A Genetic Programming Encoder for Increasing Autoencoder Interpretability

Finn Schofield[1], Luis Slyfield, and Andrew Lensen[1 [0000−0003−1269−4751]]

School of Engineering and Computer Science,
Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand
{schofifinn, slyfieluis, andrew.lensen}@ecs.vuw.ac.nz

Abstract. Autoencoders are powerful models for non-linear dimensionality reduction. However, their neural network structure makes it difficult to interpret how the high dimensional features relate to the low-dimensional embedding, which is an issue in applications where explainability is important. There have been attempts to replace both the neural network components in autoencoders with interpretable genetic programming (GP) models. However, for the purposes of interpretable dimensionality reduction, we observe that replacing only the encoder with GP 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 a GP based method that does not use an autoencoder structure. As visualisation is a common goal for dimensionality reduction, we also evaluate the quality of visualisations produced by our method, and highlight the value of functional mappings by demonstrating insights that can be gained from interpreting the GP encoders.

Keywords: Genetic programming · Autoencoder · Dimensionality reduction · Machine learning · Explainable artificial intelligence

1 Introduction

Dimensionality reduction (DR) involves taking high-dimensional data and producing a representation of it in a space with considerably less dimensions. This is beneficial both as a pre-processing step to improve the effectiveness or tractability of using the data for machine learning, and to gain a better understanding of the data during exploratory data analysis (EDA), for example visualising the data in two or three dimensions. Non-linear dimensionality reduction (NLDR) [11] is of particular interest for data with more complicated relationships that can only be expressed by non-linear relationships. Many existing NLDR methods are able to produce low-dimensional representations of complex datasets while retaining much of the original information [6, 17]. However the best performing methods have two main drawbacks: they only produce the new data points in low-dimensional space without a means to process new points without retraining, and the way in which they produce the embedding is difficult for a human to directly check and interpret.

Having interpretable models is important for a variety of reasons. One reason is for tasks with human impact such as in medical settings, where there is an ethical obligation to explain any decisions made. Another is as governments move to legislate around the use of artificial intelligence, there is increasing legal obligation for model interpretability. In the case of NLDR, an interpretable model means we can build a clearer picture of how the data in the constructed lower dimensional space relate to the real data in the original space, and as a result have more confidence using the method in settings where this is important. This also makes NLDR a more effective tool for EDA, as we can use both the low dimensional representation and the transformation itself to understand more about the data and the relationships within it.

An autoencoder (AE) is a type of artificial neural network architecture that performs NLDR [5]. It has two main components: an encoder that transforms from the space containing the original data into a space with considerably less dimensions, and a decoder that transforms from the low dimensional space back to the higher one. An AE is an appealing method to use for NLDR because its neural network architecture is capable of representing powerful relationships, it produces a functional mapping that makes it possible to easily transform previously unseen data points into the latent space, and its ability to reconstruct the original data is an appealing metric of success at the DR task. However to achieve high performance, AE architectures usually have far too many parameters to be interpretable without using external methods to provide an explanation.

Genetic programming (GP) is an evolutionary computation (EC) technique where computer programs are evolved over generations [2, 18]. GP has inherent potential for interpretability, because it evolves solutions that combine user-selected terminals and functions. GP has recently been demonstrated to be a capable NLDR technique which produces functional mappings [12, 14]. Some of these approaches have used a multi-tree GP representation with a custom fitness function for evaluating embedding quality [12–14, 22, 24], and other work has looked into GP specifically for autoencoding [16, 19].

Combining GP and autoencoding is an appealing concept: it has the potential to benefit both from the power of AEs and the inherent interpretability of GP. Existing research into GP for autoencoding has attempted to evolve both the encoder and decoder structure using GP, but has struggled due to the strong dependency between the encoder and decoder, limiting performance [16]. Other attempts instead forgo the architecture, instead using a fully tree-based approach which gives representations that are complex and difficult to interpret [19].

In this work we propose an AE-based approach to NLDR that replaces the encoder component with a multi-tree structure trained by GP but keeps the neural network decoder. We suggest that due to demonstrated suitability of multi-tree GP to NLDR 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 but keeping the benefit of the AE training method and its potentially powerful neural network decoder.

The contributions of this work are summarised by the research goals:

- Investigate existing work relevant to GP for NLDR and autoencoding;
- Propose a novel autoencoding method that replaces the encoder with a multi-tree GP individual while retaining the ANN decoder;
- Evaluate how the method compares at the task of NLDR to conventional AEs and GP-based NLDR;
- Compare how the method performs at visualisation to the baselines; and
- Investigate what potential the method has for interpretability.

