VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Genetic Programming Encoder for Autoencoding

Finn Schofield

Supervisor: Andrew Lensen

Submitted in partial fulfilment of the requirements for
Master of Artificial Intelligence.

Abstract

Autoencoders are powerful models for non-linear dimensionality reduction. Due to autoencoders conventional reliance on neural networks, it is difficult to interpret how the high dimensional features relate to the low-dimensional embedding. Several approaches have attempted to replace neural networks in autoencoders with genetic programming (GP) to find interpretable models. However, for the purposes of interpretable dimensionality reduction, we argue that replacing only the encoder with a GP individual while is sufficient. In this work, we propose the *Genetic Programming Encoder for Autoencoding* (GPE-AE). GPE-AE uses a multi-tree GP individual as an encoder, while retaining the neural network decoder. We demonstrate that GPE-AE is a competitive non-linear dimensionality reduction technique compared to conventional autoencoders and another GP based method. We also evaluate the quality of two-dimensional visualisations produced by our method, and highlight the value of functional mappings by demonstrating insights that can be gained from interpreting the GP encoders.

Acknowledgments

Thank you to Andrew for all your help over the last few years with my studies and academic development. The research opportunities you've provided me have been truly valuable, and in many ways this work represents the culmination of them.

Thank you Holly for all you do. I wouldn't have been able to finish this without your unconditional love and support.

Contents

Figures

Chapter 1

Introduction

Machine learning algorithms allow us to be better understand and explain data by learning important patterns that can be used for inference [27]. In contrast to supervised learning, where models are trained to learn a pre-identified structure, unsupervised machine learning methods can be used to *discover* structure in data without the need for the data to be labeled [50].

Dimensionality reduction is an important technique in unsupervised machine learning [22]. Dimensionality reduction techniques learn representations of data in a lower-dimension space, making it potentially easier and to analyse and more efficient to work with. Within dimensionality reduction, non-linear dimensionality reduction (also known as manifold learning) is the class of techniques that are capable of producing more complex reduction in dimensionality than by the sole use of linear combinations.

Autoencoders are a powerful class of unsupervised learning algorithms for non-linear dimensionality reduction [9]. Autoencoders typically follow a dual architecture, containing an encoder and a decoder. The encoder maps the original data to a low-dimension embedding space (sometimes referred to as a bottleneck in this context), while the decoder attempts to reconstruct the original input from the embedding, thereby evaluating the quality of the embedding space produced by the encoder.

1.1 Problem Statement

Non-linear dimensionality reduction techniques can be divided into two paradigms: mapping and non-mapping. Mapping techniques are able to produce an interpretable mapping of instances from the original space to the low-dimension space, while non-mapping techniques simply produce a lower dimension embedding of the data. Access to a functional mapping is useful for two reasons. First, it allows unseen data instances to be placed in the embedding space without the need to re-run the dimensionality reduction algorithm. Secondly, it allows for better understanding and explainability of how the original features are used to form the embedding, and can be used to generate additional insight into the original data and the importance of the individual features.

Along the non-linear dimensionality reduction paradigm, the trained encoder of an autoencoder can be thought of as a ”functional mapping”. The conventional approach using artificial neural networks (ANN) means that while the functional mapping is re-useable, interpreting and explaining *how* the original features are transformed to the embedding by the encoder is made difficult by the opaque structure of ANNs [29].

1.2 Motivation

For the purposes of dimensionality reduction, while the trained decoder can be informative, it is the encoder where interpretability is primarily valuable. Genetic programming (GP) is an evolutionary computation (EC) technique where computer programs are evolved over generations [31]. GP has recently been demonstrated to be a capable non-linear dimensionality reduction technique which produces functional mappings [20, 21]. Some of these approaches have used a multi-tree GP representation with an ad-hoc method for evaluating embedding quality, and other work has looked into GP specifically for autoencoding [34, 24].

Existing research into GP for autoencoding has attempted to replicate the encoder-decoder structure using GP, but this is difficult as the EC approach of applying stochastic changes to the encoder or decoder separately can have significant side-effects on the performance of the other [24]. Other attempts have forgone the architecture to focus on replicating the autoencoder behaviour directly, although using representations that are complex and difficult to interpret [34]. We suggest that due to demonstrated suitability of multi-tree GP to non-linear dimensionality reduction in general, that by replacing *only* the encoder with a multi-tree GP individual whilst retaining the ANN decoder, the value of GP interpretability can be harnessed while avoiding the problems of evolving both the encoder and decoder simultaneously.

1.3 Goals

In this work, we propose an approach to use a GP encoder for autoencoding (**GPE-AE**). We will:

- Explore the background of evolutionary computation, genetic programming, dimensionality reduction and autoencoders, and evaluate existing work relevant to GP for non-linear dimensionality reduction and autoencoding;
- Propose a new autoencoding method that replaces the encoder with a multi-tree GP individual while retaining the ANN decoder;
- Evaluate how our proposed method compares to a conventional ANN autoencoder and another GP for non-linear dimensionality reduction method;
- Analyse some 2D visualisations produced by our method and the comparison methods; and
- Analyse selected GP trees produced by our method to demonstrate the value of interpretability.

Chapter 2

Literature Review

2.1 Chapter Overview

In this chapter, we review the existing relevant literature to genetic programming autoencoders. We begin by outlining the areas of machine learning and unsupervised learning in general. We then introduce genetic programming (GP), first by explaining evolutionary computation (EC) and genetic algorithms (GA), then by outlining genetic programming and the typical tree-based representation. Multi-output approaches to GP are also explored.

Dimensionality reduction (DR) and the overlapping paradigm of feature manipulation (FM) are introduced, with focus given to non-linear dimensionality reduction (manifold learning). We provide a brief overview of simple neural networks and, and introduce standard neural network based autoencoders.

We explore related work in two categories: EC for dimensionality reduction, and EC for autoencoding. We summarise that an approach replacing only the encoder of a conventional neural network based autoencoder for the purposes of dimensionality reduction represents an unexplored direction that can address some of the shortcomings of existing GP for autoencoding approaches.

2.2 Machine Learning

Machine learning (ML) is the study of computer programs which learn from experience [27]. Generally ML is the use of algorithms that can be used to construct models through interaction with data that allow us to perform inference and explain the structure of the data.

In discussing machine learning and the data it uses, there are a range of terms that are often interchanged. As such, it is important to provide some early clarity to some of the terms used. We describe individual observations of data as *instances*, the attributes of data as *features*, and a whole collection of data as a *dataset*.

A simple example of a machine learning algorithm is k-nearest neighbors (KNN). KNN is an algorithm for *classification*: assigning labels to unseen data using a training dataset of labeled data. Due to its reliance on labeled data, classification is a form of *supervised learning*. KNN doesn't require training as such, but instead classifies new instances as the most common label of the k closest instances in the training data to the new instance. KNN demonstrates the core concept of classification, and the use of machine learning to identify and exploit patterns in data.

2.3 Unsupervised Learning

Within machine learning, unsupervised learning algorithms are those which learn without the used of data labels [50]. Generally speaking, unsupervised learning is for finding structure in data, *without* the need for that structure to be known in advance.

Clustering, or cluster analysis, is a typical unsupervised learning technique [39]. A clustering algorithm seeks to identify clusters of instances such that instances within the same cluster are as similar as possible, while instances in separate clusters are maximally dissimilar. Unlike classification, there are is no labeled training data with which to build a model from to assign instances to clusters.

A key aspect of unsupervised learning is that often objective "ground truth" measures of model quality are unavailable. Instead, unsupervised techniques often rely on more subjective measures, that can value different components of a solution. For clustering, for example, there are measures of similarity within clusters, measures of dissimilarity between different clusters, and measures that combine different aspects in different ways [39].

2.4 Evolutionary Computation

Evolutionary computation (EC) techniques are a family of machine learning algorithms inspired by biological evolution [10]. As with biological evolution, EC techniques emphasise a trial and error approach to optimisation, generally employing the use of a population of candidate solutions. An important aspect of EC techniques is the balance between exploration and exploitation. Stochastic trial and error is used to explore the solution space for new promising candidate solutions, while high performing found solutions are focused on to exploit promising areas.

2.4.1 Evolutionary Algorithms

An important subset of EC is evolutionary algorithms (EA) [48]. EAs are stochastic population-based search algorithms. They make use of evolutionary operators inspired by biological evolution, such as mutation and reproduction (crossover). A key strength of EAs is the low requirement for domain knowledge of the problem they are applied to.

The general structure of an EA is presented in Fig. 2.1. The steps involved are:

1. *Initial Population*: Generally initialised with random solutions.
2. *Evaluation*: Each individual of the population is evaluated using objective functions.
3. *Fitness Assignment*: Each individual is assigned a fitness based on performance.
4. *Selection*: A selection function is used to select individuals from the population based on their fitnesses.
5. *Variation*: The selected individuals are varied to produce an offspring population, replacing the original population.

This process continues until some stopping criteria is met, for example, a set number of generations.

There are two common variation operators used in EAs:

- *Crossover (Reproduction)*: The recombination of parent individuals to create child solutions containing elements of both their parents.

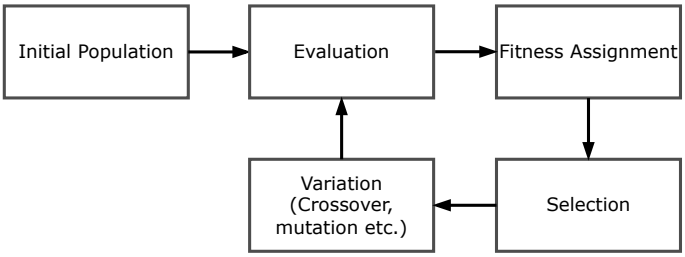


Figure 2.1: The general structure of a simple Evolutionary Algorithm

- *Mutation*: Stochastic changes made to individuals to introduce new elements to the overall population.

At the core of EA is the *fitness* of a solution. A *fitness function* is used to assign fitnesses to individuals. Due to EC using a trial-and-error approach to optimisation, there are very few constraints on the form of the fitness function. In addition, EA algorithms can optimise multiple objective functions simultaneously, which may be competing, using EA multi-objective algorithms such as NSGA-II and SPEA-II [41].

Important to EAs is the *selection strategy*. The selection strategy is the method used to select high-performing individuals from the population to use for variation to generate the population for the next generation. Selection strategies generally have to balance exploitation and exploration. Simply selecting the highest performing k individuals to carry through would mean that potentially good components of weaker individuals may be lost, lessening the explorative aspect of the EA. However, high performing individuals should still be favoured for selection in some way to ensure the increasing fitness of the population. t-tournament selection is a typical selection strategy which balances exploitation and exploration. With t-tournament selection, t individuals are randomly sampled from the population, and then the individual of the t with the highest fitness is selected. To ensure that the highest performing individuals are not lost by the selection process, elitism can be employed, which guarantees a set number of high performing individuals are carried over to the next generation in place independently of the selection strategy.

A well known subset of evolutionary algorithms are genetic algorithms (GA) [33]. In GA, candidate solutions are represented as fixed-length genetic individuals. Each individual has a genotype, which can be expressed in varying ways depending on the problem. A common approach is as a string of 0s and 1s, which we consider for the remainder of this section to demonstrate a simple implementation of a EA.

Crossover is performed between two bit-string individuals with fixed-length n:

1. Select a random point uniformly from $i \in 0...n$.
2. Split each parent A and B into two sub-strings, $[0, i-1]$ and $[i, n]$.
3. Recombine the first sub-string of parent A with the second sub-string of parent B, and vice versa.

Mutation can be performed simply by selecting a random index of an individual and ”flipping” it’s value.

The bit-string approach to GA is the most simple, but there are many other approaches including the use of integers or continuous values. Likewise, the mutation and crossover presented here is just a simple example, there are many other domain specific approaches.

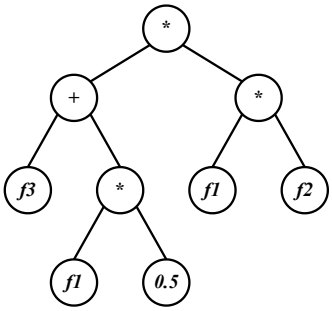


Figure 2.2: An example of a tree-based GP individual.

2.4.2 Genetic Programming

Genetic programming (GP) is a EA approach with some similarities to GA. However, instead of individuals representing fixed-length candidate solutions to a problem, genetic programming individuals are variable length computer programs [31].

The most common representation of GP programs is as trees. In a tree-based individual, the leaves of the tree are a combination inputs for the problem and random constants, while the internal nodes are functions which take their children as inputs. As such, a tree is evaluated bottom up, with the root representing the output of the program.

An example of a tree-based GP individual is provided in Fig. 2.2. This individual represents the function $(f3 + (f1 * 0.5))) * (f1 * f2)$. The terminals (leaves) are three of the problem inputs $f1$, $f2$ and $f3$; and a single constant 0.5. The tree is evaluated bottom up, the top node producing the final output.

For a GP run, the set of functions that can be used is referred to as the *function set*. In addition, the nodes that are able to be leaves of a tree is known as the *terminal set*. Typically the terminal set is the inputs to the problem GP is being used to solve, as well as ephemeral random constants. In standard GP it is required that terminal and function sets conform to *closure*, meaning that all of the functions can use *any* of the terminals or outputs of the other functions as inputs. There are variations, such as Strongly-Typed GP, where functions that use different data types can be used [28].

GP has several important hyper-parameters that must be set, which are presented in Table 2.1.

Parameter	Description
Generations	No. generations to run the evolution.
Population Size	No. of individuals in population.
Min. Tree Depth	Minimum depth of individuals.
Max. Tree Depth	Maximum depth of individuals.
Initialisation	Initialisation strategy used.
Selection	Selection strategy to use.
Mutation Prob.	Probability of applying mutation to selected individual.
Crossover Prob.	Probability of applying crossover to selected individual.
Elitism	No. of best performing individuals to carry over unmodified.

Table 2.1: GP parameters and their descriptions.

