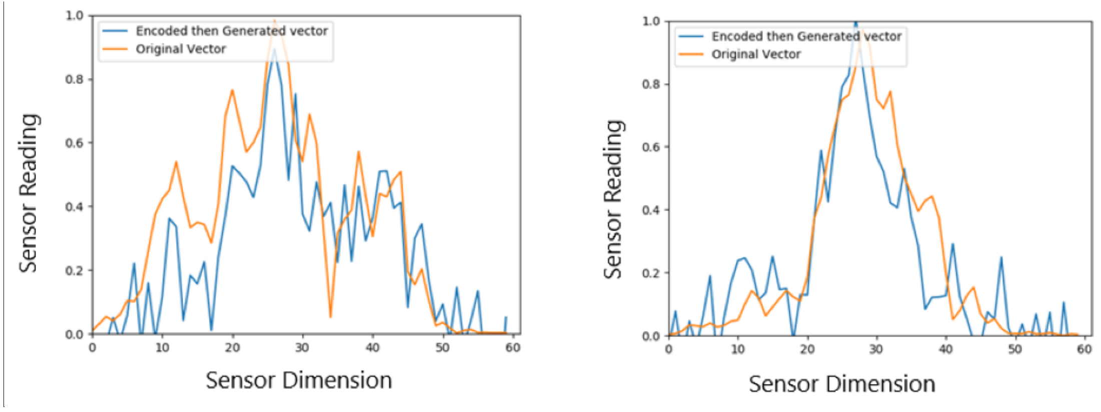

Figure 40 - Encoded-decoded vectors of class 1

6.1.4 Fresh Classifier 10-Fold Training Results

6.1.4.1. Context

The above results provided satisfactory justification that the system is performing acceptably. A 10-fold cross validation test was then performed using this system, using the procedure outlined in Section 5.2. The algorithm is outlined as follows:

```
For each 10-fold train/test split:
    Train the BiCGAN on the training data
    Use the trained Generator + Encoder + Classifier to generate some number of fake data
    samples
```

Train a new Neural Network Classifier on the generated data samples
Evaluate the performance of this Neural Network Classifier on the unseen test data

6.1.4.2. Results Graph

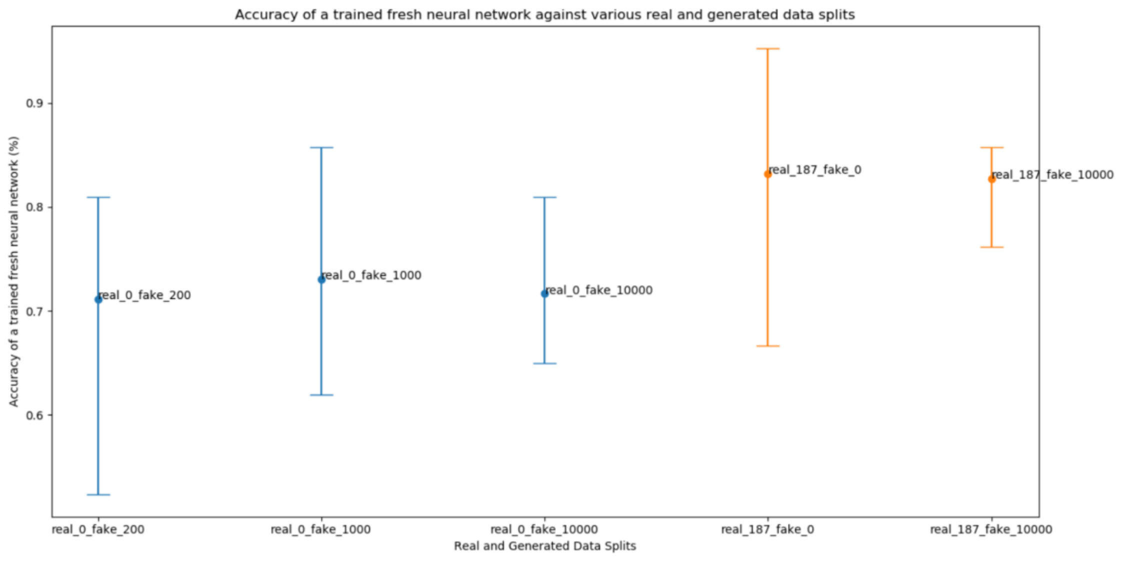

Figure 41 - Summary of 10-fold cross validation

The above graph shows the average results of the 10-fold cross validation on a variety of generation sizes. Each data point (real_X_fake_Y) represents the number of real samples and number of generated samples that were used to train the classifier. The blue results are entirely fake samples, and the orange results include the real samples in the training. The error bars represent the minimum and maximum result across each cross validation iteration.

The average result of the fresh classifier when trained on the generated vectors is approximately 73 %, compared to the result of a fresh classifier when trained on the original vectors of 83 %. An approximate reduction in accuracy of 10 % (corresponding to a reduction in performance of 12.5 %) has been observed when generating synthetic variables for the sonar dataset.

6.1.5 t-SNE Visualisations

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. It maps the multi-dimensional data to a lower dimensional space, by attempting to minimise the sum of Kullback-Leibler divergence of the overall data points using a gradient descent method.

The below diagram shows the original data with t-SNE applied. The red points represent the mines class, the blue points represent the rocks class. Some clustering can be observed, however the plot is generally unordered.