1.1 Structure

The rest of the paper is structured as follows. Section 2 discusses related work to provide a context for the method we present. Section 3 outlines our proposed method for creating an autoencoder with a genetic programming encoder. Section 4 describes the experiment design used to test the method, and Section 5 the results of running these tests. Further analysis including visualisation examples are presented in Section 6, and the paper is concluded in Section 7.

2 Background and Related Work

2.1 Non-linear Dimensionality Reduction

While traditional dimensionality reduction techniques such as PCA [7] can be sufficient for producing high-quality embeddings of data, 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 [1].

NLDR techniques can be divided into two classes: mapping and non-mapping. *Mapping* techniques are those which produce the data embeddings in the low-dimensional space, as well as a functional mapping to produce them from the high-dimensional space. *Non-mapping* techniques on the other hand provide only the low-dimensional 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) [6]. 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. Lastly 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) [17] follows a similar process to t-SNE. However, instead of using probability distributions and minimising the KL divergence, UMAP uses fuzzy graph representations of the data in the high dimensional and low dimensional space.

Both t-SNE and UMAP are non-mapping. There have been parametric variations proposed for both that use neural networks to allow for reusable mappings, although they are still extremely complex and difficult to interpret [15, 21].

2.2 Evolutionary Computation for Dimensionality Reduction

Various EC techniques have been applied to DR. The most straightforward DR task, feature selection, is simply isolating the most important features. This has been approached by a range of EC techniques, such as bit-string genetic algorithms [10], particle swarm optimisation (PSO) [26], and ant colony optimisation (ACO) [8].

The more complex problem of feature construction involves creating a reduced number of new features using the original features as components for transformation. For this task, the programmatic 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. The functional structure of GP trees lends itself to produce not just mappings which are reusable, but also ones that are interpretable.

Genetic programming has been proposed as a potential approach to learn functional mappings for non-linear dimensionality reduction, such as in *Genetic Programming for Manifold Learning* (GP-MaL), which proposed the use of GP for NLDR [12] without the use of an AE architecture. GP-MaL uses a multi tree representation, with w trees to represent w dimensions of the embedding. The fitness function used by GP-MaL is based on the preservation of orderings of neighbours from the original feature space to the embedding. The GP-MaL fitness is somewhat ad-hoc: it uses a particular formulation of neighbour preservation. Later work has proposed other fitness functions, such as that used by the state-of-the-art non-mapping UMAP [17] method [22].

2.3 Genetic Programming for Autoencoding

There are some existing methods that incorporate GP with an AE framework.

The *Genetic Programming Autoencoder* (GPAE) replaced the AE entirely with a linear GP representation [16]. Each individual is comprised of two linear GP programs that represent an encoder and a decoder. Instead of a population-based search, GPAE mutates a single individual using hill-climbing. The multi-tree approach is ruled out due to the inability to share calculations between trees. In fact, given that it is desirable to have minimal redundancy (shared information) between the embedding dimensions, sharing calculations may be a *downside* of GPAE.

Structurally Layered Genetic Programming (SLGP) [19] uses a dual-forest representation, with w trees to construct the embedding and v trees to reconstruct the original input. Representing the decoder as a forest requires a tree for each original feature, which is difficult to train on high-dimensional data. To avoid this, SLGP decomposes the problem to smaller, independent GP runs, each of which considers a subset of features. Thus, the mapping can only take into account combinations of original features which are in the same subset. Learning a tree for each original dimension is also very expensive.

Genetic Programming for Feature Learning (GPFL) is an AE-like approach to feature learning using GP [20]. Feature learning is a tangential task to NLDR that involves learning representations of image data. While GPFL uses a multi-tree representation, it learns the trees sequentially, with each subsequent tree correcting earlier errors. The final individual is a linear combination of the trees. One drawback of GPFL is the 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.

3 Proposed Method: GPE-AE

After considering the existing work on the subject discussed above, we developed the novel Genetic Programming Encoder for Autoencoding (GPE-AE) method. The overall design of GPE-AE is presented in Fig. 1. Here, an example of learning a 3-dimension embedding of n features is used. Taking the original input dataset with features f, these are used as 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.

3.1 GP Representation of Encoder

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. This is consistent with previous work that showed success with a multi-tree approach [12]. While other work has argued against a multi-tree representation for autoencoding due to its inability to share calculations between trees [16], 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.

