Contrastive Embedding of Structured Space for Bayesian Optimisation

Josh Tingey
University College London
j.tingey.16@ucl.ac.uk

Zhenwen Dai
Spotify
zhenwend@spotify.com

Ciarán Gilligan-Lee
Spotify
University College London
ciaranl@spotify.com

Abstract

Bayesian optimisation (BO) has been used to search in structured spaces described by a context-free grammar, such as chemical molecules. Previous work has used a probabilistic generative model, such as a variational autoencoder, to learn a mapping for the structured representations into a compact continuous embedding within which BO can take advantage of local proximity and identify good search areas. However, the resultant embedding does not fully capture the structural proximity relations of the input space, which leads to inefficient search. In this paper, we propose to use contrastive learning to learn an alternative embedding. We outline how a subtree replacement strategy can generate structurally similar pairs of samples from the input space for use in contrastive learning. We demonstrate that the resulting embedding captures more of the structural proximity relationships of the input space and improves BO performance when applied to a synthetic arithmetic expression fitting task and a real-world molecule optimisation task.

1 Introduction

There is a family of valuable search problems that deal with discrete structured data, such as arithmetic expressions, molecular structures, source code, and neural architectures. Commonly, these structured search spaces contain both a high number of effective dimensions and a vast number of possible candidate points. Therefore, using a sample efficient search technique such as Bayesian Optimisation (BO) in such settings is desirable. However, most BO methods are developed for continuous, low-dimensional space—making their application challenging. Previous work has addressed this problem using probabilistic generative models, such as Variational Autoencoders (VAEs), to learn a mapping (on a meta-level) from the structured input space to a low-dimensional and continuous embedding. BO can then be performed within this space to optimise the search objective at hand. This procedure is usually referred to as Latent Space Optimisation (LSO) and has been successfully applied in a range of scenarios, including chemical design (1; 2; 3; 4), neural architecture search (5; 6), and the automatic statistician (7).

However, by design, the generative nature of VAEs does not necessarily produce an optimal embedding within which to perform optimisation. Indeed, generative models must explicitly model the complete variance of the input data domain; whilst this generally leads to a smooth local structure, it ignores the fact that a large amount of the data variance may be unimportant to the objective—and more distant structural relationships may be helpful. One advantage of structured input spaces is that we typically have prior knowledge of how structurally similar data points are, and so a method that explicitly uses these distance relations, unlike a VAE, could improve the quality of the embedding for optimisation.

Recently an alternative representation learning methodology, contrastive learning, has shown considerable success in learning generalised representations of data in a self-supervised manner (8).

5th Workshop on Meta-Learning at NeurIPS 2021, held virtually.

Unlike a VAE, contrastive learning learns a representation of the input data by explicitly modelling the similarity and dissimilarity between points. The learning procedure attempts to map similar data points close together in the embedding and dissimilar data points further apart. Most successfully applied to computer vision tasks (9; 10; 11; 12), contrastive learning has now also been applied to the audio domain (13; 14) and graph-structured data (15; 16; 17).

It naturally follows that an embedding learnt using contrastive learning and a notion of similarity between structured data points may lead to an improved embedding space to conduct optimisation within. This approach would allow for greater control over the distance relationships within the embedding and allows domain-specific knowledge to be easily incorporated to help the subsequent BO optimisation ignore known unimportant variance in the data. Although just an initial step, we believe that this idea could prove helpful in many structured meta-learning applications, where the distance relationships that are deemed important to the objective could also be learnt on a meta-level.

Following the same observation as (1), we use the fact that such structured data can commonly be described by the parse trees of a context-free grammar. We show that this allows for a simple subtree replacement strategy to be used to generate similar points within the input space with which to train an embedding via contrastive learning. By applying this approach to two tasks: arithmetic expression fitting and SMILES chemical drug-likeness, we explore how this impacts the learnt embedding and the following BO procedure.

The main contributions of our work are:

1. We propose to use contrastive learning to learn a low-dimensional, continuous embedding for structured data defined by a context-free grammar and develop a subtree replacement strategy to generate structurally similar pairs for contrastive learning.
2. We demonstrate that contrastive learning places structurally similar points closer to each other within the embedding. Importantly, it appears to be more resilient than a VAE.
3. We empirically show the contrastive embedding outperforms a VAE in certain BO scenarios.