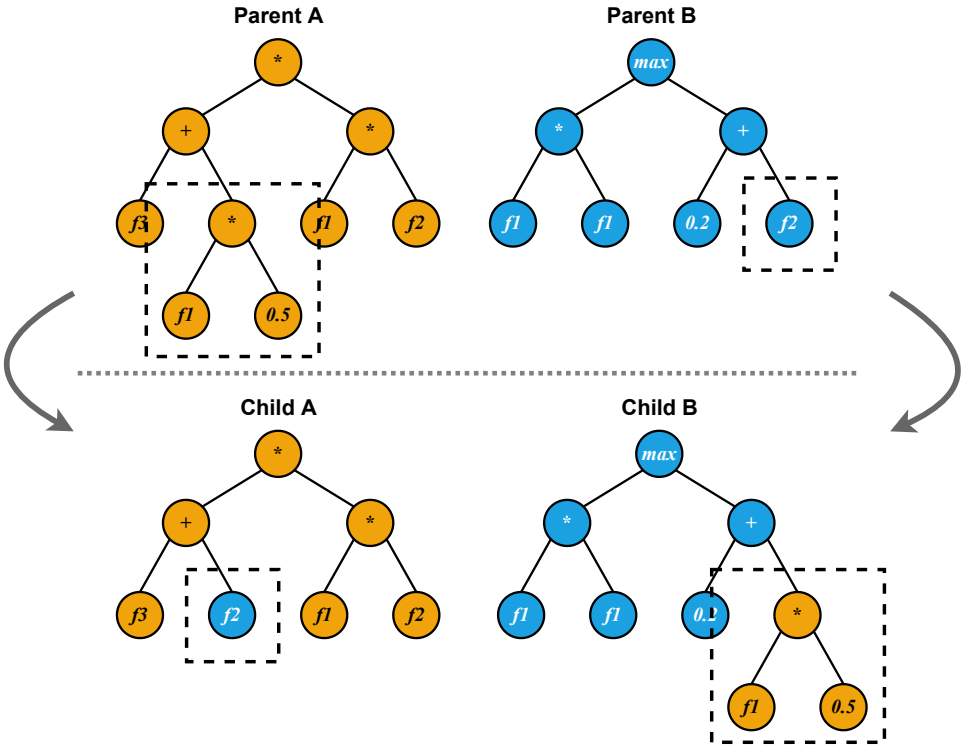


Figure 2.3: An example of crossover performed on two tree-based GP individuals.

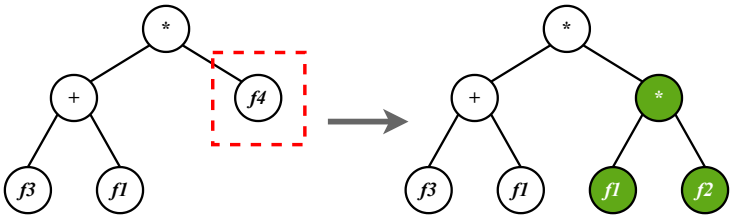


Figure 2.4: An example of GP mutation. A random sub-tree is replaced with a newly generated one.

Variation

As with GA, crossover and mutation operators are a key component of GP. The typical crossover method for GP is *subtree crossover*. An example of this can be seen in Fig. 2.3. A random subtree is selected from each of the parents, and then two new children are created by swapping the subtrees. In the example, we can see an example of a single terminal node from Parent B being swapped with a 3-node subtree from Parent A.

Mutation is generally performed by selecting a random subtree of an individual, and replacing it with a new randomly generated subtree. An example of standard GP mutation is presented in Fig. 2.4. We can see that the randomly selected sub-tree, here a single node $f4$, is replaced with a newly generated sub-tree, representing the function $f1 * f2$.

Initialisation

GP individuals are initialised top down recursively. At each node, a child node is selected randomly for each input the node requires. Each child is then itself initialised, up to some maximum tree depth d where only terminals will be selected, terminating the growth of the

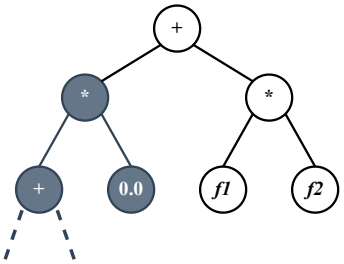


Figure 2.5: An example of an intron, (non-expressed genetic code). Here, the grey sub-tree (not shown in full) has no impact on the output of the program.

tree. There are three common strategies for randomly initialising a GP population, which vary in how they choose the children nodes:

- *Grow*: Until d is reached, children can selected randomly from either the terminal set *or* the function set. This can allow for trees of varying sizes, as selecting a terminal terminates any further growth of the sub-tree.
- *Full*: Until d is reached, only function nodes will be selected. This ensures individuals are full trees of depth d.
- *Ramped half-and-half*: To encourage more diversity in the initial population, half of the population is initialised with grow, and half with full.

Bloat and Introns

Due to the dynamic size of GP individuals, trees can in theory grow to be arbitrarily large over the evolution, even though the extra nodes may provided negligible fitness increases. This is a problem know as *bloat*. Setting a maximum tree-depth is a common and simple bloat-control technique. Other techniques such as parsimony pressure seek to penalise larger trees, thereby encouraging smaller solutions [43].

Sometimes a GP individual can contain sub-trees which are unexpressed, also know as *introns* [49]. By unexpressed, we mean the output of the sub-tree has no impact on the final output of the whole tree. An example of an intron is presented in Fig. 2.5. Here, the left-most sub-tree of the root has no effect on the output of the program, as it always evaluates to 0. Despite the additional structure, this GP individual represents the simple function $f1 * f2$.

Even though these introns could be thought of as contributing to bloat, there is evidence that allowing for *some* introns can have positive effects on GP evolution [49]. This is due to the fact that even though the code is unexpressed in the current individual, they can impact the crossover and mutation of individuals.

2.4.3 Multi-output GP

In traditional tree-based GP, individuals produce a single output. However, in many domains multiple outputs are required to form a complete solution. In some of these domains, problems can be decomposed into independent single output problems. In these cases, it can be sufficient to perform multiple independent GP runs, each producing an individual to solve the sub-problem. Another approach is co-operative co-evolution, where separate populations are maintained during a single GP run, each assigned with finding an individual for a sub-problem [32]. However, it can be beneficial to be able to represent multi-output programs in a single individual. There have been various adaptations and new representations proposed for individuals to produce multiple outputs.

Tree Based

Tree-based GP has been adapted in two ways to represent multi-output individuals: the use of multiple trees in individuals, and allowing for multiple output nodes in a single tree. Multi-tree approaches are an intuitive extension, simply comprising individuals of as many trees as the problem requires outputs, with all trees sharing the same terminal and function set. For variation, standard tree-based GP operators can be used, although some method is required to determine which of the individuals internal trees are used.

Multiple interactive outputs in a single tree (MIOST) is an an adaptation to single-tree individuals to allow for multiple outputs [23]. MIOST combines two concepts: multiple output nodes in a single tree, and pointers within the tree. The special output nodes are unique to each output of the problem, and a tree is only valid if *all* the output nodes are present. To avoid problems with variation affecting the number of output nodes, mutation is constrained to not delete trees containing an output node, and all nodes of a tree are classified at initialisation as to whether crossover performed here will lead to an invalid individual or not. The pointers within individuals allow for a subtrees to be "deactivated", with their output replaced by the sub-tree they point to, allowing for better sharing of calculations within a tree. While this approach may be valuable for problems with highly related outputs, in problems where separability of the outputs is valuable MIOST may perform poorly.

Alternative Representations

Beyond tree-based multiple-output representations, there are approaches that abandon the tree structure entirely.

Cartesian Genetic Programming (CGP) is a graph-based GP representation [26]. A CGP program is a directed a-cyclical graph, represented as two-dimensional grid of nodes. These nodes can be terminals or functions, as in tree-based GP. A node in a column can take its input from any node in any previous column, but not from a successive column. The nodes in the final layer represent the outputs of the solution. In some ways, CGP is similar to a neural network (discussed in Section 2.6). However, the ability to "look-back" beyond just the preceding layer is behaviour not present in a conventional neural network.

Single Node Genetic Programming (SNGP) is another graph-based representation for multiple outputs [11]. In SNGP, each individual is a single node, either a terminal or a function, which maintains a set of predecessor nodes (inputs) and successor nodes (outputs). When an individual is evaluated, it is used as the root to evaluate the program. As such, each individual essentially represents a different arrangement of the population, each with a different output. A single variation operator is used for SNGP, where one of a nodes successors is changed. If the performance of the node increases, the change is kept, else it is reversed.

Linear Genetic Programming (LGP) is a class of GP representations in which individuals are linear sequences of instructions, as opposed to functional expressions [6]. It is important to note that this where the name is derived from: it does not mean that LGP is suitable only for linearly separable problems. The set of instructions takes inputs from registers, performs the instruction, then stores the outputs in registers. Which registers, instructions and the order of these is what is encoded by a LGP individual. Registers can be designated as input registers, where the original input data is first stored, output registers, where the outputs to the program are stored (which can be as many as the problem requires), and the intermediary registers, which can be used to store intermediate values.

2.5 Dimensionality Reduction

The dimensionality of a dataset is the number of dimensions (columns) that comprise a single instance (row). Dimensions are also often referred to as features, or attributes. As the dimensionality of data increases, the more difficult it can be to work due to the increasing sparsity, an example of the *curse of dimensionality* [7]. This gives motivation for techniques to reduce the dimensionality of data to make it more interpretable or visualiseable, or as a pre-processing step make it more efficient to run other machine learning algorithms over the data. We refer to a lower dimensional representation of a dataset as an *embedding*.

Dimensionality reduction is an unsupervised machine learning task: there are no labels to target, and no objective way to measure the quality of an embedding. Some measures of quality include measuring the variance between embedding dimensions, or comparing instance neighborhoods in the original space and the low dimensional embedding. Additionally, there is often no way to determine how many dimensions the data should be reduced to. Some techniques require the number to be preset by the user, while others are capable of dynamic embedding sizes.

Principal Component Analysis (PCA) is perhaps the most well known dimensionality reduction technique [12]. PCA works by finding *components*, which are linear combinations of the original features. PCA finds the coefficients for the components one at a time, such that the variance of the component is maximised while being orthogonal to any preceding components. Maximising the variance is effectively maximising the information captured by the component, while the orthogonality constraint minimises redundancy between components. While PCA requires the number of components to be set, more can be calculated without the need to re-calculate the earlier components.

2.5.1 Feature Manipulation

Feature manipulation techniques are a family of techniques concerning the manipulation of features. These can be broadly divided into two distinct categories: Feature Selection (FS) and Feature Construction (FC) [42]. FS techniques work simply be selecting a subset of the original features, while FC techniques construct *new* features from the original features. FC techniques represent the simplest approach to dimensionality reduction, in simply removing redundant or irrelevant features. FC approaches comprise any technique that uses a *transformation* of the original space. Along the FM paradigm, PCA can be though of as a feature construction technique. In practice, both can be valuable.

FS and FC techniques are often classified into three categories [45]:

- **Wrapper**: FM techniques which evaluate candidate feature sets by directly using a ML algorithm;
- **Filter**: FM techniques which use some external measure for feature set quality independent of a specific ML algorithm, such as variance (as in PCA); and
- **Embedded**: ML algorithms which implicitly perform FM, such as in decision tree learning.

Generally speaking there are tradeoffs between these paradigms. Wrapper methods are generally better at finding high quality embeddings for the task for which they are required. However, as each evaluation of an embedding requires the full training and evaluation of a machine learning model, it can have significant run-time costs. There is also a problem of specialisation: an embedding found using certain classification algorithm may not be

well-suited to an alternative classification algorithm it was not trained on, or some other alternative use. Wrapper methods are also somewhat removed from traditional unsupervised dimensionality reduction if the wrapper algorithm used is supervised, as this requires labeled data.

Filter methods are generally faster to train than wrapper methods, however they are likely to be outperformed by wrapper methods on the ML algorithm used for training.

Embedded methods exist between wrapper and filter methods in run-time and performance, as they are somewhat of a hybrid of both paradigms.

2.5.2 Non-linear Dimensionality Reduction

While techniques such as PCA can be sufficient for producing high-quality embeddings, often the underlying structure of a dataset is too complex to be captured by linear combinations and transformations. For these datasets, Non-Linear Dimensionality Reduction (NLDR) techniques are required. These are also sometimes referred to as *manifold learning* techniques [4].

NLDR techniques can be divided into two classes: mapping and non-mapping. Mapping techniques are those which produce the data in the low-dimension space, as well as a functional mapping for instances from the high-dimension space. Non-mapping techniques on the other hand provided only the low-dimension embedding.

Having access to a mapping has a few key advantages. Firstly, it allows for better interpretation of how the dimensionality reduction has been achieved by being able to identify which of the original features are important to the found embedding. Secondly, it allows for new instances of the data to be placed in the low-dimension space without the need to re-run the DR algorithm again.

A canonical example of a high performing NLDR algorithm is t-distributed Stochastic Neighborhood Embedding (t-SNE) [8]. t-SNE works by constructing a probability distribution over pairs of instances in the original feature space, such that instances close together have a high probability. Then, a second probability distribution is constructed over the instances in the desired low-dimensional space. Then t-SNE minimises the Kullback-Liebler (KL) divergence between the two distributions with respect to the locations of the instances in the low dimensional space, resulting in the final embedding.

The more recent state-of-the-art Uniform Manifold Approximation and Projection (UMAP) [25] follows a similar process to t-SNE. However, instead of using probability distributions and minimising then KL divergence, UMAP uses fuzzy graph representations of the data in the high dimensional and low dimensional space. In the UMAP fuzzy graphs, edges between instances represent probabilities the instances are in the same k neighborhood. The embedding is found by minimising the cross-entropy between the fuzzy graph representations.

Both t-SNE and UMAP are non-mapping. There have been parametric variations proposed for both that use neural networks to allow for reuse-able mappings, although they are still extremely complex and difficult to interpret [47, 36].

2.6 Artificial Neural Networks

Artificial neural networks (ANNs) are machine learning systems modeled on the neural networks found in biological brains [40]. They are typically composed of nodes (neurons) which are connected by edges (connections). The most common NN architecure is a the directed a-cyclical feedforward network, where data is ”fed” in at one end, with the pre-

dictions of the input data being output at the other end. These can be used for a range of machine learning tasks, such as classification and regression [2].

While neural networks are a popular choice for machine learning tasks due to their high performance, the large number of connections and shared computations between inputs and outputs makes them something of a black-box: while they may produce a good output, it can be difficult to tell *how* that output relates to the specific values of the input [29].

2.6.1 Single Layer Perceptron

The simplest feedforward ANN is the single layer perceptron (SLP). In a SLP, there is a single neuron y, connected to input data x. There is a connection between each dimensions of the input x_i and the output y with a weight $w_{i,j}$. The value of output node j can be calculated as:

$$y = \sum_{i=1}^{x} w_i x_i + b \tag{2.1}$$

where b is the bias, a constant independent of the input value. How the values of w_i and b are learned is explained in section 2.6.3.