Our GP encoder representation uses 12 functions, as shown in Table 1. In addition to standard arithmetic operators ($+$, $-$, $\times$), we utilise: absolute addition ($|+|$) and subtraction ($|-|$), an addition function which takes 5 inputs ($5+$), and protected division ($\%$) which returns 1 when the denominator is zero. Absolute

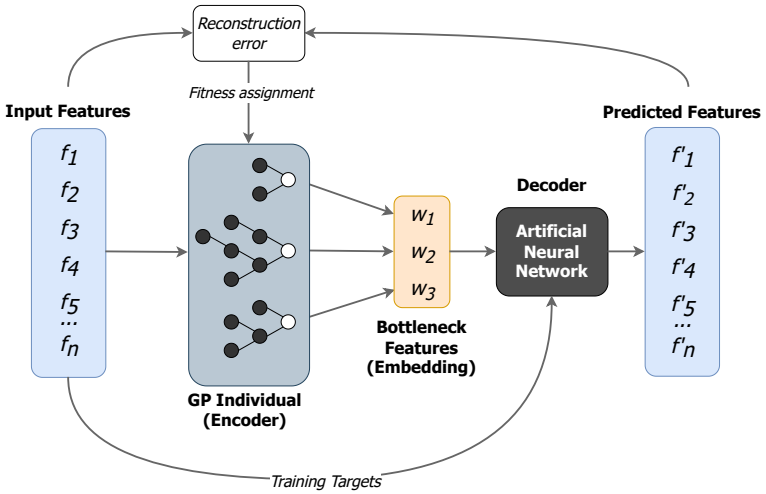


Fig. 1: An overview of GPE-AE, with n features reduced to a 3D embedding.

Table 1: Functions used by GPE-AE. All functions take/produce numeric values.

Category	Arithmetic							Logical			Non-Linear	
Function	+	5+	−	|+|	|−|	×	%	max	min	if	ReLU	sigmoid
No. Inputs	2	5	2	2	2	2	2	2	2	3	1	1

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. Three logical operators are also included: max, min, and if. if takes three inputs — if the first input is greater than zero, it outputs the second input; otherwise it outputs the third. These allow for more expressive use of the original features beyond arithmetic combinations. To allow for further non-linear transformations, the ReLU and sigmoid functions are used. 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 [12, 14].

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

Multi-tree GP requires adapting traditional tree-based GP crossover and mutation. In this work, we use *All Index Crossover (AIC)*, where standard crossover is performed on all pairs of trees (with the same index within the multi-tree representation) across each parent. For mutation, we perform standard GP mutation on a randomly selected tree.

3.2 Fitness Evaluation

The fitness of GPE-AE individual I with w trees occurs is evaluated 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 in standard AEs, the objective/fitness function is the reconstruction error between the inputs and the predicted outputs. We use root mean squared error (RMSE), as it better penalises large errors [3]. RMSE is defined as:

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

for n instances, where $\mathbf{x}_i$ is the i^{th} instance of the input, and $\mathbf{x}'_i$ is the predicted value of instance i (after encoding and decoding).

3.3 Decoder Architecture

The architecture of the ANN decoder requires special consideration. In a conventional AE, it is common 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. 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.

In GPE-AE, the role of the ANN decoder is only to evaluate the performance of the GP encoder, and such a decoder needs to be trained for each evaluation (of which there can be many). Therefore, we do not need the decoder to be *perfect*, but merely to be able to reliably distinguish the performance of encoder candidates in a consistent way, and training needs to be relatively short to keep the overall GPE-AE running time computationally feasible. As such, we suggest the use of simpler decoder architectures. This also has the benefit of pressuring the evolution process towards better-structured embeddings: if a reasonably simple decoder cannot sufficiently reconstruct the input, then the embedding is likely ill-formed.

As ANN training is stochastic, we use a hash of the encoder as the seed for training the decoder, ensuring a GP individual always has the same fitness.

4 Experiment Design

To examine the role the complexity of the decoder plays in the performance of GPE-AE, we perform experiments with decoders with either 1, 2, or 3 hidden layers. The number of neurons at each layer is presented in Table 2. We follow the common "funnel" architecture of AEs, where the decoder incrementally expands the data from the bottleneck to the final reconstruction. As the depth of the

Table 2: The NN architectures used by GPE-AE and CAE in the experiments. GPE-AE only makes use of the decoder.

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

Table 3: GP parameters used for GPE-AE and GP-MaL.

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

decoder increases, so does the number of connections, allowing for more complex decoding structures. However, introducing more connections has a significant impact on the computational cost of evaluating fitness.

We are also interested in how GPE-AE performs on NLDR tasks of varying difficulty. To evaluate this, we perform experiments across a range of embedding sizes (1, 2, 3, 5, and 10), which represent decreasing levels of challenge.