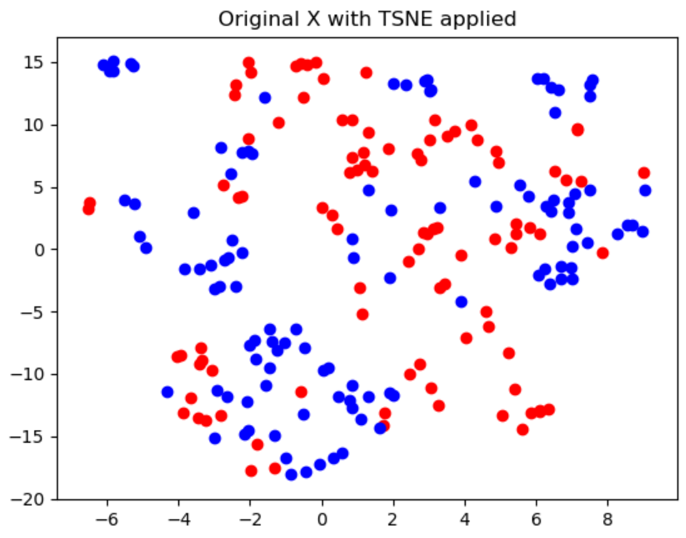

Figure 42 - Original X with TSNE applied

The below diagram shows the t-SNE applied to the encoded original data. Increased regions of similar classes appear to have been generated, however this is difficult to measure and remains a qualitative assessment.

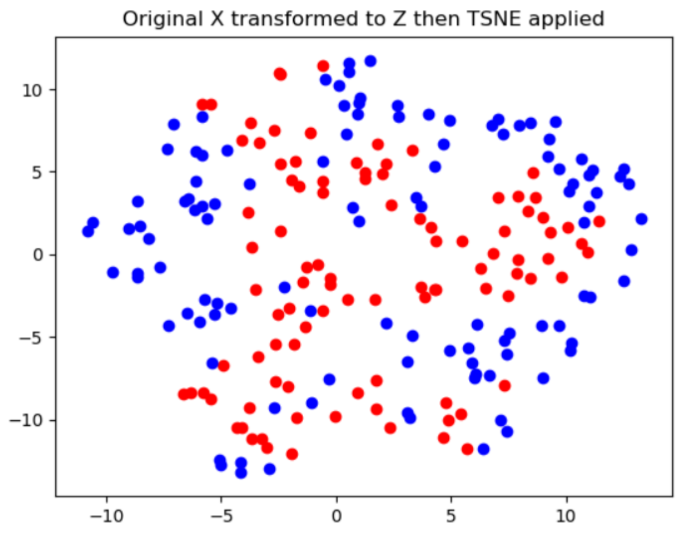

Figure 43 - Original X transformed to Z then TSNE applied

The below image shows the t-SNE applied to the generated data samples. These samples can be clearly grouped into distinct regions based on their class.

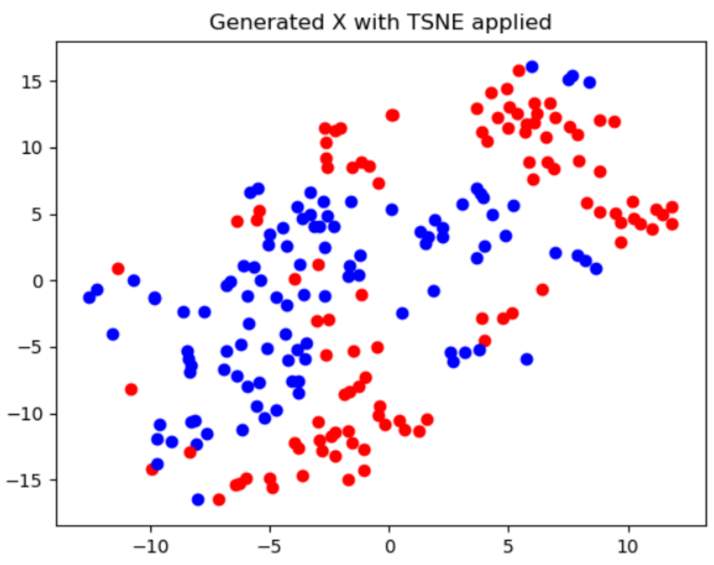

Figure 44 - Generated X with TSNE applied

The below image shows an example of the generated t-SNE vectors for reference. These Z vectors are sampled from the vector perpendicular to the classification boundary, so are easily split into two classes by the t-SNE process, as they are linearly separable. This figure is provided for reference to display how a perfect t-SNE transformation would be applied.

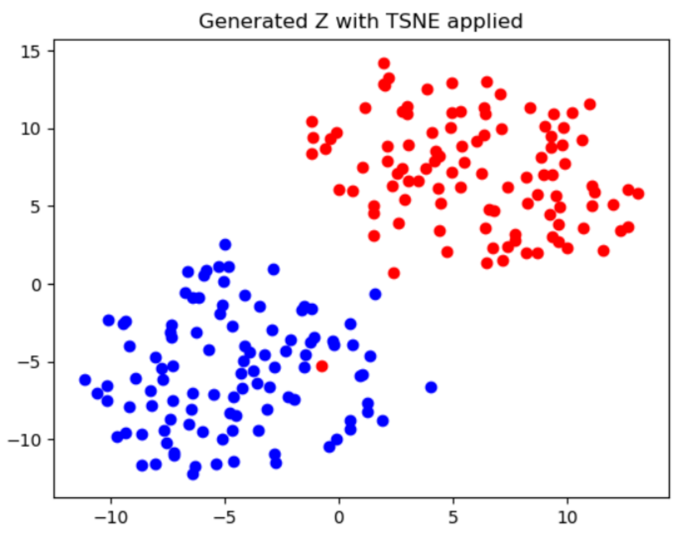

Figure 45 - Generated Z vectors with t-SNE applied

6.1.6 Latent Vector Interpolation

Recall Figure 25, which outlined the principle of sampling random vectors in the latent space in order to sample from regions near to the maximal change vector, to produce vectors that had some variation but were still close to the ‘ideal’ vector. The following figure represents an exercise in which the ideal vector is sampled, and then scaled as it transitions from the ideal vector of class 0, to the ideal vector of class 1.