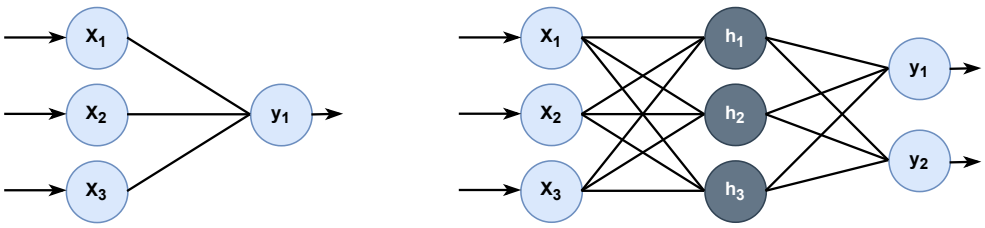
The value of the output node often has an *activation function* applied to produce an output in the desired form. For example, a threshold function may be used to produce either a 0 or 1 for the purposes of classification, or the logistic function may be used to produce a continuous function for regression.

An example of a SLP for a three input problem is presented in Fig. 2.6a

2.6.2 Multi-Layer Perceptron

SLPs can only find linearly separable patterns. As such, a more complex architecture is required for more complex problems.

Multi-layer perceptrons (MLPs) are a natural extension to SLPs. MLPs have *hidden layers* between the input and output layer, facilitating more connections and thus more involved computations. Each hidden layer is fully connected: there is a connection between each node of the layer to each node in the next layer. As with the SLP, there is a learned weight associated with each connection, and a learned bias associated with each node. The neurons in the hidden layer also make use of activation functions, such as the Rectified Linear Unit (ReLU) to add the potential for non-linear computations to the model.



(a) Single Layer Perceptron (b) Multi Layer Perceptron

Figure 2.6: Examples of both single and multi layer perceptrons. Both have three inputs. The SLP has a single output, while the MLP has two. The MLP has a single hidden layer with three neurons.

2.6.3 Training Neural Networks

The most common method of training neural networks (learning the parameters) is back-propagation. In a feed-forward network, labeled training data is fed-forward, and an objective *loss function* is used to calculate the error between the output and the target labels. Then, the weights are updated from the end of the network to the front, optimising the weights with respect to the loss value of the input-output pairs. There are various popular methods for performing this optimisation, such as gradient descent, stochastic gradient descent, and Adam [14].

The feed-forward instances, back-propogate errors loop continues until a termination criteria is met, with each pass through known as an *epoch*. The *learning rate* hyperparameter in the range $[0, 1]$ is used to determine how much the weights change between epochs, with a lower learning rate leading to smaller changes, requiring potentially more epochs.

Over-fitting is a common problem in neural networks [38]. Overfitting occurs when the network is trained to match the training data too closely, failing to capture the generalities of the true data. To avoid this, validation data with early stopping can be used. At each epoch, a validation dataset (that is **not** used for learning the weights) is evaluated on the learned model. If validation performance begins to drop between epochs, it is a sign that the model has started over-fitting to the training data, and the training can cease.

2.6.4 Other Types of Neural Networks

Beyond the standard MLPs, there are a range of other neural networks. Deep neural networks (DNN), which are commonly associated with deep learning, are a special case of neural network that have a large number of hidden layers and neurons [17]. Convolutional neural networks (CNN) make use of a special *convolution layers* to extract features from data with some sort of positional structure, such as images or audio [1].

2.7 Autoencoders

Autoencoders (AEs) are structures used to learn representations of data [9]. Often the task that autoencoders are used for is referred to as *representation learning*. In practice, the learned representation is almost always desired to be of lower dimensionality than that of the input, making autoencoding a dimensionality reduction method in effect. Conventionally, AEs are implemented with neural networks.

There are two main parts to autoencoders: the *encoder* and the *decoder*. The encoder is trained to map the inputs to the lower-dimension representation space, while the decoder is trained to reconstruct the original inputs, using the encoded outputs. As autoencoding does not require the use of labels, it is an unsupervised learning technique, although due to it's use of the input data as labels it has also been referred to as *self-supervised learning*. The lower dimension space is often called the latent space or the *bottleneck*, which is equivalent to the concept of an embedding along the dimensionality reduction paradigm. By forcing the data through this space, it ensures only the most important information is stored efficiently.

In terms of NLDR, the encoder can be thought of as a functional mapping. However, due to conventional AEs using NNs for encoding and decoding, even though this mapping is resuseable, it is unlikely to interpretable. Once trained, the decoder can be effectively discarded, as for dimensionality reduction its purpose is only to validate and guide the training of the encoder.

In practice, a simple AE is essentially just an MLP, with a "special" hidden layer with fewer nodes than input features. Additionally, rather than using a loss function on the pre-

dicted labels, the loss function is the *reconstruction error*. The reconstruction error used is generally mean-square error (MSE) across the whole training set:

$$MSE = \frac{\sum_{i=1}^{n}(D(E(x_i)) - x_i)^2}{n} \tag{2.2}$$

Where n is the number of instances in the training set, x_i is instance i, E is the encoder and D is the decoder, represented as functions.

An example of a a simple NN based AE is presented in Fig. 2.7. In this example, the AE is learning a representation b of dimensionality 2 for data x with a dimensionality 5. Both the encoder and decoder have a single hidden layer with 5 neurons. The feedforward nature of the network means that each layer only has access to the outputs of the preceding layer. This means that layer h_2 of the decoder can *only* use information that is able to be captured in layer b, and the output layer must make the prediction of the input x' with only what is passed from h_2. Therefore, if a low-error reconstruction is able to be produced by the neural network, the representation at layer b must contain the important information about x.

2.7.1 Variational Autoencoders

There are several variations of the standard autoencoder. Perhaps the most well known is variational autoencoder (VAE) [15]. Instead of learning a direct latent representation of the input, the encoder learns a latent *distribution*, usually a multi-variate Gaussian. Instead of the values at latent neurons being used as direct encodings of the input, they are used as parameters for the distributions. These distributions can then be sampled from by the decoder, with the sampled values used to reconstruct the input. The loss function used by a VAE combines the reconstruction error with some sort of measure of distance between distributions to constrain the latent dimension towards a certain shape.

The strength of VAEs is that once trained, the encoder can be discarded, and the learned latent distribution sample from to generate *new* data using the decoder. VAEs implemented with CNNs have been used to generate new images, such as faces.

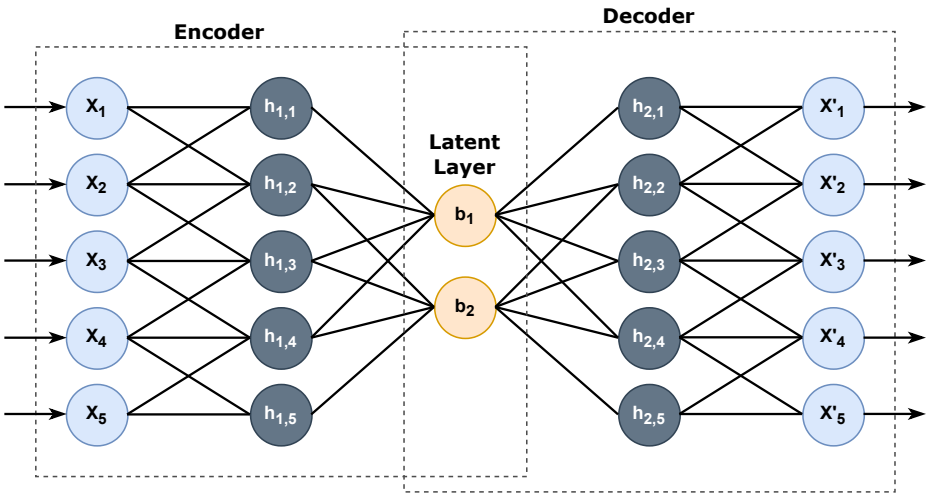


Figure 2.7: An example of a basic neural network autoencoder.

2.8 Related Work

2.8.1 Evolutionary Computation for Dimensionality Reduction

Various EC techniques have been applied to DR. The most straightforward DR task, feature selection, has been approached by a range of EC techniques, such as the bit-string GA we described in 2.4.1 [18], particle swarm optimisation (PSO) [51], and ant colony optimisation (ACO) [13].

For the more complex problem of feature construction, the programatic structure of GP is an obvious candidate. By using the original input features as the GP terminal set, and setting an appropriate fitness function, GP can learn high-performing combinations of features in an explainable way with little constraint on the form of the learned functions. In fact, due to GPs functional combination of original features, any machine learning task that GP is applied to, such as classification, results in an inherent selection of features. This makes is comparable to a *embedded* technique.

Along the feature construction paradigm, DR is generally assumed to be unsupervised, or *filter* based. Beyond this, there has been other work that has looking into using GP for wrapper-based FC, where machine learning algorithms are used to directly evaluate the performance of GP individuals. The multi-tree approach has been applied to wrapper-based FC for classification [46] and clustering [19]. In the case of clustering, this is an example of unsupervised wrapper based FC, as clustering itself is an unsupervised technique.

Genetic programming has been proposed as a potential approach to learn functional mappings for non-linear dimensionality reduction, as opposed to the canonical non-mapping techniques such as t-SNE and UMAP. The functional structure of GP trees lends itself to produce not just mappings which are re-useable, but also ones that are interpretable.

GP-MaL: Genetic Programming for Manifold Learning

Genetic Programming for Manifold Learning (GP-MaL) is one such proposed GP for NLDR technique [20]. GP-MaL uses a multi tree representation, where each tree represents a functional mapping of the original high-dimension feature space into a single embedding dimension. As such, w tree individuals are used to represent w dimensions of the embedding. In the original paper, the number of trees is fixed, although further work has looked into the ability to change the number of trees over the evolution using multi-objective optimisation (GP-MaL-MO) [21]. GP-MaL makes use of conventional GP functions, as well as the non-linear ReLU and sigmoid functions to allow for non-linear mappings to be found.

The fitness function used by GP-MaL is based on the preservation of orderings of neighbors from the original feature space to the embedding space. A neighbor ordering for an instance is a list of every instance, in order of closest to furthest. GP-MaL measures how close these orderings are in the high and low dimensional spaces for each instance.

The first part of the GP-MaL fitness is a formula for evaluating the similarity between two neighborhoods, N and N'. This is a sum of the differences between the positions of each neighbor, a, in the two ordered neighborhoods, with a slight Gaussian penalty applied through an agreement function:

$$Similarity(N, N') = \sum_{a \in N} Agreement(|Pos(a, N) - Pos(a, N')|) \tag{2.3}$$

Where $Pos(a, X)$ gives the index of a in an ordering X, and the agreement function gives higher values for smaller deviations, thereby allowing small deviations without significant penalty.

Finally, the final GP-MaL fitness is calculated as the sum of every instances neighborhood similarity across the whole dataset X with n instances, with a high dimensional neighborhood N and a low dimensional neighborhood N':

$$fitness = \frac{1}{n^2} \sum_{I \in X} Similarity(N_I, N'_I) \tag{2.4}$$

The GP-MaL fitness is fairly ad-hoc. As is common in unsupervised learning, it optimises subjective measures of embedding quality, in this case the single measure of neighborhood preservation. Other work has extended on the GP-MaL approach using the UMAP cost function and UMAP embeddings directly as a measure of fitness [37].

ManiGP

ManiGP is another approach to GP for explicit manifold learning [30]. ManiGP uses the same multi-tree GP representation as GP-MaL. ManiGP fitness evaluation is as follows:

1. A clustering algorithm is run on the instances in the embedding space.
2. All instances is each separate cluster are assigned to a single class of the data based on similarity.
3. A balanced classification accuracy is calculated using the found class labels, which is used as the fitness.

ManiGP's use of class labels makes it a wrapper-based technique, as opposed to the filter-based GP-MaL. This makes it insufficient for true unsupervised learning, but perhaps valuable when a lower dimensional embedding of labeled data is required, for example for 2D visualisation.

2.8.2 Evolutionary Computation for Autoencoding

EC techinques have been applied to autoencoding primarily in two ways: using EC to evolve structures for NN-based AEs, or using EC directly for autoencoding. We explore the first only briefly, as any interpretability gained from the use of EC in designing NNs helps for explainibility of only the NN architecture, but not for the low dimensional embeddings directly.

The Evolutionary Autoencoder (EvoAE) is one such approach to evolving NN structures [16]. An individual in EvoAE is a set of nodes, each representing a neuron in the single hidden layer of an simple NN-AE. Additionally, each node contains weights for each input value. Crossover is done simply by swapping nodes (including their learned weights) between parents. Mutation allows for nodes to be removed or added, thus making this an approach which allows for a dynamic embedding dimensionality. After variation in each generation, the new individuals weights are updated using gradient descent, initialised with the weights set by the evolution.

Sereno et al. propose a technique for evolving NN AEs which allows for a dynamic number of layers and embedding dimensionalities [3]. In this work, the representation is a dynamic length array of integers, with the number of entries representing the number of hidden layers, and the values at each entry representing the number of neurons in the hidden layer. Rather than having a fixed layer that is always used as the bottleneck, the smallest layer is selected for each individual, thereby making the separation between the encoder and the decoder dynamic. The fitness evaluation of each AE involves three measures: the number of layers in the decoder, the number of neurons in the bottleneck, and the

classification obtained using the embedding. The use of classification and data labels makes this a supervised approach to dimensionality reduction.

There is a small amount of existing work that looks at applying specifically GP to autoencoding. We now review three significant contributions.

Genetic Programming Autoencoder

Genetic Programming Autoencoder (GPAE) is a proposed technique that replaces the autoencoder entirely with a GP representation [24]. The authors motivate GPAE as an autoencoder with the capability for more interpretable encoder and decoder functions, and the potential to harness GPs ability to optimise multiple non-differentiable objectives.

GPAE uses a linear GP representation, due to its suitability for multi-output problems. Each individual is comprised of two linear GP programs that represent an encoder and a decoder, with a bottleneck between them. Linear GP is selected for both the encoder and decoder for simplicity, although it is acknowledged that any multi-output representation could be sufficient, and that the representation of the encoder and decoder do not need to necessarily match. The multi-tree approach is ruled out due to the inability to share calculations between trees. Work such as GP-MaL demonstrates that multi-tree GP is capable of finding non-linear functional mappings for dimensionality reduction, suggesting that it may be suitable as an encoder despite this. In fact, given that we want minimal redundancy (shared information) between the embedding dimensions, it can be argued that the ability to share calculations across each dimensions functional mapping only allows for more information sharing.