Standard GP parameters used by GPE-AE for all experiments are shown in Table 3. For the decoder, standard neural network parameters are used, as follows. A limit of 100 epochs is used to reduce computational cost, as a NN is required to be trained for each fitness evaluation. As the decoders are reasonably simple, this was considered sufficient. The ReLU activation is used to add non-linearity between hidden layers (as in GPE-AE). The Adam optimiser was used as it performs well in similar problems [9]. A learning rate of 0.001 and mini-batch size of 200 were found to be sufficient in exploratory testing.

For each dataset, embedding size, and number of hidden layers, we perform 30 runs using GPE-AE and both of the comparison methods stated in Section 4.1. This accounts for the stochastic nature of the evolutionary process.

4.1 Comparison Methods

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

Conventional Auto-Encoder: we perform experiments using the same hidden layer configurations as GPE-AE. The architecture of the encoder mirrors the decoder, as shown in Table 2. By mirroring the decoder, we can be sure that any structure capable of being found by the encoder 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, and with the same MSE objective function.

GP-MaL: Our motivation behind GPE-AE is to use the AE structure to produce functional mappings for NLDR with GP. Thus, it is also valuable to

compare it to another multi-tree GP NLDR method, such as GP-MaL. This allows us to test the hybrid GPE-AE method against both a pure CAE method and a pure multi-tree GP method. Our GP-MaL experiments use the same parameters and terminal/function sets as in GPE-AE.

4.2 Evaluation Measures

As NLDR and autoencoding are unsupervised tasks, there is no "gold-standard" objective measure to compare different methods. For GPE-AE and the CAE, we can directly compare their reconstruction error. However, GP-MaL does not perform reconstruction. While we could use the GP-MaL fitness function to compare all methods, this would then favour GP-MaL, which used it as the optimisation criterion. To avoid this, we propose using the classification accuracy obtained using the low-dimension embedding. This approach has been used in previous GP for NLDR work [12,14], and relies 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 important structure within the data has been retained in the embedding.

As this is unsupervised learning, the data labels are not given to GPE-AE or our comparison methods, eliminating bias towards any particular approach. The classification algorithm used for evaluation (after the embeddings have been learned) is the scikit-learn Random forest implementation, using 100 trees. Random forest is an efficient and robust algorithm, making it a good choice for unbiased evaluation [23]. We calculate the classification accuracy using 10-fold-cross-validation on the low dimensional embedding.

Measuring Complexity To measure the complexity of the models, we can calculate the number of *connections* that the GP trees and the ANN encoder have. As both are directed graphs, we are counting the number of edges in each. For a GP individual, this is $|nodes| - w$, for w roots (trees), as each node except the root have a single parent. For a fully-connected neural network, this is defined by the equation $\sum_{i \in L}^{L-1} L_i L_{i+1}$, where L is the number of layers in the network, and L_i is the number of nodes as layer i. The number of connections approximately represents the complexity or uninterpretability of a model.

4.3 Datasets

The datasets used are presented in Table 4. Clean1 is from openML [25], while the rest are from the UCI Repository [4]. The selected datasets have a range of different dimensionalities, classes and instances to evaluate the performance of GPE-AE across different problems.

Table 4: Datasets used for testing.

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

5 Results

We first compare the three methods (GPE-AE, CAE, and GP-MaL) in terms of their classification accuracy and complexity (number of connections). These results are shown in Table 5. The results are grouped vertically by the dimensionality of the embedding. For GPE-AE and the CAE, there are three rows per dimensionality, for the three different configurations of hidden layers. 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 both the decoder and encoder. The number of neurons at each layer was presented in Table 2. GP-MaL does not use a decoder architecture, and so has only one row per dimensionality.

Classification Accuracy: The mean classification across the 30 runs for each method on each dataset is presented in the Accuracy column. A Wilcoxon significance test was performed with a p-value of 0.05. The tests were performed using each configuration of hidden layers, with the GPE-AE and CAE methods being tested for each configuration and dataset. A "+" next to a GPE-AE accuracy indicates that GPE-AE significantly outperformed CAE with the same hidden layer configuration on a dataset; a "−" indicates GPE-AE performed significantly worse. GPE-AE was also compared to GP-MaL, with a ↑ indicating that GPE-AE significantly outperformed GP-MaL, and a ↓ indicating the opposite.