2 Background and Problem Statement

Structured Input Spaces

In this work we consider the case were $\mathcal{X}$, the *input space* is described by the structure of a context-free grammar (CFG) (18). A CFG is defined by the 4-tuple $G = (V, \Sigma, R, S)$, were V is a finite set of *non-terminal symbols*, Σ is a finite set of *terminal symbols*, R is a finite set of *production rules*, and S is a distinct non-terminal *start symbol*. Each production rule in R describes a mapping $\alpha \to \beta$, for $\alpha \in V$ and $\beta \in (V \cup \Sigma)^*$, with * denoting the Kleene closure, taking a single non-terminal symbol to a sequence of non-terminal and/or terminal symbols.

The application of a production rule R to a non-terminal symbol defines a tree where the symbols of β become child nodes for the parent non-terminal of α. Given this, the complete grammar G encompasses the set of all possible trees that can be formed beginning from the start symbol root node S and recursively applying production rules in R to non-terminal nodes until all leaf nodes are terminal symbols in Σ. The left-to-right traversal of the leaf nodes in a tree produces a corresponding string sequence of terminal symbols; across G the complete set of these sequences defines the *language* of the grammar, which we use as the structured input space $\mathcal{X}$. Conversely, every string sequence $\mathbf{x} \in \mathcal{X}$ can be parsed by the rules of the grammar to a corresponding parse tree representation $\mathbf{x}_{tree}$.

Latent Space Optimisation

Within $\mathcal{X}$ we wish to optimise an *objective function* $f : \mathcal{X} \mapsto \mathbb{R}$. The function f is typically an expensive to evaluate *black-box* that has no derivative information and no analytic form. Hence, it is desirable to conduct the optimisation procedure in a *sample-efficient* manner by evaluating f as few times as possible. Instead of performing the optimisation directly in the discrete high-dimensional space of $\mathcal{X}$, latent space optimisation (LSO) proposes to instead learn a mapping $g : \mathcal{X} \mapsto \mathbb{Z}$ to a continuous low-dimensional *embedding space* $\mathbb{Z}$ within which the optimisation can be conducted. The mapping g therefore takes the discrete data points $\mathbf{x} \in \mathcal{X}$ to continuous ones $\mathbf{z} \in \mathbb{Z}$. To enable optimisation, a *latent objective model* $m : \mathbb{Z} \mapsto \mathbb{R}$ can be constructed in $\mathbb{Z}$ such that it approximates the objective function $f(x) \approx m(g(\mathbf{x}))$, $\forall \mathbf{x} \in \mathcal{X}$.

Previously, the mapping g has typically been chosen to be the encoder of a VAE (19; 20) trained on a sample from $\mathcal{X}$. Furthermore, m is commonly chosen to be a probabilistic model such as a Gaussian process (21) allowing for the application of BO to perform sample-efficient optimisation (22).

Contrastive Learning

We propose to learn an alternative mapping g using contrastive learning. Contrastive learning, put simply, aims to encode similar pairs of points from the input space close together in the embedding and dissimilar pairs of points far apart. Usually used as a self-supervised pre-training step to learn a good representation of the data before fine-tuning on some downstream task, contrastive learning has succeeded in various domains. Although the authors do not know of any previous application of contrastive learning to structured space as defined by a context-free grammar, previous work has used contrastive learning in the structured setting of graphs (15; 16; 17), and relevant to this work, the graphs of chemical molecules (23; 24).

Here we outline the notation of contrastive learning as given in (8) which builds upon that first discussed in (11). A generalised contrastive learning framework consists of the following components:

1. **Query, Key:** The *query* and *key* refer to a pair $(\mathbf{q}, \mathbf{k})$ of either positive (similar) or negative (dissimilar) views of an input sample $\mathbf{x} \in \mathcal{X}$.
2. **Similarity Distribution:** A *similarity distribution* $p^{+}(\mathbf{q}, \mathbf{k}^{+})$ is a distribution over a pair of input samples that describes the notion of similarity. A key is considered positive $\mathbf{k}^{+}$ for a query $\mathbf{q}$ if it is sampled from the similarity distribution and is considered negative $\mathbf{k}^{-}$ if it is sampled from the *dissimilarity distribution* $p^{-}(\mathbf{q}, \mathbf{k}^{-})$. As is done in this work, it is uncommon to sample the dissimilarity distribution explicitly, and instead, dissimilarity is defined by any key-query pair not explicitly sampled from the similarity distribution.
3. **Encoder:** A neural network (of any architecture) that maps both views $(\mathbf{q}, \mathbf{k})$ of $\mathbf{x}$ to an embedding vector $\mathbf{z} \in \mathbb{Z}$. In the context of this work, once trained, the encoder acts as the mapping g from the input space to the embedding $\mathbb{Z}$ for use in LSO.
4. **Projection Head:** A small dense neural network that maps all embeddings $\mathbf{z} \in \mathbb{Z}$ to a space where the contrastive loss is applied $\mathbf{v} \in \mathbb{V}$. The inclusion of the projection head separates learning a good embedding representation that generalises well to multiple tasks from learning an effective embedding to maximise the singular task of similarity between points.
5. **Contrastive Loss:** A loss function that takes as input a set of embedded pairs in $\mathbb{V}$ $\{(v, v^{+}), \ldots, (v, v^{-})\}$. By calculating a measure of similarity between the embeddings and maximising the similarity of positive pairs, and minimising the similarity of negative pairs, the contrastive loss can learn a representation of the inputs.

3 Contrastive Embedding of Structured Space

In this section, we describe how we learn a continuous low-dimensional embedding of the structured input space $\mathcal{X}$ using contrastive learning as is schematically illustrated in Figure 1. Firstly, we introduce how we define the similarity distribution $p^{+}(\mathbf{q}, \mathbf{k}^{+})$ in the input space and how we use this to generate positive pairs $(\mathbf{q}, \mathbf{k})$ for use during training (Section 3.1). We then detail the specifics of how we implement contrastive learning, including the choice of the encoder, projection head, and contrastive loss function (Section 3.2).

3.1 Data Augmentation by Subtree Replacement

As we wish to define similarity within $\mathcal{X}$ based on the underlying structure imposed by the context-free grammar, it naturally follows that structurally similar parse tree representations in $\mathcal{X}$ should be considered similar. Therefore, given a sample $\mathbf{x}$, by making changes to its parse tree representation $\mathbf{x}_{tree}$ we can generate a similar (positive) sample. We propose to make these changes using a simple subtree replacement strategy.

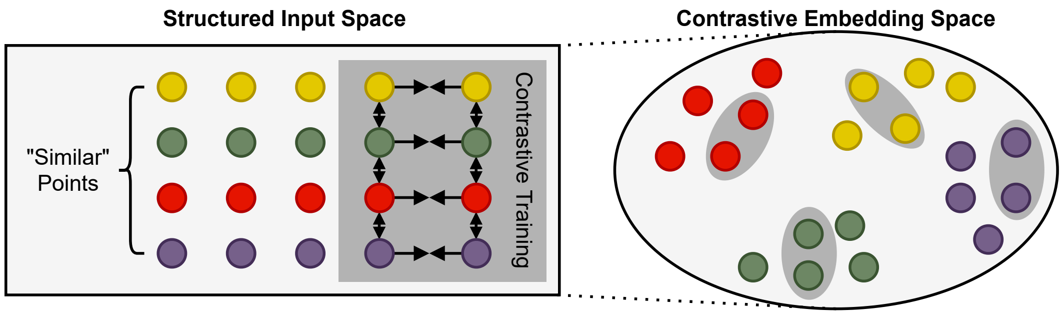

Figure 1: Schematic illustration of contrastive learning for structured space. The structured input space is shown on the left, whilst the learnt contrastive embedding is shown on the right. We use the structural similarity of points in the input space (indicated by the colours) to embed similar points close together in the embedding and dissimilar points further apart. The grey (contrastive training) square indicates the subset of the input space used for training, with the arrows indicating how contrastive learning 'pulls' similar points together and 'pushes' dissimilar points apart.

Given an input sample of data from $\mathcal{X}$ we first generate a subtree replacement lookup dictionary L. For every $\mathbf{x}$ in the sample we generate its parse tree representation $\mathbf{x}_{tree}$ and then iterate over all subtrees, adding those not currently found to sets of subtrees in L with a key given by a combination of their root node non-terminal symbol and their height.