The first image in the sequence is the data sample produced when using the exact latent vector of the ideal representation of class 0. The ideal latent vector corresponds to a generated vector which, when re-encoded and classified, should produce a sample with classification class 0.

This vector is then progressively reduced in magnitude, until it reaches a scaling of 0, representing the origin of the latent space. The origin in the latent space represents a region in which generated vectors will not be well-classified by the Classifier. These vectors will correspond to a classification of approximately 50 % class 0, and 50 % class 1. These samples can be considered as valid samples, however of uncertain class.

The vector is then negatively scaled, until it reaches a scaling of -1. At this point, the vector corresponds to the ideal representation of class 1. This region has the same properties as described for class 0, however reflected about the classification boundary in the latent space.

The results pictured in the below figure appear to be of high quality. The ideal classes both appear to be representative of a sample from their respective class. In addition, the transition between samples is smooth, indicating a healthy latent space encoding which permits valid samples to be drawn from any latent vector.

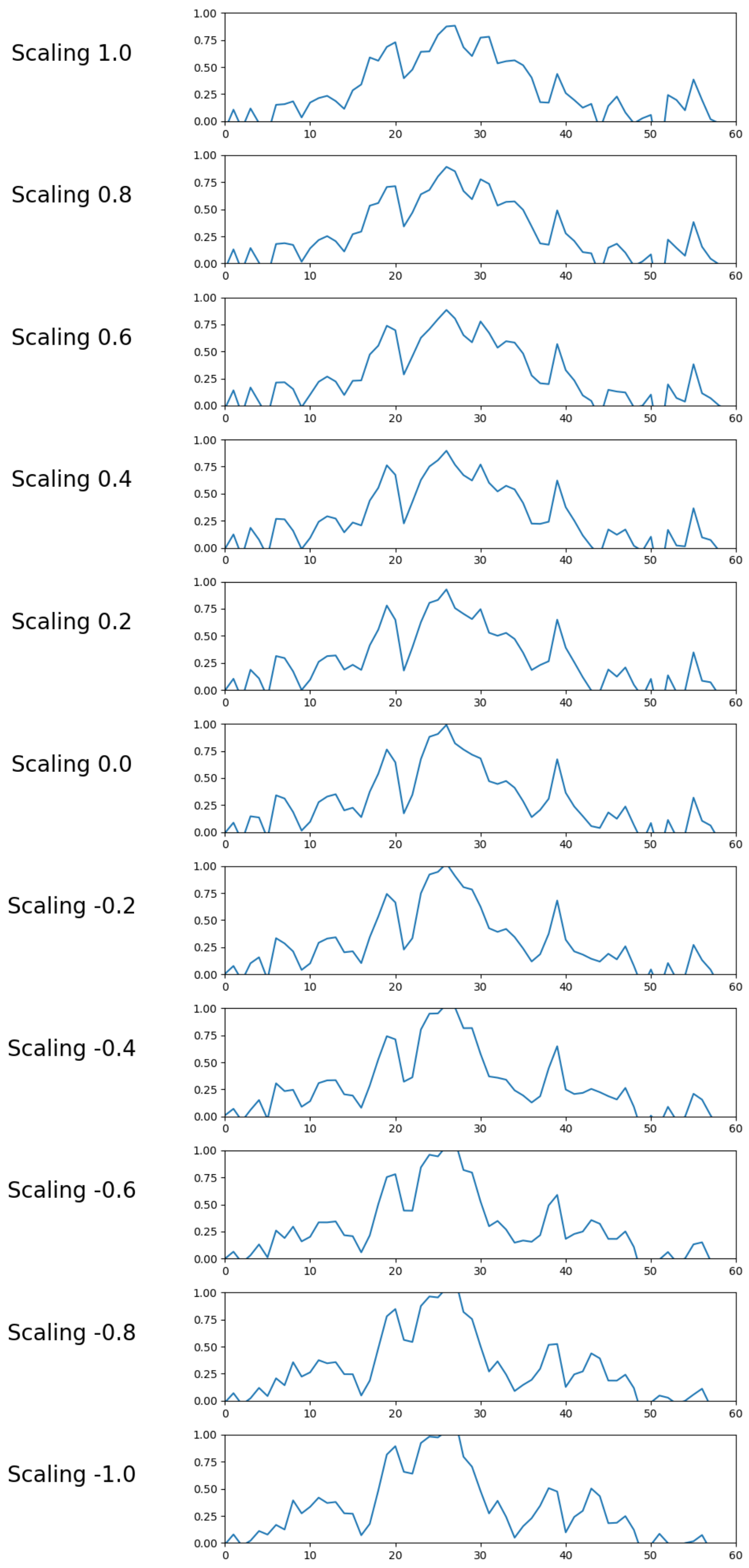

Figure 46 - Latent Vector Interpolation, from the ideal Class 1 to the Ideal Class 0

6.2 Credit Card Dataset (Minority Sampling)

6.2.1 Dataset Context

The Credit Card Dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where there were 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. ("Credit Card Fraud Detection", 2019)

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, the original features and more background information about the data cannot be provided. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'.

For the purposes of this project, the Time and Amount columns were dropped, as attempting to generate these values will require special consideration. In addition, features 10, 12 and 14 were dropped, due to these dimensions having the most outliers. This left the remaining 25 principal components to be used as the data sample.