From our results, GPE-AE was generally better than CAE for the "easier" datasets with 34 or fewer features. This indicates that the GP approach is able to find embeddings with a more separable structure of classes. On the harder problem of the Clean1 dataset with 168 features, the CAE generally outperformed GPE-AE — albeit with a relatively small magnitude of difference. Clean1 has two classes, split 54%:46% — giving a "baseline" accuracy of 54%. All the methods are very close to this when reducing to one dimension, indicating the classifier is only doing slightly better than randomly assigning labels. This is not surprising, as reducing 168 features to a single dimension is inherently a difficult task assuming most of the features are not irrelevant or redundant. GP-MaL often had the highest performance of the three methods, but was only significantly better than GPE-AE in some tests on the Dermatology and Segmentation datasets. Neither of the AE methods show clear trends across different hidden layer configurations. All three methods show clear improvements as embedding dimensionality increases; more dimensions allows for more structure to be retained.

Number of Connections: For GPE-AE and GP-MaL, the average number of connections of the best individuals found across the 30 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. To highlight the effect the complexity of the decoder has on the complexity of the encoder, the smallest average individual size for each test configuration is shown in bold. Even though the CAE designs we propose are fairly simple, they still have a very large number of connections compared to GP, especially on the Clean1 dataset.

Table 5: GPE-AE compared to conventional auto-encoders (CAE) and GP-MaL.

	Clean1		**Derma.**		**Iono.**		**Segmen.**		**Wine**	
Method	Acc.	Conn.	Acc.	Conn.	Acc.	Conn.	Acc.	Conn.	Acc.	Conn.
1 Dimension										
GPE 1HL	0.527−	182	0.813+ ↓	**159**	0.850+	**189**	0.653+	**141**	0.914+	216
GPE 2HL	0.544−	302	0.803+ ↓	210	0.871+	237	0.663+	214	0.908	**178**
GPE 3HL	0.546−	132	0.784↓	219	0.865+	192	0.641+	201	0.912	181
CAE 1HL	0.553	21632	0.719	4480	0.714	4480	0.489	2560	0.767	1792
CAE 2HL	0.571	32320	0.712	6592	0.724	6592	0.594	3712	0.811	2560
CAE 3HL	0.581	34336	0.746	8608	0.725	8608	0.561	5728	0.793	4576
GP-MaL	0.582	**128**	0.915	371	0.868	211	0.649	217	0.883	217
2 Dimensions										
GPE 1HL	0.600	374	0.891↓	324	0.883+	273	0.701+	288	0.942	470
GPE 2HL	0.622	276	0.894+ ↓	301	0.886+	244	0.754+	**262**	0.933	342
GPE 3HL	0.582−	**254**	0.872+ ↓	**267**	0.890+	238	0.707+	392	0.916+	352
CAE 1HL	0.591	21760	0.874	4608	0.817	4608	0.573	2688	0.872	1920
CAE 2HL	0.641	32384	0.855	6656	0.823	6656	0.619	3776	0.909	2624
CAE 3HL	0.645	34368	0.852	8640	0.81	8640	0.605	5760	0.911	4608
GP-MaL	0.621	348	0.935	301	0.889	**227**	0.714	325	0.937	**267**
3 Dimensions										
GPE 1HL	0.639−	257	0.909	371	0.896+	326	0.799+ ↓	336	0.938	380
GPE 2HL	0.642−	328	0.909+	425	0.897+	342	0.779+ ↓	299	0.938	**367**
GPE 3HL	0.628−	**106**	0.911+	**305**	0.893+	**277**	0.786+ ↓	**257**	0.940	518
CAE 1HL	0.665	21888	0.910	4736	0.856	4736	0.676	2816	0.928	2048
CAE 2HL	0.676	32448	0.886	6720	0.862	6720	0.709	3840	0.937	2688
CAE 3HL	0.665	34400	0.865	8672	0.860	8672	0.713	5792	0.940	4640
GP-MaL	0.639	301	0.928	450	0.899	348	0.830	320	0.95	419
5 Dimensions										
GPE 1HL	0.692−	323	0.918−	451	0.906	**242**	0.889+	549	0.944+	588
GPE 2HL	0.697−	**242**	0.922+	436	0.903	431	0.888+	**327**	0.931	385
GPE 3HL	0.668−	431	0.897+ ↓	**346**	0.904+	297	0.873+	338	0.936+	547
CAE 1HL	0.707	22144	0.940	4992	0.900	4992	0.806	3072	0.934	2304
CAE 2HL	0.739	32576	0.911	6848	0.895	6848	0.812	3968	0.932	2816
CAE 3HL	0.707	34464	0.882	8736	0.885	8736	0.767	5856	0.927	4704
GP-MaL	0.689	480	0.953	492	0.911	428	0.895	595	0.947	**295**
10 Dimensions										
GPE 1HL	0.746−	410	0.949	808	0.906+	387	0.928+	673	0.956+	604
GPE 2HL	0.747−	**356**	0.933↓	772	0.905−	**378**	0.916+ ↓	**384**	0.964+	832
GPE 3HL	0.717−	491	0.943+ ↓	**452**	0.912−	562	0.911+ ↓	559	0.949+	655
CAE 1HL	0.769	22784	0.946	5632	0.920	5632	0.870	3712	0.938	2944
CAE 2HL	0.775	32896	0.930	7168	0.915	7168	0.856	4288	0.935	3136
CAE 3HL	0.744	34624	0.910	8896	0.905	8896	0.821	6016	0.924	4864
GP-MaL	0.755	577	0.963	754	0.907	457	0.932	839	0.963	**556**