Instead of a traditional population-search approach, GPAE uses a single individual which is optimised using step-counting hill-climbing. As such, there is not crossover, just a mutation operator. Mutation is performed randomly on either the encoder or decoder program. The hill-climbing algorithm allows for some decrease in performance resulting from a mutation, allowing for local-optima to be escaped.

Structurally Layered Genetic Programming

Structurally Layered Genetic Programming (SLGP) is another proposed approach that replaces the whole autoencoder with GP [34]. SLGP uses a dual forest representation, such that w trees are used to produce the low-dimension representation, and v trees are used to reproduce the original input.

Representing the decoder as a forest of trees can be a challenging task for high-dimensional data, where a tree is required for each original feature. As such, SLGP decomposes the problem to smaller, independent GP runs. That is, for some selected constant N, N GP runs are performed, each on learning an embedding of v/N original features. Once all runs are complete, the partial individuals can be combined to create a complete individual with the encoding and decoding forest for the full v original features, with an embedding size of wN.

There are obvious limitations to this approach. Primarily, the dimensionality reduction mapping can only take into account combinations of original features which are in the same subset. However, this is hard to get around with the tree-based GP decoder due to large number of decoding trees potentially needed for each dataset, one for each original feature. Trying to evolve a large number of trees simultaneously represents an obvious problem. We argue that the drawbacks of having to learn a decoding tree for each original dimension ultimately outweigh the benefits of being able to interpret the decoder, which essentially just serves to evaluate the encoding trees.

Genetic Programming for Feature Learning

Genetic Programming for Feature Learning (GPFL) is an autoencoder-like approach to feature learning using GP [35]. While not strictly dimensionality reduction, feature learning is a tangential task that involves learning informative representations of data for use in a ML algorithm. Rather than modelling GPFL on the neural network architecture, GPFL mimics the reconstructive behaviour of autoencoders directly without relying on the encoder-decoder model. GPFL is presented for 2D image feature learning, although it is suggested that it could be extended to more general dimensionality reduction problems.

GPFL uses a tree based representation. It uses multiple GP trees, but in a significantly different way than the more simple multi-tree approach. Trees are learned one at a time, for either a set amount of iterations or until some stopping criteria is met. *Dynamic targeting* is used, such that each tree is trained to focused on the areas the previous trees performed poorly on. Each tree uses pixel coordinates as inputs, producing a single output value, representing a pixel value in the reconstruction. The final individual calculates the value as a linear combination of its trees, referred to as *partial models*. The linear scaling coefficients for the combination are derived from the input image, and as such essentially represent the "encoding".

One significant drawback of GPFL is it's relatively indirect model structure. A major potential key benefit to using GP for autoencoding is the ability to produce a clear functional mapping to the low dimension space, which GPFL does not provide.

Summary of GP for AE

One notable observation from the existing research on GP for autoencoding is the significant drawbacks of techniques that have replaced the whole ANN autoencoder with a GP representation, motivated by GPs interpretability. For the purposes of interpretability, we argue that it is the encoder primarily where there is value. It is the encoder which actually reduces the dimensionality of the data, while the decoder can be thought of as existing merely to evaluate the encoders performance.

Another significant drawback of the work replacing the encoder and decoder with GP individuals such as GPAE and SLGP is that these are fundamentally reliant on each other. A stochastic change in the encoder which *could* lead to to increased autoencoding performance can only ever be recognised if the decoder has the correct structure to identify and evaluate it as such. Since the decoder itself relies on stochastic variation, it is likely these potentially valuable contributions to the encoder will never be sufficiently recognised.

With these drawbacks in mind, we argue there is room for a autoencoder in which the encoder is replaced with GP, while the NN decoder is kept, allowing for consistent evaluation of the the encoder individuals.

2.9 Summary

We have presented a thorough overview of the areas of evolutionary computation, genetic programming, dimensionality reduction and autoencoding. In analysing existing work using GP for NLDR and autoencoding, we believe we have identified a promising new direction for research. Existing GP for autoencoding work has been motivated by the desire for interpretable functional mappings. We believe there is no research that has examined replacing only the encoder of an AE with GP. This is a potentially highly valuable approach, as for the purposes of dimensionality reduction, it is the encoder where the interpretability is primarily useful. By retaining the ANN decoder, we can exploit its proven suitability for the

task of autoencoding, while also gaining interpretability from the use of a GP encoder. This also avoids the problem of trying to simultaneously maintain a GP encoder and decoder, where stochastic changes in one can have drastic effects on the performance of the other. This approach also offers an alternative to existing GP for NLDR work. Instead of using an ad-hoc measure of embedding quality, we can rely solely on whether or not the embedding contains sufficient information to reconstruct the input.

Chapter 3

GPE-AE: Genetic Programming Encoder for Auto-Encoding

3.1 Chapter Overview

In this chapter, we introduce the Genetic Programming Encoder for Auto-Encoding (GPE-AE). First we present the GPE-AE model as whole. Then, we describe the GP representation used for the encoder and the terminals and functions used. Finally, we describe the fitness evaluation of the GP individuals, and the considerations that need to be taken into account when designing the ANN decoder.

3.2 Proposed Model

The conventional ANN approach to autoencoders makes interpretation of the functional mapping difficult due to their opaque structure [29]. Genetic programming (GP) offers a way to learn explainable and re-useable data embeddings [20]. As such, we propose the Genetic Programming Encoder for Autoencoding (GPE-AE). To date, research into GP for autoencoders has focused on replacing the whole ANN autoencoder with GP [24, 35]. We argue that to introduce the benefits of GP to traditional autoencoder representation learning, only the encoder needs to be replaced. This is due to the encoder being the component of autoencoder where the representation is actually learnt, while the decoder is used to validate the quality of the encoding. By keeping the decoder as an artificial neural network, the strengths can be maintained, while still harnessing the advantages of explainable GP models.

GPE-AE also avoids the critical flaws that arise when trying to evolve GP encoders and decoders simultaneously. The EC approach of making small stochastic changes, then reevaluating the solutions relies on consistent and reliable evaluation. However, a stochastic change in the encoder that does lead to a better embedding can *only* be identified as valuable if the decoder has the appropriate structure to recognise it as such. Therefore, approaches which use GP for both the encoder and decoder rely on the fact that the variations being made are going to not just improve the performance, but also be consistent with the behaviour of the other at the time of evaluation. By using a neural network which is trained using the embedding produced by the encoder, GPE-AE is able to accurately evaluate any encoder consistently across the entire evolution.

The overall design of GPE-AE is presented in Fig. 3.1. Here, an example of learning a 3-dimension embedding of n features is used. Taking the original input dataset with features f, these can be used an inputs to a multi-tree GP individual to produce a lower dimension

embedding W, where w_i is dimension i of the embedding. W is then used as the input for the ANN decoder, while the original features f are used as training targets. Once the decoder has been trained, it can then output a prediction of the original features f'. This prediction can be used to evaluate the quality of the embedding, and thus the quality of the GP individual.

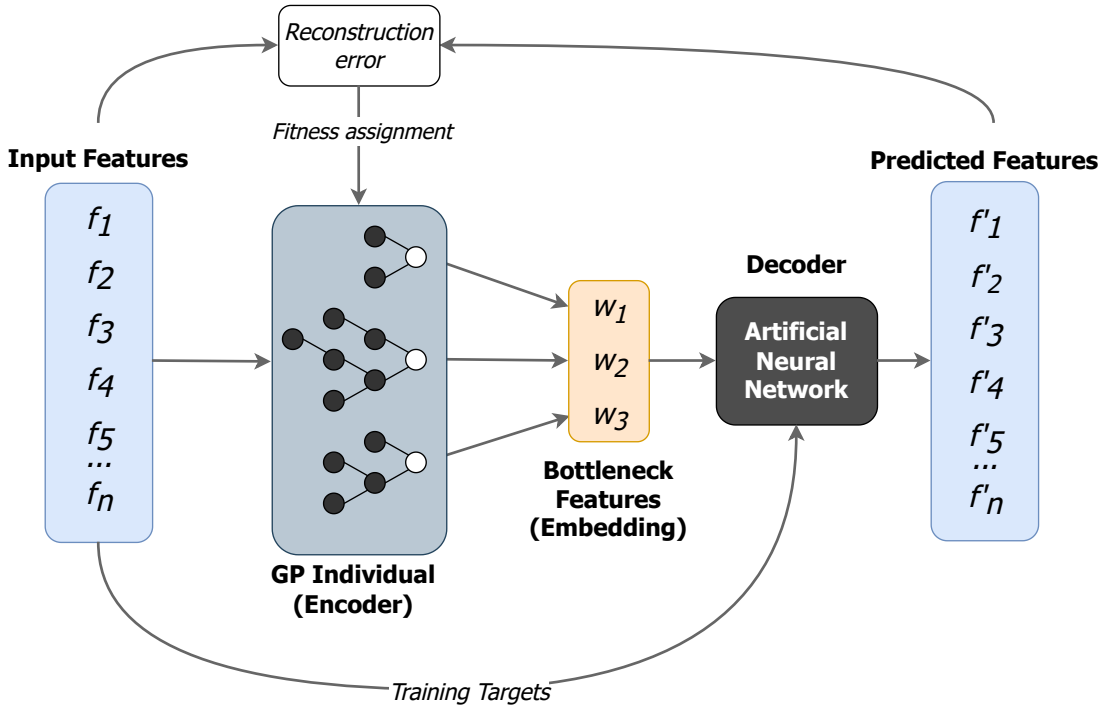


Figure 3.1: An overview of GPE-AE. Here, n features are reduced to a three-dimensional embedding by the GP encoder.

3.3 GP Representation of Encoder

As we require the encoder to take f inputs and produce w outputs, we use a multi-tree GP representation with each individual being comprised of w trees. Each of these trees represents a functional mapping of the f inputs to a single dimension of the w-dimension space. As the encoder is simply performing dimensionality reduction, previous work demonstrating the suitability of the multi-tree representation for this task provides strong motivation for our choice [20].

Other work such as GPAE has argued against the multi-tree representation for autoencoding due to it's inability to share calculations across trees [24], and instead opted for alternate representations. We argue that for the purposes of dimensionality reduction, separating calculations is actually a strength. In theory, each dimension of embedding should have as little shared information as possible, to ensure they are capturing independent parts of the underlying distribution. With the in mind, the ability to share calculations between dimensions could be argued to allow for less separability between embedding dimensions.

3.3.1 Terminal and Function Set

In total 12 functions are used by the encoders, which are presented in Table 3.1. Basic arithmetic operators $+$, $-$ and $\times$ are used. Four modified arithmetic functions are also used.

These are absolute addition and subtraction, an addition function which takes 5 inputs, and protected division % which returns 1 when division by zero is attempted. Absolute arithmetic operators allow for the easier comparison of magnitudes of inputs. The 5+ function allows for more aggressive combination of sub-trees in a more space efficient way. Protecting the division from terminating in error upon a division by zero enforces the *closure* requirement of our GP representation. With the exception of the five addition function, all these functions take two inputs.

Beyond arithmetic, three logical operators are included. These are max and min and if. max and min return the maximum and minimum of their two inputs, respectively. if takes three inputs, using the second as an output if the first is greater than 0, otherwise outputting the third, as in Eq. 5.4. These allow for more expressive use of the original features beyond arithmetic combinations.

$$if(x, y, z) = \begin{cases} y & x < 0, \\ z & \text{otherwise} \end{cases}$$

To allow for non-linear transformations, the ReLU and sigmoid functions are used, defined in Eq. 3.1 and Eq. 3.2. These are commonly used as activation functions in neural networks, adding the capacity for non-linear learning. Existing GP for NLDR work has also used these [20, 21].

$$\text{ReLU}(x) = \max(0, x) \tag{3.1}$$

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

Category	Arithmetic							Logical			Non-Linear	
Function	$+$	$5+$	$-$	$\lvert+\rvert$	$\lvert-\rvert$	$\times$	%	max	min	if	ReLU	sigmoid
No. Inputs	2	5	2	2	2	2	2	2	2	3	1	1

Table 3.1: GP Functions used by GPE-AE. All functions take numeric inputs and produce a single numeric output.

The GP terminals used are the original f features of the data, as well as ephemeral random constants. Ephemeral random constants are random values uniformly sampled over the range $[-1, 1]$, and remain constant over the evolution once initialised. The use of random constants allows for additional dynamic behaviour, such as consistent scaling and offsetting of features.

3.3.2 Crossover and Mutation

The multi-tree GP approach requires adaptions of traditional tree-based GP crossover and mutation.

For extending crossover to the multi-tree case, we suggest three intuitive approaches:

- *Single Index Crossover (SIC)*: Select a single random index of the multi-tree individual, and perform a standard crossover of the trees at the index from each parent.
- *All Index Crossover (AIC)*: Perform standard crossover on *all* pairs of individuals at the same index for each parent.

- *Random Index Crossover (RIC)*: Perform standard crossover from a random tree of each parent.

We argue that using RIC can prevent specific trees from specialising across the population. SIC and AIC both allow this, although AIC is significantly more aggressive. Our exploratory results suggest that there is negligible difference between the approaches in performance, but AIC generally performed best, and as such is the crossover method we use in this work.

For mutation, we simply select a random tree from the individual, and perform standard GP mutation on it.

3.4 Fitness Evaluation

GPE-AE fitness evaluation of an individual I with w trees occurs as follows:

1. The features of input data X are used as inputs for I, producing the embedding W with w dimensions.
2. W is used as input to the ANN decoder, with X serving as training targets.
3. Once training of the decoder is complete, a final prediction X' is made using W as the input to the trained model.
4. The reconstruction error is calculated between the original data X and the reconstruction X', which is assigned to I as the fitness.

As with a standard auto-encoder, the objective function which we seek to optimise is the reconstruction error between the inputs and the predicted outputs. For the GP encoder, we can use this reconstruction error directly as the fitness function of the GP encoder. Specifically, the root mean squared error (RMSE) between the input and output is used. RMSE is defined as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\mathbf{x}'_i - \mathbf{x}_i)^2}{n}} \tag{3.3}$$

where $\mathbf{x}_i$ is the i^{th} instance of the input, and $\mathbf{x}'_i$ is the predicted value of the instance after it has been encoded by the GP individual and reconstruction by the ANN decoder. n is the number of instances in the dataset. We use RMSE as it is better at reflecting performance when dealing with large errors [5].

3.5 Decoder Architecture

The architecture of the ANN decoder requires special consideration. In a conventional auto-encoder, it is common practice to use a "funnel" architecture where the encoder has hidden layers with a decreasing number of neurons, with the decoder reflecting the encoder. GPE-AE, however, uses a dynamically structured GP encoder. In this work, we focus on using a decoder with fixed architecture and hyper-parameters for all individuals in the population although future work could potentially explore dynamic decoders that take the structure of the GP encoder into consideration.