6.2.2 Training Strategy

The BiCGAN was trained with a test / train split of 0.8. Due to the significant class imbalance, the minority class was oversampled to provide class data in a 50/50 split. Each batch consisted of 50 % majority class and 50 % minority class samples. The BiCGAN was run for 1500 epochs.

6.2.3 Image Comparisons

6.2.3.1. Context

Once training had completed, the following image comparisons were generated using the completed elements of the BiCGAN, ie a combination of the final Generator, Encoder, Classifier and Discriminator.

6.2.3.2. Closest Generated Sample to each Original Sample

The below figures display the closest generated vector to a random selection of original vectors. Each original vector appears to have a representative generated vector, indicating that the generator is able to capture the entire data sample space. These images are consistent with results throughout the entire output set.

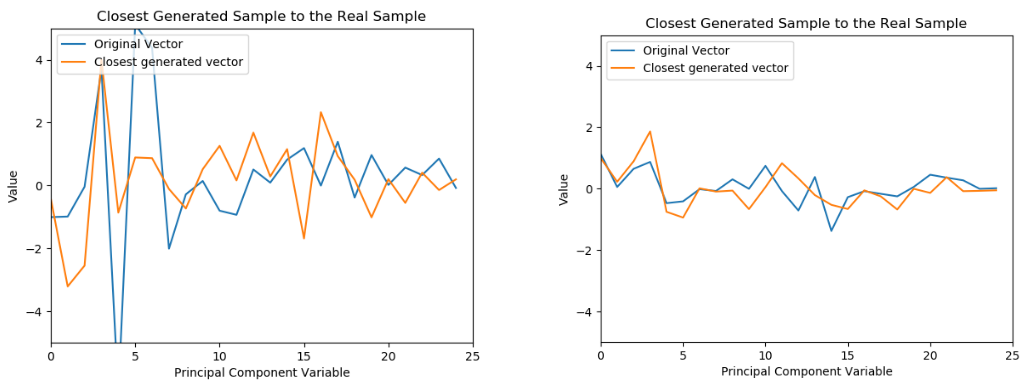

Figure 47 - 2 randomly chosen original samples, with the closest generated sample to each

6.2.3.3. Closest Original Sample to each Generated Sample

The below figures represent 2 randomly generated samples in blue and show the closest original sample to each. Again, these vectors are qualitatively similar, but still include some noisy differences. This indicates that the generator is not simply memorising and replicating the original dataset.

However, note that these generated samples are much closer to the original when compared to the equivalent results of the Sonar dataset. This suggests that the Credit Card Dataset might be overfitting.

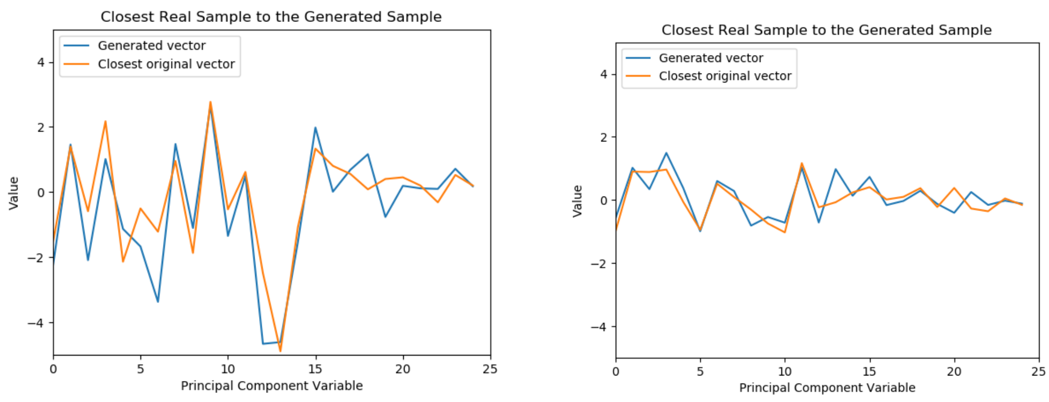

Figure 48 - 2 randomly chosen generated samples, with the closest original sample to each

6.2.3.4. Encoded – decoded vectors

The below figures represent 2 randomly chosen encoded-decoded vectors. The original vector is seen in orange. This vector is fed to the encoder, then to the generator, and the resulting output is plotted in blue. The encoded-decoded representation can be seen to be clearly capturing the peaks of the original data. In some cases it appears to denoise the original data, but in others it seems to introduce noise.

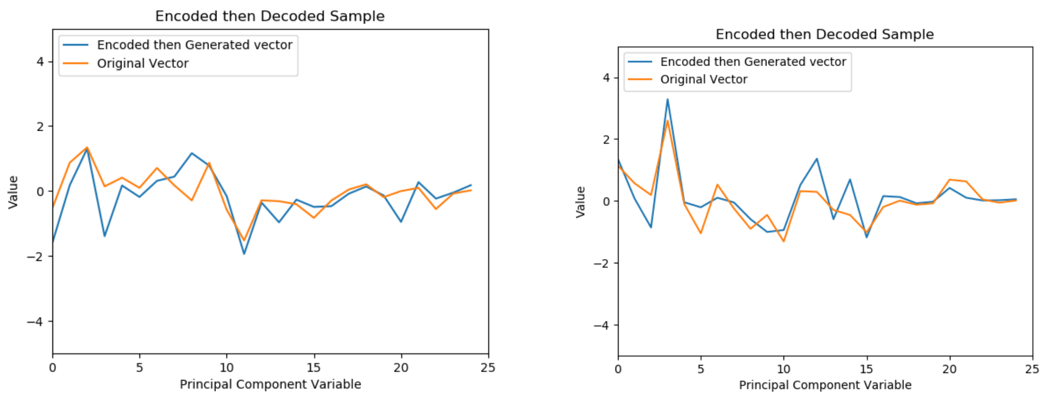

Figure 49 - 2 randomly chosen original vectors are encoded and decoded

6.2.4 Generated Sample Batch Comparisons

Due to the considerably skewed class distribution, additional checks were implemented to ensure the minority class generator was behaving acceptably, that is to say that it had not regressed to mode collapse, and was producing outputs that look like minority class members.