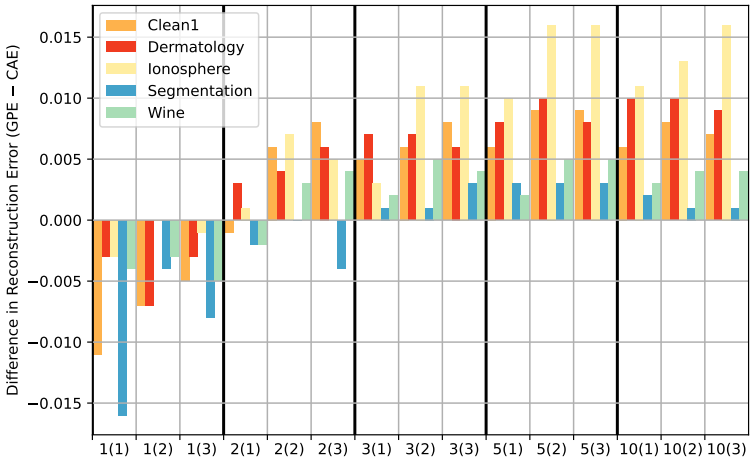


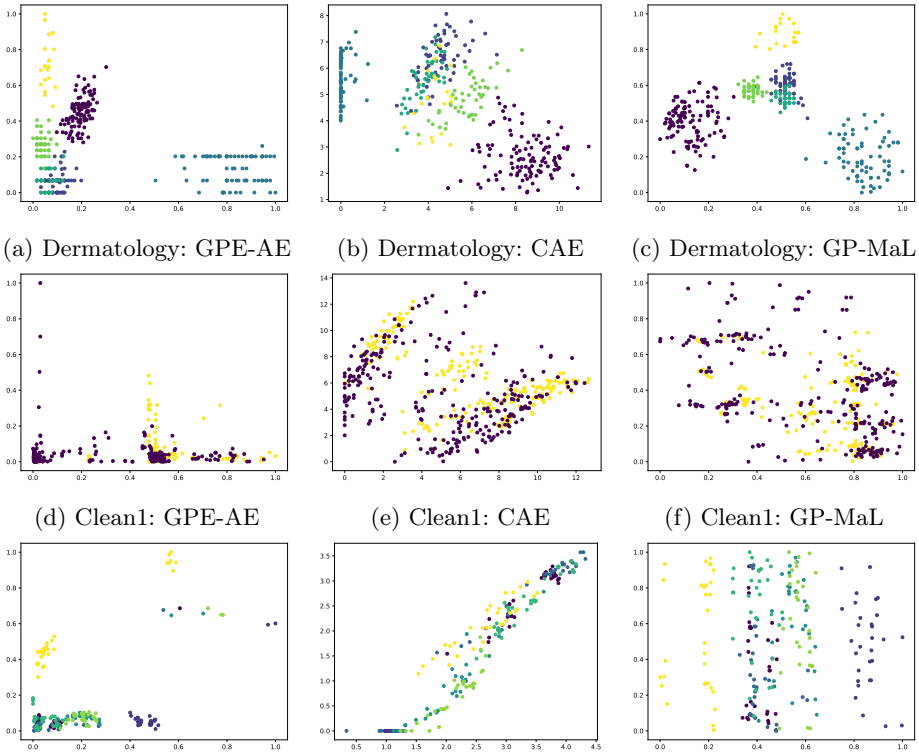
Fig. 2: The differences between the reconstruction error achieved by GPE-AE and CAE. A -ve value means GPE-AE had a lower reconstruction error than CAE. 1(3) represents a dimensionality of 1 with 3 hidden layers in the decoder.