For GPE-AE, we propose the use of a simple multi-layer perceptron for the decoder. The input layer has w inputs, one for each embedding dimension. The output layer has f

outputs, one for each of the features of the data in the original space. The parameters that must be determined are the design of the hidden layers. More specifically, we must choose how many hidden layers there are, and how many neurons each has.

Within GPE-AE, the role of the decoder is only to evaluate the performance of the GP encoder. As such, we do not require the decoder to be *perfect*, but merely to be able to reliably distinguish the performance of encoder candidates in a consistent way. In fact, due to requirement of training a NN decoder for each evaluation (of which there can be many), using a complex, deep ANN decoder can come at a significant computational time cost. As such, we suggest the use of fairly simple decoder architectures.

As ANN training is not entirely deterministic, we use a raw numerical representation of the encoder as a random seed for the training of the decoder, ensuring an individual always evaluates to same fitness.

Chapter 4

Experiment Design

4.1 Chapter Overview

In this chapter, we outline the design of the experiments run to evaluate the performance of GPE-AE. First, we state the different embedding sizes and decoder architectures we propose to test GPE-AE with, as well as the parameters that we hold constant throughout the experiments. Then, we present the two methods that we use to compare the performance of GPE-AE to, and introduce the measures that are used to compare the different methods. Finally, we present the datasets the the methods will be tested on.

4.2 Experiment Configurations

We are interested in the role the complexity of the decoder plays in the performance of GPE-AE. As such, we suggest three different decoder configurations to be used in our experiments. We perform experiments with decoders with 1, 2 and 3 hidden layers. The number of neurons at each layer is presented in Table 4.1. We follow the common "funnel" architecture of AEs, such that the decoder incrementally expands the data from the bottleneck until the final reconstruction is made. As the depth of the decoder increases, so does the number of connections. This can allow for more complex decoding structures to be modeled, and potentially allow for better evaluation of the encodings. More calculations also means more parameters to learn. This can have a significant impact on run time, which is valuable considering the number of individual evaluations in GP.

No. Hidden Layers	Encoder Arrangement	Decoder Arrangement
1	[128]	[128]
2	[128, 64]	[64, 128]
3	[128, 64, 32]	[32, 64, 128]

Table 4.1: The NN architectures used by GPE-AE and CAE in the experiments. GPE-AE only makes use of the decoder.

Additionally, for each of the hidden layer configurations and datasets used, we are also interested in how GPE-AE performs on dimensionality reduction tasks of varying difficulty. To evaluate this, we perform experiments using 1, 2, 3, 5 and 10 embedding dimension sizes. We select 1, 2 and 3 dimensions as these are more difficult dimensionality reduction tasks, and can be useful for visualisation. 5 and 10 dimensions represent easier, but still useful dimensionality reduction tasks.

Standard GP parameters used by GPE-AE for all experiments, which are presented in Table 4.2.

For the decoder, standard neural network parameters are used, which are presented in Table 4.3. We choose 100 epochs for the sake of keeping fitness evaluation computational costs down, as a NN is required to be trained for each evaluation. We argue that this is sufficient for evaluating and comparing individuals: even though it is possible the decoder could achieve a higher reconstruction performance with more epochs, it could also introduce problems with overfitting. Additionally, since all individuals are evaluated with the same constraint, it should not effect the ability to compare them. The ReLU activation is used to add non-linearties between hidden layers, much in the same way GP can use ReLU functions. The Adam optimiser has been found to perform well for regression problems, which reconstruction essentially is [14]. The learning rate and mini-batch size were found to allow sufficient evaluation performance in exploratory testing.

Parameter	Setting	Parameter	Setting
Generations	1000	Pop.Size	100
Mutation	20%	Crossover	80%
Elitism	top 10	Pop. Init.	Half-and-half
Selection	Tournament	Tourn. Size	7
Min. Tree Depth	2	Max. Tree Depth	8

Table 4.2: GP parameters used for GPE-AE and GP-MaL.

Parameter	Setting
Epochs	100
Learning Rate	0.001
Mini-batch Size	200
Optimiser	Adam
Activation	ReLU

Table 4.3: Neural network parameters used for decoder in GPE-AE and encoder/decoder in conventional AE.

For each dataset, embedding size, and number of hidden layers, we perform 10 runs using GPE-AE and both of the comparison methods stated in Section 4.3. This accounts for the stochastic nature of the evolutionary process.

4.3 Comparison Methods

To evaluate the performance of our proposed GPE-AE algorithm, we compare it to two relevant baselines. The first is a conventional ANN auto-encoder (CAE), and the second is GP-MaL.

4.3.1 Conventional Auto-Encoder

To better evaluate the quality of the GP encoder, we compare GPE-AE to a CAE. As with GPE-AE, we perform experiments using the same hidden layer configurations as GPE-AE. The architecture of the encoder mirrors that of the decoder, as presented in Table 4.1. By

mirroring the decoder, we can be sure that any structure capable of being found by the decoder is capable of being reversed by the decoder. We train the CAE using the same standard NN hyper-parameters as we use for the GPE-AE decoder, as in Table 4.3.

The objective function optimised by the CAE is the reconstruction error–the MSE between the input and the output prediction.

4.3.2 GP-MaL

While it is useful to compare GPE-AE to a CAE, the motivation behind GPE-AE is to use the autoencoder structure to produce functional mappings for NLDR with GP. As such, it is valuable to compare it to another multi-tree GP manifold learning method, GP-MaL. By comparing GPE-AE to GP-MaL, we can better evaluate GPE-AE as a NLDR method.

As opposed to evaluating the embedding by attempting to reconstruct the original values, GP-MaL measures how well *neighborhood orderings* are preserved between the original space and the embedding space. The exact GP-MaL fitness is presented in Section 2.8.1.

Our GP-MaL experiments use the same evolutionary parameters and terminal and function sets as the GPE-AE experiments. The only way in which the GPE-AE and GP-MaL differ in our experiments is in fitness evaluation.

4.4 Evaluation Measures

4.4.1 Classification Accuracy

As NLDR and autoencoding are unsupervised, there is no "gold-standard" objective measure to compare methods using different approaches. Using either the reconstruction error or the GP-MaL fitness would favour the methods which use the selected objective directly. As such, to evaluate and compare the performance of our GPE-AE and the two baseline methods for dimensionality reduction, we propose using the classification accuracy obtained using the low-dimension embedding. This approach has been used in previous GP for NLDR work [20, 21].

The performance of a classification algorithm on a embedding relies on data labels, which are not available to GPE-AE or our comparison methods during training, thereby easing concerns of bias towards any particular approach. This does however rely on the assumption that the data labels are important to the structure of the data. We argue that this assumption generally holds, as the ability of the classification algorithm to separate data in a low-dimensional space indicates that some sort of sensible structure exists.

It is important to acknowledge that classification accuracy is only useful for comparing different methods on the same dataset. That is, a single methods classification accuracy on one dataset being higher than on another does not mean that method is better at performing dimensionality reduction on the first dataset, as we can not assume the level of association between the labels and the true structure of the datasets.

The classification algorithm used for evaluation is the scikit-learn Random forest implementation, using 100 trees. Random forest is a relatively cheap algorithm which has been found the perform well on a range of datasets, making it a good choice for unbiased evaluation [44]. We calculate the classification accuracy using 10-fold-cross-validation on the low dimensional embedding.

4.4.2 Number of Connections

For comparing GPE-AE to CAE, we measure the number of *connections* that the GP encoder and the ANN encoder have. As both are essentially graphs, we are counting the number of edges in each. For a GP individual, this is $|nodes| - w$, where w is the number of trees in the individual, as each node except the root nodes have a single parent. For a fully-connected neural network, this is defined by the equation:

$$\sum_{i \in L}^{L-1} L_i L_{i+1} \tag{4.1}$$

Where L is the number of layers in the network, and L_i is the number of nodes as layer i.

This isn't necessarily a *fair* comparison for a few reasons. The number of edges in a GP individual is dynamic while connections in a NN are fixed, and the edges in a conventional fully-connected NN represent more consistent operations than the range of functions a GP node can take. However, providing this number can highlight the ease of interpreting a GP individual compared to a NN when the number of connections is *significantly* smaller.

4.5 Datasets

The datasets used are presented in Table 4.4. These are mostly real-world datasets. Clean1 is from openML [1], while the rest are from the UCI Repository [2]. The selected datasets have a range of different dimensionalities, classes and instances to evaluate the performance of GPE-AE across different problems.

Dataset	Instances	Features	Classes
Clean1	476	168	2
Dermatology	358	34	6
Ionosphere	351	34	2
Segmentation	2310	19	7
Wine	178	13	3

Table 4.4: Classification datasets used for experiments.

[1] `https://www.openml.org/`
[2] `https://archive.ics.uci.edu/`

Chapter 5

Results & Discussion

5.1 Chapter Overview

In this chapter, we present and discuss the results of our experiments. First, we assess the performance of GPE-AE against a conventional autoencoder (CAE). We do this using our two selected measures: classification accuracy and number of connections. Then, we assess the results of the experiments comparing to GP-MaL, using classification accuracy. To better understand the qualitative performance of GPE-AE, we present two further analysis methods. First, we analyse some two-dimensional visualisations produced by GPE-AE and the two comparison methods, then we analyse some of the GP individuals produced by GPE-AE to demonstrate the value of interpretability.

5.2 Results

5.2.1 GPE-AE vs. CAE

The results of the experiments comparing GPE-AE to CAE are presented in Table 5.1. The results are grouped vertically by the dimensionality of the embedding. GPE represents our approach, GPE-AE, and CAE represents the conventional autoencoder comparison method. HL is the number of hidden layers used for the decoder in GPE-AE, and both the encoder and decoder in CAE. For example, GPE 3HL is GPE-AE with a 3-hidden layer decoder, while CAE 2HL is a conventional autoencoder with two hidden layers in each the decoder and encoder. The number of neurons at each layer is presented in Table 4.1.

Classification Accuracy

The mean classification across the 10 runs for each method on each dataset is presented in the Acc. column. A Wilcoxon significance test was performed with a p-value of 0.05. The tests were performed using each configuration of hidden layers. GPE 1HL is tested against CAE 1HL, GPE 2HL against CAE 2HL and GPE 3HL against CAE 3HL for each for of the embedding sizes, and for each dataset. A "+" next to a GPE accuracy indicates that GPE significantly outperformed CAE with the same hidden layer configuration on this dataset, while a "−" indicates GPE performed significantly worse.

On the Clean1 dataset, the two methods perform similarly on all dimensionality reduction tasks. Using three hidden layers, GPE-AE outperforms the CAE reducing to two dimensions, and using one hidden layer the CAE outperforms GPE-AE reducing to ten dimensions. This is the only experiment on which the CAE outperformed GPE-AE.

On the Dermatology dataset, GPE-AE outperformed the CAE on most experiments. There was no significant difference found for three of the configurations: Using one hidden layer to reduce to five and ten dimensions, and using two hidden layers to reduce to ten dimensions.

On the Ionosphere dataset, GPE-AE outperformed the CAE on all but two of the experiments. These were using two hidden layers to reduce to one dimension, and using one hidden layer to reduce to three dimensions.

On the Segmentation dataset, GPE-AE outperformed the CAE in all experiments configurations.

On the Wine dataset, using one hidden layer GPE-AE outperformed the CAE on all dimensionality reductions. Using two hidden layers, GPE-AE outperformed the CAE in for all dimensions except three and five. Using three hidden layers, GPE-AE outperformed the CAE only when reducing to one dimension. For all other experiments, no significant difference was found.

From our results, GPE-AE was generally better at producing embeddings for classification for all the "easier" datasets with 34 or less original features. This indicates that the GP approach is able to find embeddings with a more separable structure of classes. This may be due to the fact that GP attempts random changes to the encoding, then evaluates their quality. An extreme separation may be found by the GP encoder, which may be kept as long as the decoder is able to produce a sufficient reconstruction from it. The conventional NN approach to autoencoding however relies on directed iterative, small changes. Once a sufficient minimal separation of instances is found for reconstruction, the training is likely to converge. This embedding, while maybe good for reconstruction using the NN decoder, is not necessarily as separated as the GP-produced embedding could be.

On the significantly harder problem of the Clean1 dataset with 168 features, the performance was generally equivalent. Clean1 has two classes, and as such all methods obtaining approximately 0.54 classification accuracy when reducing to one dimension on the embedding indicates the classifier is only doing slightly better than randomly assigning labels. It is possible that the underlying structure of the data is not related to the labelling, however the classification performance increasing with the embedding size indicates there is *some* structure, it is just not able to captured in a single dimension by either of the methods. This is unsurprising, as reducing 168 features to a single dimension is inherently a difficult task assuming most of the features are not irrelevant or redundant.

Neither of the methods seem to perform better or worse comparatively with different hidden layer configurations. Using one hidden layer, GPE-AE outperforms CAE on 17 experiments, using two hidden layers on 16 experiments, and using three hidden layers on 17 experiments. Likewise, the dimensionality of the embedding seems to effect the comparative performance of the two methods. Reducing to one dimension, GPE-AE outperforms the CAE on 11 experiments, reducing to two dimensions on 12 experiments, and reducing to three, five and ten dimensions on 0 experiments each.

Number of Connections

For GPE, the average number of connections of the best individuals found across the 10 runs for each method is presented. The number of connections in the neural networks is consistent for all runs of the same configuration, and is presented for comparison.

No significant testing is used to compare numbers of connections, as it is immediately obvious (and expected) that GP individuals have significantly less connections than a neural network encoder. However, this helps to highlight the interpretability gap between the two approaches. Even though the CAE designs we propose are fairly simple, they still have an

extraordinary number of connections compared to GP individuals, especially on the larger Clean1 dataset.

To highlight the effect the complexity of the decoder has on the complexity of the encoder, the smallest average individual size for each dataset reducing to each dimensionality is in bold. We can see that for all except Segmentation to two dimensions, the two and three hidden layer decoders lead to the smallest GP encoders. This is likely explained by the fact that the more complex the decoder, the less "work" the encoder has to do. That is, the more sophisticated transformations the decoder is capable of learning, the better it can be at reconstructing the embeddings resulting from simpler encoders.