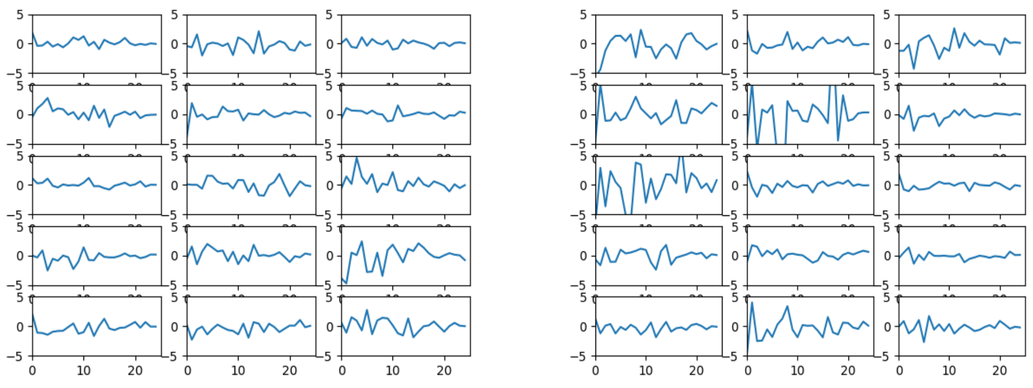

Figure 50 - (Left) Real Data: Majority Class | (Right) Generated Data: Majority Class

The above figure shows 15 randomly chosen members of the majority class from both the original data (left), and the generated data (right). By qualitative visual inspection, there do not appear to be any obviously erroneous generated data samples. The generator appears to be matching the majority class.

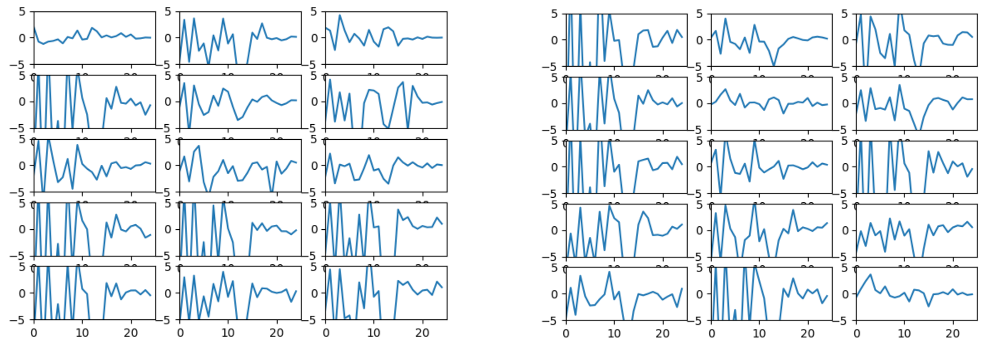

Figure 51 - (Left) Real Data: Minority Class | (Right) Generated Data: Minority Class

The above figure shows 15 randomly chosen members of the minority class from both the original data (left), and the generated data (right). By qualitative visual inspection of the minority class, it would appear that the generator may have experienced mode collapse, and is repeatedly generating a sample with large peaks for dimensions 0 to 10, which trails off in later dimensions.

However, when the generated data is considered in context of the real data, it appears that the real minority class data samples generally occur in a very similar shape. Comparing the generated samples to the original data sample shows that the generator actually appears to be correctly matching the distribution of real minority class samples.

6.2.5 Fresh Classifier Training Results

6.2.5.1. Baseline (Train a fresh classifier on the original data)

The results of a classifier must be considered with respect to a network that has been trained purely on the original data. The following confusion matrix shows the accuracy when a conventional network is trained on the training data, and evaluated on the unseen test data, with no sampling strategy.

	Actual Fraud	**Actual Non-fraud**
Predicted Fraud	60	9
Predicted Non-fraud	38	56855

Accuracy: 0.999 Precision: 0.869 Recall: 0.612

The following confusion matrix shows the results when a conventional network is trained on the training data, evaluated on the unseen test data, with a 50/50 ratio oversampling strategy.

	Actual Fraud	**Actual Non-fraud**
Predicted Fraud	78	57
Predicted Non-fraud	20	56807

Accuracy: 0.998 Precision: 0.553 Recall: 0.795

As expected, oversampling the data causes the precision to decrease and the recall to increase. These parameters can be balanced by adjusting the oversampling ratio.

6.2.5.2. Equal Sampling (Train a fresh classifier on 50/50 samples of generated data)

The below confusion matrix shows the results when a fresh classifier was trained on generated data, with the data being generated at a 1:1 split between the majority and minority class. 50000 majority class and 50000 minority class samples were generated and used to train the classifier.

	Actual Fraud	**Actual Non-fraud**
Predicted Fraud	87	755
Predicted Non-fraud	11	56109

Accuracy: 0.999 Precision: 0.268 Recall: 0.877

These results have deteriorated when compared to using the original data. In particular, the precision has fallen by approximately 0.3.