Reconstruction Error: The mean difference in reconstruction error between GPE-AE and the CAE is shown in Fig. 2 for each dataset and hidden layer configuration. The x-axis represents increasing embedding dimensionality from left to right, with the three different hidden layers configurations shown at each embedding dimensionality. A negative difference indicates that GPE-AE achieved better reconstruction error than CAE. There are two clear trends: firstly, that the two methods are very similar in their reconstruction errors (differing by at most ~ 0.016); and secondly that GPE-AE has lower reconstruction error on the most challenging task of embedding in one dimension, while CAE has lower errors at higher dimensions. This is encouraging for GPE-AE: with many fewer connections, it can achieve a lower reconstruction error with a one-dimensional embedding. It does, however, demonstrate the need for further research into improving the performance of multi-tree GP at higher embedding sizes.

6 Further Analysis

A common NLDR task is the reduction of data to two dimensions, explicitly for the sake of visualising the data. To evaluate the suitability of GPE-AE for visualisation, we show representative two-dimensional embeddings produced by the three methods in Fig. 3 on Dermatology, Clean1 and Segmentation datasets. These datasets represent a variety of difficulty in terms of number of original features, from 19 for Segmentation to 168 features for Clean1.

On Dermatology, the GPE-AE (and, to a lesser degree, GP-MaL) visualisation is that instances seem to be grouped in rigid “steps" along the y axis. The CAE learns “smoother" mapping functions due to the large number of connections, whereas the GP methods are able to find lower-complexity trees that are sufficient in terms of fitness. For Clean1, none of the methods are able to clearly



(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

Fig. 3: Visualisations produced by the GP methods and a conventional autoencoder (CAE) on the Dermatology, Clean1 and Segmentation datasets. The median result of each was chosen for visualisation.

separate the two classes, which is consistent with the earlier results. This suggests that the relationship between the class distribution and the original feature space is quite complex: it cannot be well-represented in only two dimensions. The Segmentation visualisations are quite different to the earlier ones. The CAE represents the data along a single line, suggesting it is not making “full use" of the two dimensions. GP-MaL splits the yellow and purple classes quite clearly along the x-axis, with the y-axis mostly representing intra-class variation. GPE-AE is able to separate the same classes well, but also pushes a number instances far away from the main clusters — these are perhaps outliers in the dataset.

Evolved Program Analysis: a key strength of GPE-AE over a conventional AE is the interpretable tree-based representation. Fig. 4 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

$\sim 35\%$ of features are required to create a sufficiently good embedding. Random forest classification achieved an accuracy of 0.782 using the embedding produced by this individual.

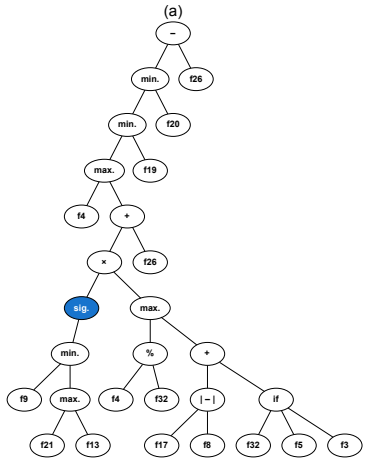


Fig. 4: Dermatology dataset (34 features) reduced to a single feature with GPA-AE using 12 unique features.

7 Conclusions

Autoencoders are a class of unsupervised learning models for learning representations of data. They simultaneously learn 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, which have an opaque structure that makes interpretation of the encoder mappings highly difficult.

Existing work has attempted to replace the entire autoencoder with GP. Some of these represent both the encoder and decoder independently with GP, however these has been difficult to evolve due to the inter-dependency between them. Other approaches have forgone the encoder-decoder architecture entirely by using GP to mimic the reconstructive behaviour of an autoencoder directly. These approaches have used complex and indirect GP representations, which makes interpretation difficult. To address this gap, we proposed the Genetic Programming Encoder for Autoencoding (GPE-AE). GPE-AE retains the ANN decoder, while using a multi-tree GP representation for the encoder. This allows for an interpretable encoding, 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 NLDR. 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 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.

Future Work: In this work, we kept the structure of the ANN decoder simple and constant for all individuals during the evolution. Future work could explore dynamic decoder structures based on the encoder structure. Simple encoders could make use of simpler decoders, reducing evolution time by reducing decoder training time for simpler solutions where complex decoders are not required.

The datasets used in this week were of relatively low dimensionality, with only one dataset over one hundred dimensions. This was suitable for an initial test of GPE-AE as a proof of concept, but future work could test the method on a wider variety of datasets, especially larger ones with over one thousand features. Performing such experiments would provide more rigorous tests of the suitability of GPE-AE for a wider range of tasks.