5.2.2 GPE-AE vs. GP-MaL

The results of the experiments comparing GPE-AE to GP-MaL are presented in Table 5.2. The results are grouped vertically by the dimensionality of the embedding. For each embedding dimension, we present GPE with the three different hidden layer configurations and GP-MaL. HL is the number of hidden layers used for the decoder in GPE-AE. The number of neurons at each layer is presented in Table 4.1. For each experiment configuration, we report the mean classification accuracy using the low dimensional embedding across the 10 experiments.

To compare the results, we have performed a Friedman significance test comparing the three hidden layer configurations to GP-MaL, which is used as a control. We use a p-value of 0.05, and if a significant difference is found, we proceed to Holm post-hoc analysis, also with a p-value of 0.05. We do this for each embedding dimension on each dataset. A "$+$" next to a method indicates that the GPE-AE configuration significantly outperformed the GP-MaL control on the given dataset in the given embedding dimension, and likewise a "$-$" indicates it performed significantly worse.

We can see that on the Clean1, Ionosphere, and Wine datasets, no significant classification performance differences were found between any of the GPE-AE configurations and the GP-MaL control. On the Segmentation dataset, GPE-AE with one and two hidden layers outperformed GP-MaL when reducing to two dimensions, with no significant different being found for any of the other embedding dimensions.

On the Ionosphere dataset, GPE-AE performed significantly worse than GP-MaL for all hidden layer configurations when reducing to one and two dimensions. When reducing to five dimensions, GPE-AE performed significantly worse using three hidden layers. When reducing to ten dimensions, GPE-AE performed significantly worse using two hidden layers For all other experiments on the ionosphere dataset, no significant difference was found.

The results of these experiments demonstrate the quality of GPE-AE at interpretable dimensionality reduction compared to a similar existing technique. On only the single Ionosphere dataset did our method perform worse. This may suggest that for this dataset, there is some relationship between the labels and the structure of the data that makes preservation of neighborhood orderings more valuable or informative than purely targeting reconstruction. Furthermore, GPE-AE with a single hidden layer decoder was only outperformed on Ionosphere on the one and two dimension reduction cases, which do represent the most difficult dimensionality reduction tasks. Furthermore, the fact that on the much higher dimensionality dataset, Clean1, no significant difference was found even for the harder dimensionality reduction tasks suggests that this may be a particular quirk with the Ionosphere dataset. This demonstrates the subjective approaches to unsupervised learning having different strengths and weaknesses.

	Clean1		Derma.		Iono.		Segmen.		Wine	
Method	Acc.	Conn.	Acc.	Conn.	Acc.	Conn.	Acc.	Conn.	Acc.	Conn.
1 Dim.										
GPE 1HL	0.542	204	0.786+	176	0.867+	214	0.631+	218	0.914+	193
CAE 1HL	0.541	21632	0.456	4480	0.84	4480	0.340	2560	0.630	1792
GPE 2HL	0.544	302	0.803+	210	0.871	237	0.663+	214	0.908+	**178**
CAE 2HL	0.526	32320	0.596	6592	0.843	6592	0.427	3712	0.723	2560
GPE 3HL	0.546	**132**	0.779+	**100**	0.865+	**192**	0.641+	**201**	0.912+	181
CAE 3HL	0.523	34336	0.674	8608	0.831	8608	0.384	5728	0.665	4576
2 Dim.										
GPE 1HL	0.605	277	0.874+	315	0.876+	266	0.740+	**231**	0.938+	386
CAE 1HL	0.593	21760	0.681	4608	0.85	4608	0.407	2688	0.766	1920
GPE 2HL	0.622	**276**	0.894+	301	0.886+	244	0.754+	262	0.933+	**342**
CAE 2HL	0.600	32384	0.753	6656	0.855	6656	0.523	3776	0.827	2624
GPE 3HL	0.583+	277	0.872+	**267**	0.890+	**238**	0.707+	392	0.916	352
CAE 3HL	0.536	34368	0.755	8640	0.850	8640	0.564	5760	0.855	4608
3 Dim.										
GPE 1HL	0.656	343	0.922+	330	0.886	498	0.779+	344	0.945 +	**302**
CAE 1HL	0.611	21888	0.783	4736	0.867	4736	0.673	2816	0.843	2048
GPE 2HL	0.642	328	0.909+	425	0.897+	342	0.779+	299	0.938	367
CAE 2HL	0.622	32448	0.817	6720	0.870	6720	0.602	3840	0.906	2688
GPE 3HL	0.628	**106**	0.911+	**305**	0.893+	**277**	0.786+	257	0.940	518
CAE 3HL	0.620	34400	0.786	8672	0.846	8672	0.564	5792	0.867	4640
5 Dim.										
GPE 1HL	0.670	389	0.927	497	0.901+	378	0.882+	428	0.954+	382
CAE 1HL	0.664	22144	0.896	4992	0.877	4992	0.715	3072	0.907	2304
GPE 2HL	0.697	**242**	0.922+	436	0.903+	431	0.888+	**327**	0.931	385
CAE 2HL	0.693	32576	0.872	6848	0.876	6848	0.680	3968	0.927	2816
GPE 3HL	0.679	391	0.897+	**346**	0.904+	**297**	0.873+	338	0.936	547
CAE 3HL	0.657	34464	0.837	8736	0.874	8736	0.619	5856	0.937	4704
10 Dim.										
GPE 1HL	0.715−	543	0.941	766	0.909+	646	0.914+	549	0.966+	799
CAE 1HL	0.746	22784	0.938	5632	0.891	5632	0.774	3712	0.944	2944
GPE 2HL	0.747	**356**	0.933	772	0.905+	**378**	0.916+	**384**	0.964+	832
CAE 2HL	0.712	32896	0.929	7168	0.882	7168	0.778	4288	0.946	3136
GPE 3HL	0.709	359	0.943+	**452**	0.911+	512	0.911+	559	0.949	**655**
CAE 3HL	0.702	34624	0.867	8896	0.870	8896	0.728	6016	0.946	4864

Table 5.1: GPE-AE compared to conventional auto-encoders (CAE).

Method	Clean1	Derma.	Iono.	Segmen.	Wine
1 Dimension					
GPE 1HL	0.542	0.786−	0.867	0.631	0.914
GPE 2HL	0.544	0.803−	0.871	0.663	0.908
GPE 3HL	0.546	0.779−	0.865	0.641	0.912
GPMaL	0.582	0.915	0.868	0.649	0.883
2 Dimensions					
GPE 1HL	0.605	0.874−	0.876	0.740+	0.938
GPE 2HL	0.622	0.894−	0.886	0.754+	0.933
GPE 3HL	0.583	0.872−	0.890	0.707	0.916
GPMaL	0.621	0.935	0.889	0.714	0.937
3 Dimensions					
GPE 1HL	0.656	0.922	0.886	0.779	0.945
GPE 2HL	0.642	0.909	0.897	0.779	0.938
GPE 3HL	0.628	0.911	0.893	0.786	0.940
GPMaL	0.639	0.928	0.899	0.830	0.950
5 Dimensions					
GPE 1HL	0.670	0.927	0.901	0.882	0.954
GPE 2HL	0.697	0.922	0.903	0.888	0.931
GPE 3HL	0.679	0.897−	0.904	0.873	0.936
GPMaL	0.689	0.953	0.911	0.895	0.947
10 Dimensions					
GPE 1HL	0.715	0.941	0.909	0.914	0.966
GPE 2HL	0.747	0.933−	0.905	0.916	0.964
GPE 3HL	0.709	0.943	0.911	0.911	0.949
GPMaL	0.755	0.963	0.907	0.932	0.963

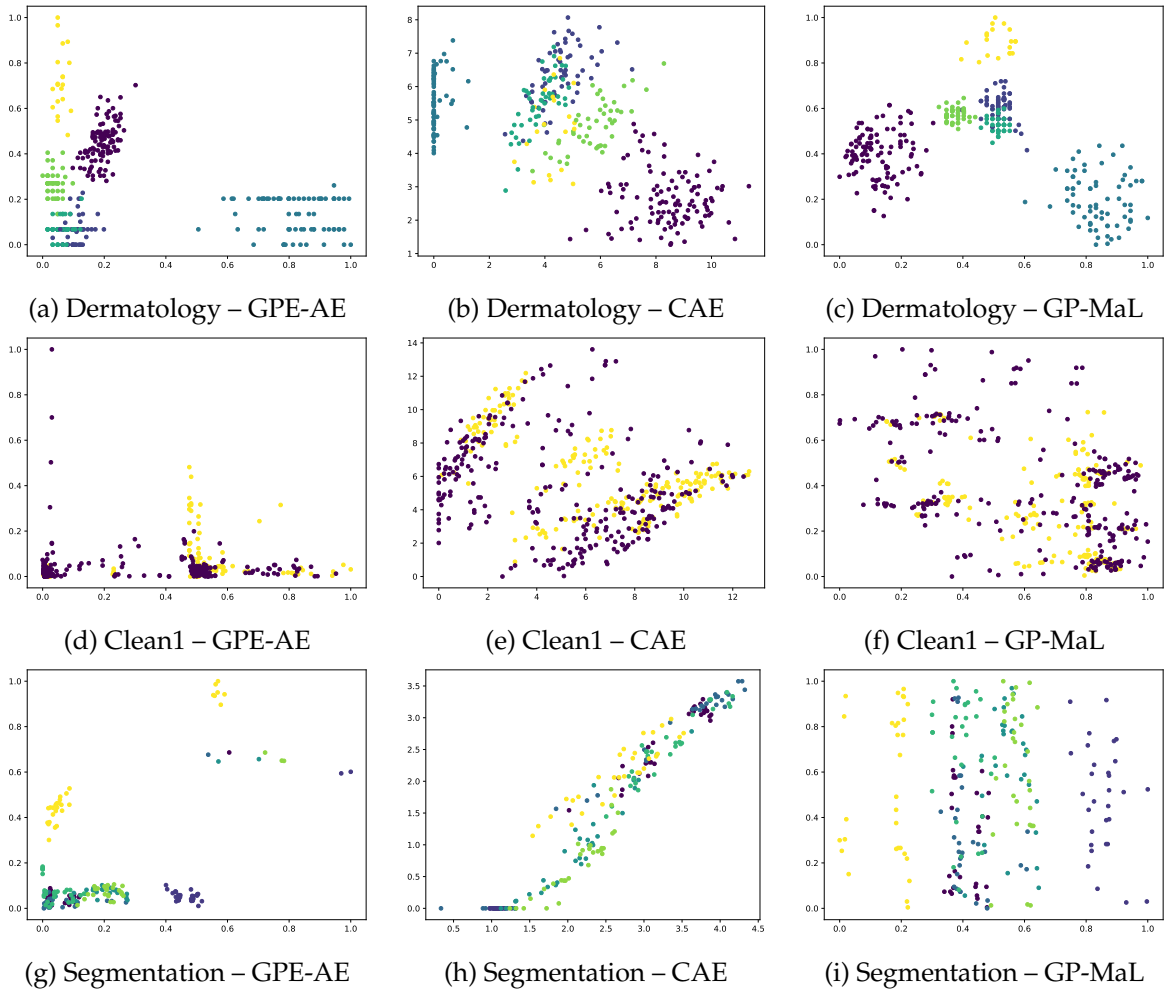
Table 5.2: GPE-AE compared to GP-MaL in classification accuracy on embedding.

5.3 Visualisation Analysis

A common dimensionality reduction task is the reduction of data to two dimensions, explicitly for the sake of visualising the data. As such, we analyse some two-dimensional embeddings produced by GPE-AE and our two comparison methods to evaluate GPE-AEs suitability for this task. In Fig. 5.1 we present two-dimensional embeddings produced by median performing runs for GPE-AE and the two comparison methods for the Dermatology, Clean1 and Segmentation datasets. We select these datasets as they represent a range of difficulties in terms of number of original features: 19 for Segmentation, 34 for Dermatology and the significantly more difficult Clean1 with 168 features.

The Dermatology visualisations are shown in Fig. 5.1a, Fig. 5.1b and 5.1c. The most apparent difference between the visualisations is that GPE-AE is more condensed than the other two visualisations. Nearly all the instances are in the left third of the x axis, with single loose cluster outside in the bottom right. The other two visualisations comparatively are more evenly distributed across the space.

Also notable in the GPE-AE visualisation is that instances seem to be grouped in rigid



(a) Dermatology – GPE-AE (b) Dermatology – CAE (c) Dermatology – GP-MaL

(d) Clean1 – GPE-AE (e) Clean1 – CAE (f) Clean1 – GP-MaL

(g) Segmentation – GPE-AE (h) Segmentation – CAE (i) Segmentation – GP-MaL

Figure 5.1: Visualisations produced by the GP methods and a conventional auto-encoder (CAE) on the Dermatology, Clean1 and Segmentation datasets. The median result of each was chosen for visualisation.

"steps" along the y axis. This is especially apparent in the darker blue class. This grid-like behaviour is also somewhat apparent in the GP-MaL visualisation, but not in the CAE visualisation. This is likely explained by the fact CAE is able to learn "smoother" mapping functions due to the large number of calculations, whereas the GP methods may have found relatively simply functional trees that were sufficient in terms of fitness. While the grid-like behaviour is not present in CAE, it has grouped the majority of the light blues instances along a single$x$ value. Notably, all visualisations have captured a defined separation of the light blue class from the others. Both of the GP methods have a clear separation of the yellow class, while the CAE visualisation does not.

The Clean1 visualisations are shown in Fig. 5.1d, Fig. 5.1e and 5.1f. Again here we see that GPE-AE has a bigger spread over the space in total, resulting in most instances being placed in the lower third of the visualisation. The other two approaches are more evenly distributed over the space. None of the methods seem to separate the two classes well, which is consistent with the reported classification accuracy of these methods on reducing the Clean1 dataset to two dimensions. This may be a result of the class distribution not being related to the actual numeric distributions of the original features, or at least not in a way that is able to expressed in two dimensions.

The Segmentation visualisations are shown in Fig. 5.1g, Fig. 5.1h and 5.1i. Follow-

ing the trend of the other visualisations, GPE-AE places most instances in a concentrated area, in this case the lower left corner, with far distant outliers. Analysing the visualisations produced by the other methods, we see that they visualisations look like one-dimensional representations projected onto a two-dimension space. In GP-MaL in particular, a decent classification accuracy could be achieved by simply separating along the x axis. The CAE visualisation does not have this property, instead grouping all instances roughly in a single diagonal, although there is still *some* grouping of classes.

Overall these visualisations suggest GPE-AE is prone to producing less compact visualisations, with certain instances being projected further out than they are by the two comparison methods. The GP-MaL approach to preserving neighborhood orderings potentially helps to keep the embeddings closer.