6.2.5.3. Sampling to match the real ratio of class imbalance

The below confusion matrix shows the results when a fresh classifier was trained on generated data, with the data being generated at a 100:1 split between the majority and minority class. The real ratio of the majority to minority class is approximately 500:1.

50000 majority class and 500 minority class samples were generated and used to train the classifier.

	Actual Fraud	**Actual Non-fraud**
Predicted Fraud	58	20
Predicted Non-fraud	40	56844

Accuracy: 0.999 Precision: 0.744 Recall: 0.592

These results are very similar to the results when the network was trained on the real data. This was the strongest result achieved by the BiCGAN and constitutes a relatively successful outcome. It appears that the fresh classifier only has a reduction in precision of 0.125 and reduction in recall of 0.02 when instead trained with generated data produced by the BiCGAN.

6.2.6 Latent Vector Interpolation

The principle of Latent Vector Interpolation has been outlined in Section 6.1.6. The below figure shows a sampling across the latent space for the Credit Card Dataset.

The result for the ideal fraudulent case (first image) appears to be in line with previous results in Figure 50. The ideal case seems to match the typical fraudulent data sample, with the prominent spikes in the 0-5th dimension region, and again in the 10th dimension region.

The result for the ideal non-fraudulent case (last image) also appears to be in line with typical non-fraudulent samples. There are no particular spikes in the data sample.

An interesting transition occurs in the reduction of scaling from 0.4 to 0.2. This region marks a significant change in mode: the tell-tale spikes in the early dimensions appear to immediately be reduced. This contrasts the gradual and smooth transitions which occurred for the Sonar dataset, and also in the other scales within this dataset.

This sharp transition could be caused by the unimodal nature of the fraudulent data samples. This transition may represent the split in the vector space in which on the fraudulent side (0.4 scaling) the latent vectors map to similar unimodal samples, and on the less fraudulent side (0.2 scaling) these vectors no longer map to this single mode. Further discussion of the modal nature of the data is provided in Section 7.2.

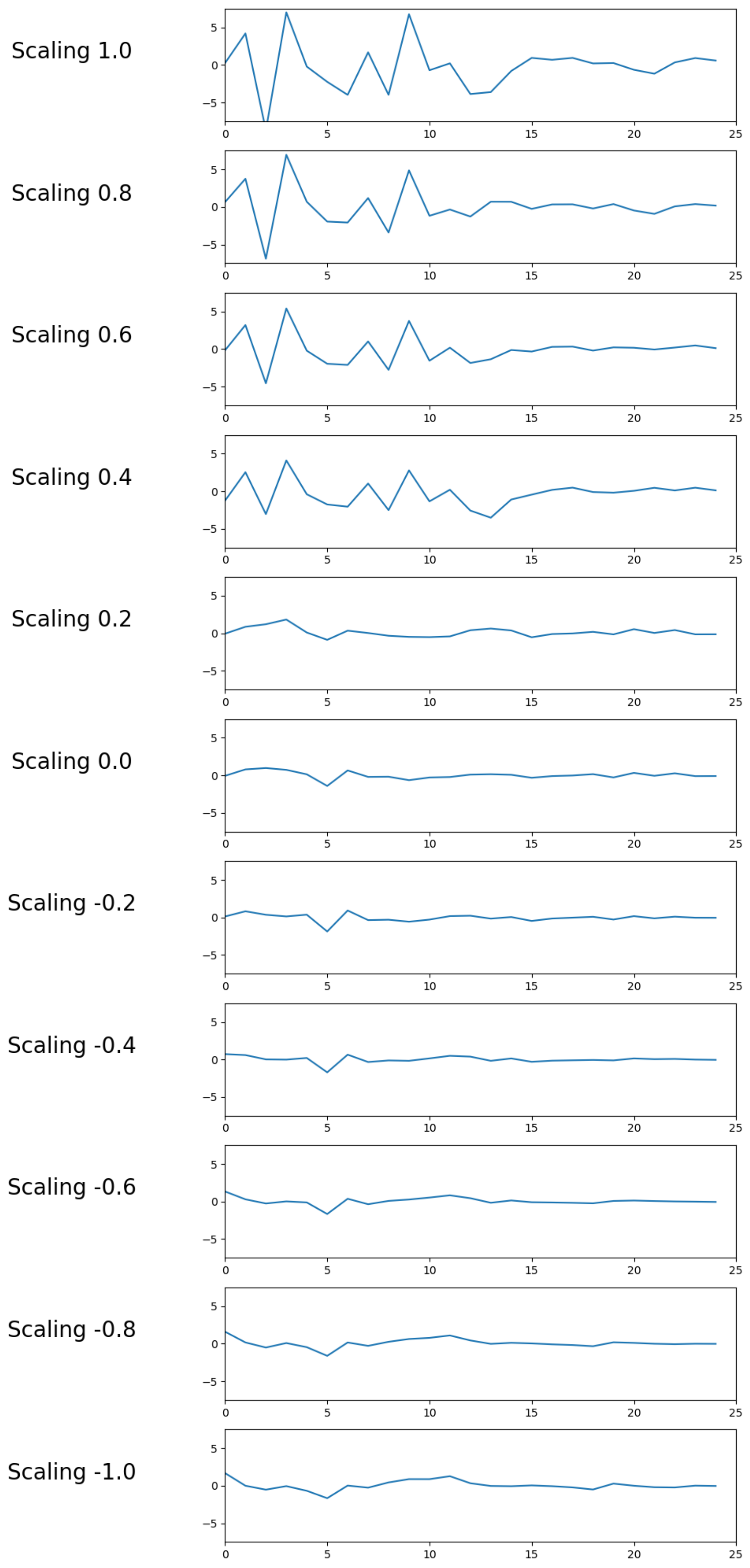

Figure 52 - Latent Vector Interpolation, from the ideal Class 1 to the Ideal Class 0

Chapter 7 Discussion

7.1 Analysis of Sonar Data Results

7.1.1 10-fold Cross Validation Results

If the measure of success is the ability to train a neural network to classify a dataset, then an overall classification accuracy of 73 % is not a practically useful result. However, classification accuracy is usually used as a quantitative measurement and comparison of different methodologies, and as classification was not the intent of this project this value should not be considered a perfect metric.