Another potential extension of this work is to Variational Autoencoders. The suitability of GP for multi-objective optimisation [14,27] would allow introducing another objective that constrains the shape of the latent distribution.

Finally, this work does not directly take into account the susceptibility of GP to program bloat. It is likely that exploring and applying suitable bloat control methods would lead to more compact and therefore interpretable trees in the GP-based encoders produced by the proposed method.

References

1. Bengio, Y., Courville, A.C., Vincent, P.: Representation learning: A review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intell. **35**(8), 1798–1828 (2013). https://doi.org/10.1109/TPAMI.2013.50, https://doi.org/10.1109/TPAMI.2013.50
2. Bi, Y., Xue, B., Zhang, M.: Evolving deep forest with automatic feature extraction for image classification using genetic programming. In: Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN). Lecture Notes in Computer Science, vol. 12269, pp. 3–18. Springer (2020)
3. Chai, T., Draxler, R.R.: Root mean square error (rmse) or mean absolute error (mae). Geoscientific Model Development Discussions **7**(1), 1525–1534 (2014)
4. Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
5. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
6. Hinton, G.E., Roweis, S.T.: Stochastic neighbor embedding. In: Becker, S., Thrun, S., Obermayer, K. (eds.) Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]. pp. 833–840. MIT Press (2002)
7. Jolliffe, I.T.: Principal component analysis. In: Lovric, M. (ed.) International Encyclopedia of Statistical Science, pp. 1094–1096. Springer (2011)
8. Kashef, S., Nezamabadi-pour, H.: An advanced ACO algorithm for feature subset selection. Neurocomputing **147**, 271–279 (2015)
9. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6980
10. Leardi, R., Boggia, R., Terrile, M.: Genetic algorithms as a strategy for feature selection. Journal of chemometrics **6**(5), 267–281 (1992)
11. Lee, J.A., Verleysen, M.: Nonlinear dimensionality reduction, vol. 1. Springer (2007)
12. Lensen, A., Xue, B., 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. Lecture Notes in Computer Science, vol. 11451, pp. 114–130. Springer (2019)
13. Lensen, A., Xue, B., Zhang, M.: Genetic programming for manifold learning: Preserving local topology. IEEE Transactions on Evolutionary Computation pp. 1–15 (2022), early Access
14. Lensen, A., Zhang, M., Xue, B.: Multi-objective genetic programming for manifold learning: balancing quality and dimensionality. Genet. Program. Evolvable Mach. **21**(3), 399–431 (2020)
15. 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. JMLR Proceedings, vol. 5, pp. 384–391. JMLR.org (2009)
16. McDermott, J.: Why is auto-encoding difficult for genetic programming? In: Sekanina, L., Hu, T., Lourenço, N., Richter, H., García-Sánchez, P. (eds.) Genetic Programming - 22nd European Conference, EuroGP 2019, Held as Part of EvoStar 2019, Leipzig, Germany, April 24-26, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11451, pp. 131–145. Springer (2019)
17. McInnes, L., Healy, J.: UMAP: uniform manifold approximation and projection for dimension reduction. CoRR **abs/1802.03426** (2018)
18. Poli, R., Langdon, W.B., McPhee, N.F.: A Field Guide to Genetic Programming. lulu.com (2008), http://www.gp-field-guide.org.uk/
19. Rodriguez-Coayahuitl, L., Morales-Reyes, A., Escalante, H.J.: Evolving autoencoding structures through genetic programming. Genet. Program. Evolvable Mach. **20**(3), 413–440 (2019)
20. Ruberto, S., Terragni, V., 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. Lecture Notes in Computer Science, vol. 12270, pp. 63–78. Springer (2020)
21. Sainburg, T., McInnes, L., Gentner, T.Q.: Parametric UMAP embeddings for representation and semisupervised learning. Neural Comput. **33**(11), 2881–2907 (2021)
22. Schofield, F., 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. pp. 704–711. IEEE (2021)
23. Svetnik, V., Liaw, A., Tong, C., Culberson, J.C., Sheridan, R.P., 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), 1947–1958 (2003)
24. Uriot, T., Virgolin, M., Alderliesten, T., Bosman, P.: On genetic programming representations and fitness functions for interpretable dimensionality reduction (2022), https://arxiv.org/abs/2203.00528
25. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: networked science in machine learning. SIGKDD Explorations **15**(2), 49–60 (2013). https://doi.org/10.1145/2641190.2641198
26. Xue, B., Zhang, M., 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. pp. 81–88. ACM (2012)
27. Zhao, H.: A multi-objective genetic programming approach to developing pareto optimal decision trees. Decision Support Systems **43**(3), 809–826 (2007)