The difference in behaviour between GPE-AE and CAE is more interesting, as they both seek to optimise the same objective function. This may be explained by the two very different approaches to this optimisation. The gradient descent approach of the CAE leads to a more iterative optimisation: the embedding is optimised in steps until training is complete. GPE-AEs trial and error approach may lead to a more aggressive approach to optimising reconstruction error: instances can be placed far away in the embedding as a result of a stochastic update to a tree, and as long as this does not effect the ability of the decoder to reconstruct the original output there is no evolutionary pressure to bring instances more ”sensibly” together.

5.4 Evolved Individual Analysis

One of the key strengths of GPE-AE over a conventional AE is the interpretable tree-based representation. As such, it is valuable to analyse some of the evolved functional mappings to demonstrate some of the insights that can be gained by using GPE-AE. Here we present four such individuals, each performing a different level of dimensionality reduction on a different dataset, and analyse them to understand the functional mapping. The non-linear operators ReLU and Sigmoid are highlighted in the trees to easier observe the non-linear components of the functional mappings.

Dermatology to 1 Dimension

Fig. 5.2 shows a functional mapping for reducing the Dermatology dataset to a single dimensions. It makes use of a single non-linear operator: a sigmoid function with the input $\min(f9, \max(f21, f13))$. This suggests these features may have some non-linear relationship to the underlying distribution of the data. This mapping uses 12 unique features of Ionosphere's original 34. From this, we can infer that only roughly 35% are required to create an embedding with sufficient information to create a reasonable reconstruction from. Random-forest classification achieved an accuracy of 0.782 using the embedding produced by the individual.

Segmentation to 2 Dimensions

Fig. 5.3 shows a functional mapping for reducing the Segmentation dataset to two dimensions. The first tree (a) is fairly simple, being f_{18} with a constant value added. This is essentially just feature selection, as the first dimension of the embedding will just be f_{18} offset by the very small 0.0006. This is an example of an intron, even though it does *technically* effect the output. Although a single f_{18} would make more sense, there is no evolutionary pressure to remove it, as the individuals would possess the same fitness.

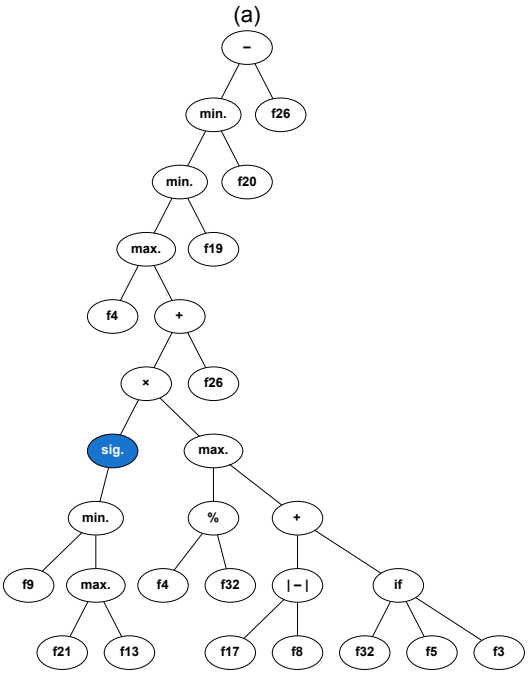


Figure 5.2: A GP encoder for reducing the Dermatology dataset containing 34 features to a single feature, of which 12 unique features are used.

The second tree (b) is significantly more complex, and makes heavy use of non-linear functions. Notably $ReLU(f_{16})$ is present twice, and $ReLU(f_{18})$ is present three times. Another interesting observation is the two chains of ReLU functions off the +5 node in the middle of the tree. Using a ReLU output as an input to another does not change the output, and as such this an example of some redundancy in tree. Due to this not affecting the output, there is no evolutionary pressure for this to take a less redundant form.

9 of the original 19 features are used. Random-forest classification achieved an accuracy of 0.829 using the 2-dimension embedding produced by the individual.

Ionosphere to 5 Dimensions

Fig. 5.4 shows a functional mapping for reducing the Ionosphere dataset to 5 dimensions.

Notably, trees (b), (c) and (d) are performing simple feature selection of the features f_{30}, f_{19}, f_{18}. Tree (e) represents the simple function $|(f_{33} + f_{33})| + f_{33}$. This functional mapping can be expressed as:

$$w_e(f) = \begin{cases} f_{33} & f_{33} < 0, \\ 3f_{33} & \text{otherwise} \end{cases}$$

Essentially, positive values are scaled by a factor of 3, while negative values are passed through untouched. Analysis of the dermatology dataset shows that roughly half of the original values of f_{33} are negative, meaning this isn't redundant behaviour that has no effect on the output: it would noticeably impact the distribution of this dimension in the embedding. This is an example of an insight that can be gained through interpretable dimensionality reduction, and can help guide further investigation and understanding of the original data.

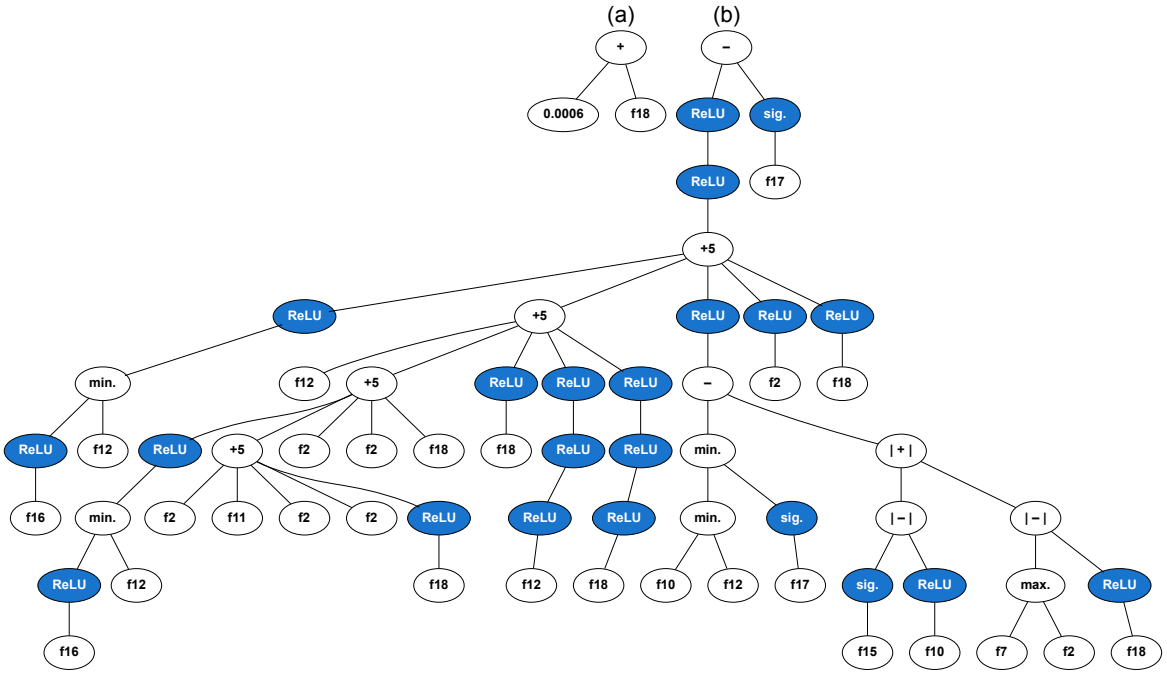


Figure 5.3: A GP encoder for reducing the Segmentation dataset containing 19 features to a 2 features, of which 9 unique features are used.

Also notable about this functional mapping is the complete lack of non-linear functions. This also provides further insights into the underlying structure of the Ionosphere dataset. It also demonstrates that GPE-AE is capable of more straightforward combinations of features if non-linear operators are not required.

11 of the original 34 features are used. Random-forest classification achieved an accuracy of 0.880 using the 5-dimension embedding produced by the individual.

Clean1 to 10 Dimensions

Fig. 5.5 shows a functional mapping for reducing the Clean1 dataset to 10 dimensions. Immediately apparent is the fact that 5 trees (a), (b), (c), (d) and (e) and performing simple feature selection. Tree (f) is also extremely simple and informative, just the single feature $f162$ with a ReLU function applied. Furthermore, two of the individuals trees, (g) and (i) are fairly simple, straightforward mathematical functions. (i) is $f118 + f65$, while (g) is a slightly more complex but still an easily expressable and understandable function with non-linear properties:

$$w_g(f) = -(|+|(f_{28}, f_{127}), Sig(f_{127})) = |(f_{28} + f_{127})| - \frac{1}{1 + e^{-f_{127}}} \tag{5.1}$$

The remaining two trees, (h) and (j) are significantly more complex. However, we can again identify the inputs to the non-linear operators to understand how these features play a role in the NLDR of the Clean1 dataset. In tree (j), for example, the function $ReLU(f_{76})$ appears *three* times, and $Sig(f_{17})$ appears twice. This potenially indicates these features playing an important role in the non-linear mapping.

Overall this tree uses 39 of the original 168 features, a significant reduction. Random-forest classification achieved an accuracy of 0.757 using the 10-dimension embedding pro-

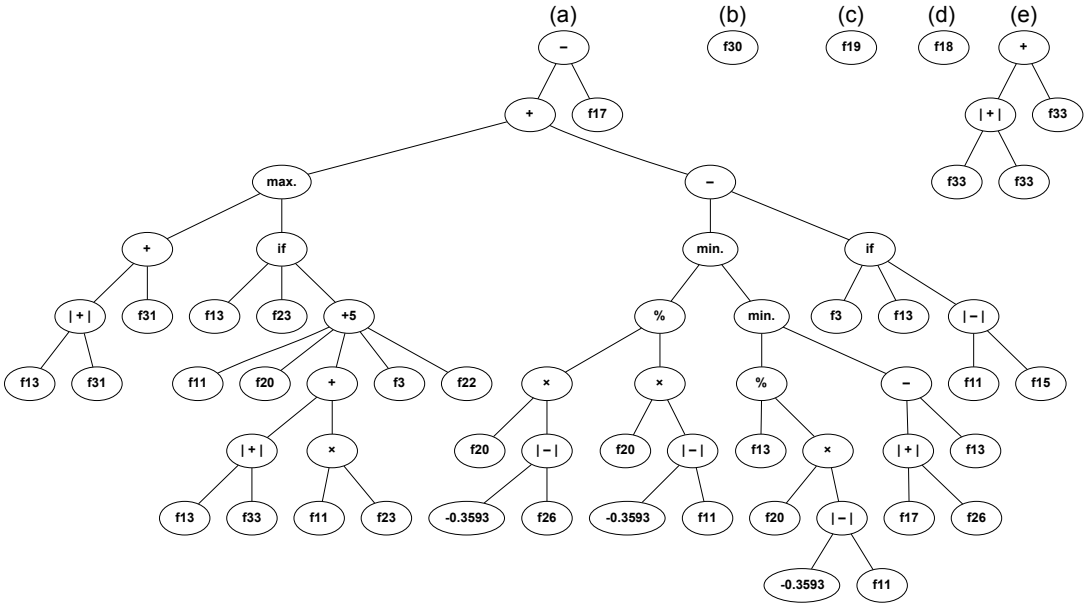


Figure 5.4: A GP encoder for reducing the Ionosphere dataset containing 34 features to a 5 features, of which 11 unique features are used.

duced by the individual.

5.5 Summary

In this chapter, we have presented the results of experiments comparing out proposed method, GPE-AE, to two comparison methods: a conventional autoencoder (CAE) and another GP for NLDR method, GP-MaL. We have shown that compared to the CAE with simple architecture, GPE-AE is able to perform competitively at producing embeddings which present some structure of the labeled data. On all datasets except for one, GPE-AE outperformed the CAE on most experiments. We suggested this may be due to the difference in optimisation strategies leading to more separation of instances for the purposes of classification. We also highlighted the significant difference in the complexity of the two models, by presenting the number of connections the two methods used in the encoder. This illustrated the potential for interpretability when using GPE-AE.

In comparing GPE-AE to GP-MaL, we demonstrated that on all except the Ionosphere dataset, GPE-AE is competitive with an existing GP for NLDR technique.

To understand the qualitative value of GPE-AE, we also analysed two-dimensional visualisations found by GPE-AE and the two comparison methods on three datasets. We found that generally GPE-AE produced visualisations with far flung groupings of instances in compared to the other two methods. We suggested this may be due to the trial-and-error approach to optimising reconstruction error. We also analysed GP encoder individuals produced found for four different datasets reducing to four different dimensionalities. Through this, we demonstrated some insights that can be gained from being able to interpret the encoder, they key strength of our approach over conventional autoencoders.

We believe that these results demonstrate that GPE-AE is both a competitive approach to interpretable autoencoding, and to interpretable non-linear dimensionality reduction in general.

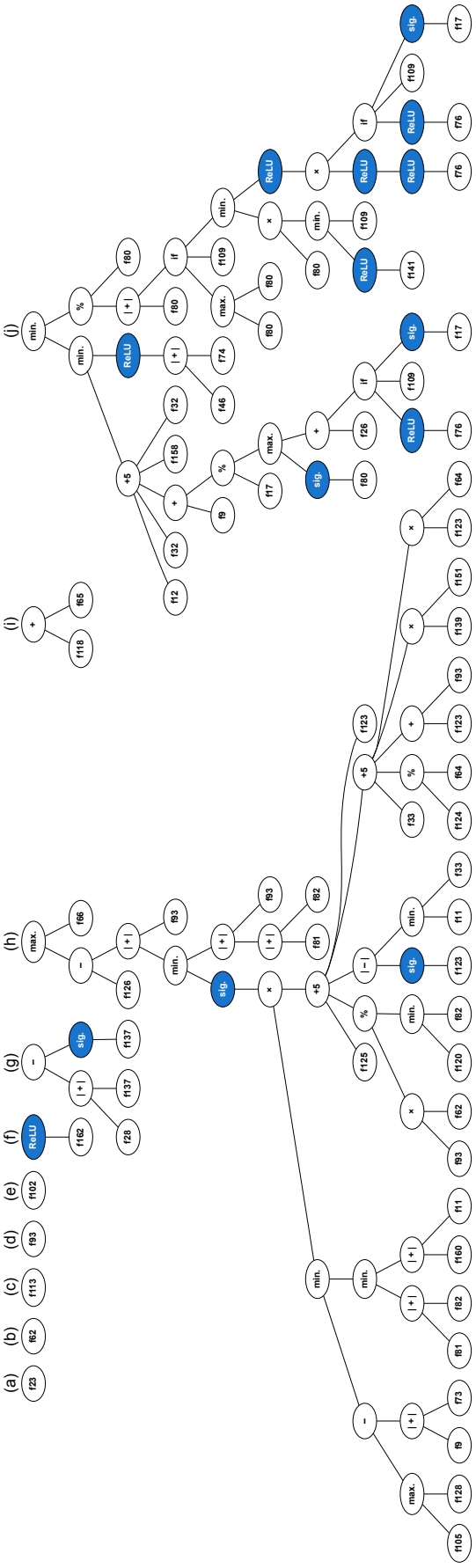


Figure 5.5: A GP encoder for reducing the Clean1 dataset containing 168 features to a 10 features, of which 38 unique features are used.