The intent of this project was to generate new data samples based on a classification condition. The image comparisons portray that this has been qualitatively achieved.

One of the motivations for the project was to determine the effect of supplementing real data samples with generated data samples. The furthest right orange bar in Figure 53 indicates the result of the classifier when trained with the original 200 data samples, and supplemented by 10000 generated data samples. The second from the right orange bar indicates the result of the classifier when trained with only the 200 original data samples, and no generated supplementation.

The classification accuracy of both these cases can be seen to remain very similar. It appears that adding 10000 fake samples has not changed the ability of the classifier to distinguish between classes. This result indicates that the generated data samples do not decrease the performance of a network trained on these results.

7.1.2 Reduction in Accuracy Across an Extended Process

The average result of the fresh classifier when trained on the generated vectors is approximately 73 %, compared to the result of a fresh classifier when trained on the original vectors of 83 %. This reduction should be considered with respect to the level of information loss caused by the extended procedure. It is unrealistic to expect that the data generation process would increase the accuracy on a small and relatively low-dimension dataset such as the Sonar Dataset. Instead success should be measured based on the degree of performance reduction when compared to the optimal accuracy.

The fresh classifier being trained and tested on some data can be considered as a single step in the classification process. The generation of data samples can therefore be considered as a second step. Assuming that the upper bound result of the classifier on the real training data is 83 %, then the effect of introducing a second step to the process should be considered with respect to this baseline. The result of this calculation is that the generation process has an ‘information conservation rate’ of 87 % ($\frac{73\ \%}{83\ \%} = 87\ \%$).

7.2 Analysis of Credit Card Data Results

The baseline result of a fresh classifier trained with the original data, is 0.869 precision and 0.612 recall. The BiCGAN result achieved 0.744 precision and 0.592 recall, when trained on a class ratio matching the original imbalance. This is a change in precision of 0.125 (a reduction of 15 %), and a change in recall of 0.02 (a reduction of 4 %). These results are much more promising than the results obtained for the Sonar Dataset.

The success of the results indicates that the dataset may be flawed. One explanation for the ability of the fresh classifier to achieve results so close to the real classifier would be that the misclassified samples are outliers and cannot be improved upon. The examination of data in Figure 54 appears to indicate the majority of the fraudulent transactions to be of a similar mode. It may be the case that the fraudulent transactions that are not of that mode could not be detected by either classifier, so the results are similar as they both achieve the local minima. This is one hypothesis for the quality of these results, and requires further investigation.

The balance of precision and recall could be adjusted by modifying the ratio of class samples which were given to the classifier. For example, when oversampled at a 50/50 split, the classifier would incorrectly predict non-fraudulent cases as fraudulent. As the minority class split is reduced, the recall reduces, and the precision improves. A stable balance is found with an imbalance similar to the real class imbalance. This is a logical result, however it was initially not expected that the prior probability would not be a major contribution to the result.

Chapter 8 Future Work

8.1 Image Datasets

Experiments have been performed utilising datasets of dimensionality 30 and 60. Upscaling to image datasets would require utilising dimensionalities of at least 32 x 32 pixels; at least 1024 dimensions. This would also require increasing the depth and breadth of the Generator, Encoder and Discriminator network structure, as they would likely need to be redesigned as convolutional networks. Training these deeper and more complex networks would require a significant increase in computational resources.

For most image datasets, such as CIFAR10, the classifier would also need to be extended to handle multiple classifications.

If using images, the simple reconstruction loss would not be able to be used as a metric to determine convergence. However, this would not affect the training, and should only influence the diagnostics. A useful diagnostic for images could be to generate a random image and display it to the user – as training improves, these images should gradually become more realistic, allowing the user to view the improvement in accuracy over time. This would serve the same function of the reconstruction loss; as providing a diagnostic to the user.

8.2 Multiple Classes

The experiments have been performed with binary class datasets only. To manage multiple classes, some minor modifications would be required.

The current system divides the latent space into two regions, one for each class. N-class datasets require the latent space to be linearly separated into N regions. To do this, the Linear Classifier would need to be modified to produce a one-hot softmax encoding, rather than a sigmoid output. Further to this change, the loss function would need to be amended (likely to a cross-entropy loss function).

The vector generation strategy would need to be amended to be compatible with an N-dimensional softmax classification vector. Current generation strategies involve choosing a vector near the vector of maximal change, however this concept loses some intuitive meaning beyond a 2-dimensional split.

Instead, vectors would need to be sampled from nearest the vector that corresponds to the desired class. Sampling samples near to this vector should result in samples that are most isolated from adjacent classes. This is easy to conceptualise when imagining a 3 dimensional vector space, defined by 3 basis vectors. Each basis vector constitutes the direction most associated with 1 of 3 classes. Between each pair of vectors is a decision boundary, where the classification changes. This concept extends to N dimensions, however becomes harder to visualise. Each of the N vectors forms a basis, and a decision boundary is formed between a given vector and N-1 adjacent vectors.

Multiclass training and generation should be easily achieved utilising the above methodology, however would constitute significantly more research and is beyond the scope of this report.