Given the generated replacement dictionary L, along with a maximum subtree replacement height H, and the maximum number of subtrees to replace N, we perform our subtree replacement strategy as outlined in Algorithm 1, whose underlying procedure is schematically shown in Figure 2. For an input (anchor) sequence $\mathbf{x}$ subtrees up to height H are randomly selected from its parse tree representation and replaced using a randomly chosen subtree from L with the same root symbol. This replacement is repeated up to N times.

Algorithm 1 Positive pair generation by subtree replacement

```
   Input: Anchor sequence $\mathbf{x}$, number of replacements $\mathbf{N}$, maximum subtree height $\mathbf{H}$, subtree
   replacement lookup dictionary $\mathbf{L}$
   Output: Positive sequence $\tilde{\mathbf{x}}$
1: Parse $\mathbf{x}$ to tree representation $\mathbf{x}_{tree}$
2: $\mathbf{n} \leftarrow$ random choice from $1, ..., \mathbf{N}$
3: for $1, \ldots, \mathbf{n}$ do
4:     $\mathbf{t} \leftarrow$ randomly select subtree in $\mathbf{x}_{tree}$ with height less than $\mathbf{H}$
5:     $\mathbf{r} \leftarrow$ randomly select replacement subtree from $\mathbf{L}$ given $\mathbf{t}$
6:     Replace $\mathbf{t}$ with $\mathbf{r}$ in $\mathbf{x}_{tree}$
7: return left-to-right traversal of the leaf nodes in $\mathbf{x}_{tree}$ as $\tilde{\mathbf{x}}$
```

Furthermore, we wrap the replacement procedure in a function to verify the output sequence. It is the case in specific input spaces such as SMILES strings for chemical molecules (introduced in Section 4) that not all string sequences correspond to valid data points, e.g. molecules. If an output sequence is invalid, the replacement procedure is repeated up to 50 times until a valid output is found. If all attempts are exhausted, the sequence is discarded.

One vital aspect of this is that, due to the branching tree structure of the input data, a greater number of smaller subtree replacements can be made than larger ones. Thus, small changes in the parse tree are more likely to occur than larger ones. In contrastive learning, this encourages more similar trees to be closer in the latent space as they occur as positive pairs more often, whilst trees with more significant differences are encouraged to be further away as they occur less often.

Unlike the common contrastive learning approach of generating two augmented versions from the same input to use as a positive pair during training, we use each example $\mathbf{x}$ as one half of each positive

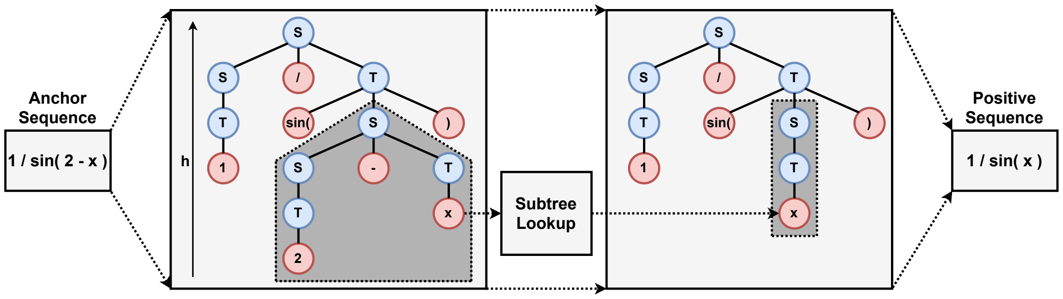

Figure 2: Diagram of the subtree replacement procedure. The anchor input sequence **x** is converted to its parse tree representation as defined by the context-free grammar, and a subtree is replaced to create the positive pair. In reality, the replacement procedure illustrated in the centre of the diagram is repeated *N* times before the final tree is converted back to its string sequence.

pair **q** and a single augmented version $\tilde{\mathbf{x}}$ as the other **k**. We find that this technique produces a better performing contrastive embedding.

Furthermore, due to the computational bottleneck of parsing the sequences to their corresponding parse tree representations, we do not generate positive pairs on the fly but instead calculate a static set of input data by iterating over the anchor input data with the subtree replacement procedure a fixed number of times before training.

3.2 Contrastive Learning of Embedding

We now describe how we implement the rest of the contrastive learning framework (see the supplementary material for full details). As input to the encoder, we follow (1), where each input is encoded as a series of 1-hot vectors representing the sequence of production rules *R* found by performing a pre-order traversal on the branches of the corresponding parse tree going from left-to-right. Each 1-hot is an indicator vector with each dimension corresponding to a singular rule within the grammar.