Chapter 6

Conclusions

In unsupervised machine learning, dimensionality reduction is an important task. Dimensionality reduction allows us the ability to find simpler representations of data, which can make it easier to interpret and work with. However, more complex data often requires the use of more powerful non-linear dimensionality reduction techniques.

These non-linear reduction techniques can be divided into two classes based on whether or not they provide a functional mapping to transform the data from the high-dimensional space to the low-dimesional space in addition to the low-dimensional embedding of the input data. These mappings are useful for re-useability, and also provide a way to interpret *how* the original features of the data relate to the low-dimensional embedding features. The canonical state-of-the-art methods UMAP and t-SNE are, however, both non-mapping.

Autoencoders are a class of unsupervised learning models for learning representations of data, by simultaneously learning a function to encode the data in a low-dimensional space (the encoder), and a function to reconstruct the input data from the encoding (the decoder). Conventional autoencoders use artificial neural networks (ANN), which have an opaque structure that makes interpretation of the mappings highly difficult.

In this work, We have explored the backgrounds of evolutionary computation (EC), genetic programming (GP), dimensionality reduction, neural networks and autoencoders. We have assessed existing work that has attempted to use GP for autoencoding, motivated by its interpretable structure. We have also reviewed other EC approaches to dimensionality reduction.

Through this review, we identified a gap in the existing research. Existing work has attempted to replace the entire autoencoder with GP, which we can group into two approaches. The first have attempted to represent both the encoder and decoder independently with GP, however this has been difficult due to the stochastic EC approach meaning independent changes in one can have significant impacts on the performance of the other. Other approaches have forgone the encoder-decoder architecture entirely while using GP to mimic the reconstructive behaviour of an autoencoder directly. These approaches have used complex and indirect GP representations, which makes interpretation difficult. We have argued that for the sake of interpretable dimensionality reduction, it is only the encoder where interpretability if valuable, as the decoder can be seen as simply serving to evaluate embeddings produced by the encoder.

To address this gap, we have proposed the Genetic Programming Encoder for Autoencoding (GPE-AE). GPE-AE retains the ANN decoder, while using a multi-tree representation for the encoder. This allows for an interpretable encoding structure, while still retaining the performance benefits of the ANN decoder.

We have presented the results of experiments to compare GPE-AE to both conventional autoencoders (CAE) and GP-MaL, a similar GP method for non-linear dimensionality re-

duction. We found that GPE-AE was competitive with both approaches for producing embeddings which retained the original structure, demonstrating the strength of the approach at finding functional dimensionality reductions. We identified some differences in performance when compared to a conventional autoencoder, which we have argued may be due to the two very different optimisation strategies. We also have compared two-dimensional visualisations produced by the methods, and assessed how the different approaches can effect these. Finally, we have analysed some selected GP encoders produced by GPE-AE to demonstrate the valuable insights that can be gained by using interpretable AE models.

6.1 Future Work

As this work represents an initial exploration into the combined GP with ANN structure of autoencoding, there are many directions future work could take.

In this work, we keep the structure of the ANN decoder simple and constant for all individuals during the evolution. However, future work could explore dynamic decoder structures, that are based on the encoder structure. For example, simple encoders could make use of simpler decoders, perhaps reducing evolution time by reducing decoder training time for simpler solutions where complex decoders may not be required.

Another potential direction is the extension of this work to Variational Autoencoders. The suitability of GP for multi-objective optimisation would make it trivial to introduce another objective to constrain the shape of the latent distribution.

Bibliography

[1] ALBAWI, S., MOHAMMED, T. A., AND AL-ZAWI, S. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)* (2017), Ieee, pp. 1–6.

[2] ARULAMPALAM, G., AND BOUZERDOUM, A. A generalized feedforward neural network architecture for classification and regression. *Neural Networks 16*, 5-6 (2003), 561–568.

[3] ASSUNÇÃO, F., SERENO, D., LOURENÇO, N., MACHADO, P., AND RIBEIRO, B. Automatic evolution of autoencoders for compressed representations. In *2018 IEEE Congress on Evolutionary Computation, CEC 2018, Rio de Janeiro, Brazil, July 8-13, 2018* (2018), IEEE, pp. 1–8.

[4] BENGIO, Y., COURVILLE, A. C., AND VINCENT, P. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell. 35*, 8 (2013), 1798–1828.

[5] CHAI, T., AND DRAXLER, R. R. Root mean square error (rmse) or mean absolute error (mae). *Geoscientific Model Development Discussions 7*, 1 (2014), 1525–1534.

[6] CIESIELSKI, V. Linear genetic programming. *Genet. Program. Evolvable Mach. 9*, 1 (2008), 105–106.

[7] HINNEBURG, A., AND KEIM, D. A. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases* (1999), pp. 506–517.

[8] HINTON, G. E., AND ROWEIS, S. T. Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]* (2002), S. Becker, S. Thrun, and K. Obermayer, Eds., MIT Press, pp. 833–840.

[9] HINTON, G. E., AND SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *Science 313*, 5786 (2006), 504–507.

[10] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. 1992.

[11] JACKSON, D. A new, node-focused model for genetic programming. In *Genetic Programming* (2012), pp. 49–60.

[12] JOLLIFFE, I. T. Principal component analysis. In *International Encyclopedia of Statistical Science*, M. Lovric, Ed. Springer, 2011, pp. 1094–1096.

[13] KASHEF, S., AND NEZAMABADI-POUR, H. An advanced ACO algorithm for feature subset selection. *Neurocomputing 147* (2015), 271–279.

[14] KINGMA, D. P., AND BA, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015), Y. Bengio and Y. LeCun, Eds.

[15] KINGMA, D. P., AND WELLING, M. An introduction to variational autoencoders. *Found. Trends Mach. Learn. 12*, 4 (2019), 307–392.

[16] LANDER, S., AND SHANG, Y. Evoae - A new evolutionary method for training autoencoders for deep learning networks. In *39th IEEE Annual Computer Software and Applications Conference, COMPSAC 2015, Taichung, Taiwan, July 1-5, 2015. Volume 2* (2015), IEEE Computer Society, pp. 790–795.

[17] LAROCHELLE, H., BENGIO, Y., LOURADOUR, J., AND LAMBLIN, P. Exploring strategies for training deep neural networks. *J. Mach. Learn. Res. 10* (2009), 1–40.

[18] LEARDI, R., BOGGIA, R., AND TERRILE, M. Genetic algorithms as a strategy for feature selection. *Journal of chemometrics 6*, 5 (1992), 267–281.

[19] LENSEN, A., XUE, B., AND ZHANG, M. Improving *k*-means clustering with genetic programming for feature construction. In *Genetic and Evolutionary Computation Conference, Berlin, Germany, July 15-19, 2017, Companion Material Proceedings* (2017), P. A. N. Bosman, Ed., ACM, pp. 237–238.

[20] LENSEN, A., XUE, B., AND ZHANG, M. Can genetic programming do manifold learning too? In *Genetic Programming - 22nd European Conference, EuroGP 2019, Held as Part of EvoStar 2019, Leipzig, Germany, April 24-26, 2019, Proceedings* (2019), vol. 11451 of *Lecture Notes in Computer Science*, Springer, pp. 114–130.

[21] LENSEN, A., ZHANG, M., AND XUE, B. Multi-objective genetic programming for manifold learning: balancing quality and dimensionality. *Genet. Program. Evolvable Mach. 21*, 3 (2020), 399–431.

[22] LIU, H., AND MOTODA, H. *Feature Selection for Knowledge Discovery and Data Mining*, vol. 454 of *The Springer International Series in Engineering and Computer Science*. Kluwer, 1998.

[23] LÓPEZ, E. G. Efficient graph-based genetic programming representation with multiple outputs. *Int. J. Autom. Comput. 5*, 1 (2008), 81–89.

[24] MCDERMOTT, J. Why is auto-encoding difficult for genetic programming? In *Genetic Programming - 22nd European Conference, EuroGP 2019, Held as Part of EvoStar 2019, Leipzig, Germany, April 24-26, 2019, Proceedings* (2019), L. Sekanina, T. Hu, N. Lourenço, H. Richter, and P. García-Sánchez, Eds., vol. 11451 of *Lecture Notes in Computer Science*, Springer, pp. 131–145.

[25] MCINNES, L., AND HEALY, J. UMAP: uniform manifold approximation and projection for dimension reduction. *CoRR abs/1802.03426* (2018).

[26] MILLER, J. F. *Cartesian Genetic Programming*. Springer Berlin Heidelberg, 2011.

[27] MITCHELL, T. M. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997.

[28] MONTANA, D. J. Strongly typed genetic programming. *Evolutionary Computation 3*, 2 (Summer 1995), 199–230.

[29] MONTAVON, G., SAMEK, W., AND MÜLLER, K. Methods for interpreting and understanding deep neural networks. *Digit. Signal Process. 73* (2018), 1–15.

[30] ORZECHOWSKI, P., MAGIERA, F., AND MOORE, J. H. Benchmarking manifold learning methods on a large collection of datasets. In *Genetic Programming - 23rd European Conference, EuroGP 2020, Held as Part of EvoStar 2020, Seville, Spain, April 15-17, 2020, Proceedings* (2020), vol. 12101 of *Lecture Notes in Computer Science*, Springer, pp. 135–150.

[31] POLI, R., LANGDON, W. B., AND MCPHEE, N. F. *A Field Guide to Genetic Programming*. lulu.com, 2008.

[32] POTTER, M. A., AND JONG, K. A. D. A cooperative coevolutionary approach to function optimization. In *Parallel Problem Solving from Nature - PPSN III, International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature, Jerusalem, Israel, October 9-14, 1994, Proceedings* (1994), vol. 866 of *Lecture Notes in Computer Science*, Springer, pp. 249–257.

[33] REEVES, C. R. Genetic algorithms. In *Encyclopedia of Database Systems, Second Edition*, L. Liu and M. T. Özsu, Eds. Springer, 2018.

[34] RODRIGUEZ-COAYAHUITL, L., MORALES-REYES, A., AND ESCALANTE, H. J. Evolving autoencoding structures through genetic programming. *Genet. Program. Evolvable Mach. 20*, 3 (2019), 413–440.

[35] RUBERTO, S., TERRAGNI, V., AND MOORE, J. H. Image feature learning with genetic programming. In *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part II* (2020), vol. 12270 of *Lecture Notes in Computer Science*, Springer, pp. 63–78.

[36] SAINBURG, T., MCINNES, L., AND GENTNER, T. Q. Parametric UMAP embeddings for representation and semisupervised learning. *Neural Comput. 33*, 11 (2021), 2881–2907.

[37] SCHOFIELD, F., AND LENSEN, A. Using genetic programming to find functional mappings for UMAP embeddings. In *IEEE Congress on Evolutionary Computation, CEC 2021, Kraków, Poland, June 28 - July 1, 2021* (2021), IEEE, pp. 704–711.

[38] SHADBOLT, J. Overfitting, generalisation and regularisation. In *Neural Networks and the Financial Markets - Predicting, Combining and Portfolio Optimisation*, J. Shadbolt and J. G. Taylor, Eds., Perspectives in Neural Computing. Springer, 2002, pp. 55–59.

[39] SHIRKHORSHIDI, A. S., AGHABOZORGI, S., AND WAH, T. Y. A comparison study on similarity and dissimilarity measures in clustering continuous data. *PLOS ONE* (2015).

[40] SIEGELMANN, H. T., DASGUPTA, B., AND LIU, D. Neural networks. In *Handbook of Approximation Algorithms and Metaheuristics*, T. F. Gonzalez, Ed. Chapman and Hall/CRC, 2007.

[41] SMOLINSKI, T. G. Multi-objective evolutionary algorithms. In *Encyclopedia of Computational Neuroscience*, D. Jaeger and R. Jung, Eds. Springer, 2014.

[42] SONDHI, P. Feature construction methods: a survey. Tech. rep., 2009.

[43] SOULE, T., AND FOSTER, J. A. Effects of code growth and parsimony pressure on populations in genetic programming. *Evol. Comput. 6*, 4 (1998), 293–309.

[44] SVETNIK, V., LIAW, A., TONG, C., CULBERSON, J. C., SHERIDAN, R. P., AND FEUSTON, B. P. Random forest: a classification and regression tool for compound classification and qsar modeling. *Journal of chemical information and computer sciences 43*, 6 (2003), 1947–1958.

[45] TALAVERA, L. An evaluation of filter and wrapper methods for feature selection in categorical clustering. In *Advances in Intelligent Data Analysis VI, 6th International Symposium on Intelligent Data Analysis, IDA 2005, Madrid, Spain, September 8-10, 2005, Proceedings* (2005), A. F. Famili, J. N. Kok, J. M. P. Sánchez, A. Siebes, and A. J. Feelders, Eds., vol. 3646 of *Lecture Notes in Computer Science*, Springer, pp. 440–451.

[46] TRAN, B., XUE, B., AND ZHANG, M. Genetic programming for feature construction and selection in classification on high-dimensional data. *Memetic Comput. 8*, 1 (2016), 3–15.

[47] VAN DER MAATEN, L. Learning a parametric embedding by preserving local structure. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009* (2009), vol. 5 of *JMLR Proceedings*, JMLR.org, pp. 384–391.

[48] VIKHAR, P. A. Evolutionary algorithms: A critical review and its future prospects. In *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)* (2016), pp. 261–265.

[49] WINEBERG, M., AND OPPACHER, F. The benefits of computing with introns. In *Proceedings of the 1st Annual Conference on Genetic Programming* (Cambridge, MA, USA, 1996), MIT Press.

[50] WITTEK, P. 5 - unsupervised learning. In *Quantum Machine Learning*, P. Wittek, Ed. Academic Press, Boston, 2014, pp. 57–62.

[51] XUE, B., ZHANG, M., AND BROWNE, W. N. Multi-objective particle swarm optimisation (PSO) for feature selection. In *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012* (2012), ACM, pp. 81–88.