8.3 Tuning Hyperparameters

The convergence rate appeared to be very sensitive to the learning rate of the optimizers. Increasing the Generator and the Encoder learning rate was found to immediately destabilize the system. Tuning the learning rate of the Discriminator was found to significantly improve the speed in which the system converged.

Further exploration could be undertaken to determine the effect of learning rates on convergence rate. In addition, new optimizers could be explored and experimented with to examine their effects.

8.4 Tuning Network Structure

The starting point for the Generator, Encoder and Discriminator was to utilise hidden layers with 512 nodes. This achieved convergence, however it was very slow. The network width was progressively reduced until the system was unable to converge. For the Sonar Dataset, non-convergence occurred at a width of 16 hidden nodes. Reducing the network width obviously improved the iteration time, at the cost of complexity.

Ultimately, a width of 64 hidden nodes was chosen for each hidden layer. This was considered to be safely beyond the critical 16 node layer, without severely impacting iteration time. Experimentation was ceased at this point, as tuning the perfect layer width is an exercise which could proceed indefinitely.

Further work would involve analysing the effects of decreasing the hidden layer width. Additionally, the width of each network component (Generator, Encoder, Discriminator) should be experimented with individually. This will allow a quantitative determination of the optimal network structure.

8.5 Balancing Losses

The introduction of multiple loss functions could mean that some loss functions are being drowned out. The Generator and the Encoder are each affected by several different loss functions measuring different outputs. The current design has not provided any scaling to these loss functions.

Further experimentation could involve increasing the strength of the signals provided by each loss function, to determine its overall effectiveness. This would largely be an iterative and time consuming process, however could provided quantitative justification for each component of the system.

8.6 Measuring Convergence

The current BiCGAN system measures convergence by examining the L2 loss from the encoded-decoded transformation of the original dataset, the performance of the classifier on the training set, and the performance of the classifier on the test set.

The performance of the classifier only assesses the performance of the encoder and the classifier, and does not consider the performance of the generator and encoder. As such, it makes it a weak method for measuring the performance of the system as a whole.

The encoded-decoded loss is also a flawed measure of convergence. This measurement only utilises the encoder and the generator, and neglects the discriminator and classifier. Encoded-decoded loss performed very poorly on the Credit Card dataset as it fluctuated significantly during the training cycle and never really stabilised. However even though this metric did not converge, the classification performance was still observed to improve over time. In addition, the quality of each generated samples appeared to be noticeably improving when testing over multiple epochs, suggesting that the system was in fact improving.

Ideally, some novel methodology for testing loss should be derived which considers all components of the system, while still considering the balancing act in the GAN minimax game.

Chapter 9 **Conclusion**

A modification to the existing BiGAN structure was produced, which implemented a Classifier. This new component enabled the system to be extended to enable the generation of synthetic data samples of a given class.

Novel methodologies were introduced to the new BiCGAN, including the Plum Pudding Loss, and a Flipped Autoencoder for latent variable preservation. Multiple strategies were explored for sampling latent vectors to produce effective generated data samples, and a novel strategy of critical Z vector data generation was utilised.

The BiCGAN was applied to the UCI Sonar Dataset with reasonable results. When tested with 10-fold cross validation, the BiCGAN was able to generate data samples that allowed a fresh neural network to be trained to achieve 73 % accuracy, compared to an 83 % that was achieved when using the original, unmodified data samples.

The BiCGAN was applied to the Credit Card Dataset, a heavily imbalanced dataset with a class ratio of approximately 500:1. Different sampling methodologies were explored to determine their effectiveness of training the BiCGAN. Ultimately, the BiCGAN was able to generate data samples that allowed a fresh neural network to be trained to achieve a performance of 0.744 precision and 0.592 recall, compared to 0.869 precision and 0.612 recall that was achieved using the original, unmodified data samples.

Using the methodologies explored in this report, the BiCGAN could be applied to further datasets, including images, in order to enable conditional generation of data samples.

Chapter 10 **References**

1. Donahue, J., Krähenbühl, P., & Darrell, T. (2016). Adversarial feature learning. arXiv preprint arXiv:1605.09782.
2. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio (2014). Generative Adversarial Networks. arXiv: 1406.2661
3. Dumoulin, V., Belghazi, I., Poole, B., Mastropietro, O., Lamb, A., Arjovsky, M., & Courville, A. (2016). Adversarially learned inference. arXiv preprint arXiv:1606.00704.
4. Zhang, J., Dang, H., Lee, H. K., & Chang, E. C. (2018). Learning Inverse Mappings with Adversarial Criterion. CoRR.
5. 2. Over-sampling — imbalanced-learn 0.5.0 documentation. (2019). Retrieved 1 December 2019, from https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html
6. Resampling strategies for imbalanced datasets | Kaggle. (2019). Retrieved 1 December 2019, from https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets
7. Despois, V. (2019). Latent space visualization. Retrieved 1 December 2019, from https://ai-odyssey.com/2017/02/24/latent-space-visualization%E2%80%8A/
8. Despois, V. (2019). Autoencoders. Retrieved 1 December 2019, from https://ai-odyssey.com/2017/02/07/autoencoders%E2%80%8A/
9. UCI Machine Learning Repository: Data Sets. (2019). Retrieved 1 December 2019, from https://archive.ics.uci.edu/ml/datasets.php
10. Credit Card Fraud Detection. (2019). Retrieved 1 December 2019, from https://www.kaggle.com/mlg-ulb/creditcardfraud
11. Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.
12. Shmelkov, K., Schmid, C., & Alahari, K. (2018). How good is my GAN?. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 213-229).