For the encoder, we use the same encoder architecture as the VAE in (1). With an embedding dimensionality of 25 and 56 for the arithmetic expression and SMILES chemical experiments introduced in Section 4, respectively. We simply employ a small dense network for the projection head consisting of three fully connected, 128 unit, dense layers.

For the contrastive loss we use the NT-Xent (the normalised temperature-scaled cross-entropy loss) from SimCLR (9; 10). Crucially, instead of explicitly sampling negative examples, we treat all other pairs of samples within each minibatch during training as negative examples. As a notion of similarity the loss uses the cosine similarity: $sim(\mathbf{u}, \mathbf{v}) = \mathbf{u}^\top \mathbf{v} / \|\mathbf{u}\| \, \|\mathbf{v}\|$. The full loss implementation for a positive pair of examples (i,j) is then defined as

$$\mathcal{L} = -\log \frac{\exp(sim(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} \exp(sim(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \tag{1}$$

where τ indicates the temperature parameter and *N* is the size of the minibatch.

4 Experiments

In this section, we aim to show: 1) How the structure of the trained embedding for both a VAE and a contrastive model differ, and 2) Empirically how the contrastive embedding performs compared to one from a VAE in BO. For this purpose, we follow the two optimisation tasks used in previous work for consistency that of arithmetic expressions fitting and SMILES chemical drug-likeness explained below. For each, we train both a VAE using the same settings as in (1) and a contrastive model using the methods outlined in Section 3.

Table 1: Table showing the average cosine similarity between a test sample of positive pairs for both the arithmetic expressions and SMILES chemicals within each models embedding.

Input Space	VAE Similarity	Contrastive Similarity
Arithmetic Expressions	0.462 ± 0.003	0.831 ± 0.003
SMILES Chemicals	0.697 ± 0.004	0.952 ± 0.004

Arithmetic Expression Task

Following the optimisation task previously explored in (1; 25) we aim to optimise the fit of single variable arithmetic expressions generated from a context-free grammar to a target expression. We randomly generate 100,000 expressions with a parse tree height less than 7 from the grammar (see supplementary material for full details) to use as training data, examples of which include sin(2), x/(3+1), x+exp(2), 1/x+exp(x)*sin(2/3). For 1000 points evenly spaced between -10 and 10 in x, we aim to find the expression with the minimum mean squared error (MSE) to the target expression 1/3*x*sin(x*x). Due to the exponential function within the grammar, in practice, we use the log(1 + MSE) score to suppress the resulting large MSE value from some expressions. We iterate over the 100,000 anchor samples 25 times to generate the static training set for use during contrastive training, with a maximum subtree replacement height H of 6 and the maximum number of subtrees to replace N of 3. For the VAE model, we train on just 100,000 samples.

SMILES Chemicals Task

Following the optimisation task initially presented in (2) and again explored in (1; 25) we aim to optimise the drug properties of molecules. The goal is to maximise the *penalised water-octanol partition coefficient* (logP) of molecules, an indicator for its drug-likeness. Within this work we instead frame the task as minimising the -logP. As training data, we randomly take 25,000 SMILES strings from the QM9 dataset of molecules with fewer than 9 heavy atoms (26). SMILES (Simplified Molecular-Input Line-Entry System) is a specification[1] that describes the structure of molecules using strings (27) and can be described using a context-free grammar (see supplementary material for full details). Some example SMILES strings include CC1=CN(C)C(=O)C1O and N=C1OC2NC1(O)C2. We iterate over the 25,000 anchor samples 25 times to generate the static training set for use during contrastive training, with a maximum subtree replacement height H of 12, and the maximum number of subtrees to replace N of 2. For the VAE model, we train on just 25,000 samples.

4.1 Latent Space Distribution

To qualitatively compare the structure of the learnt embedding for both the VAE and the contrastive model, we visualise how similar points are placed within the embedding space. We encode a test set of 100,000 positive pairs of arithmetic expression examples generated in the same way as the training data to the embedding $\mathbb{Z}$ for both the VAE and contrastive model. For both models, we centre and normalise the embeddings so that they are comparable. We then visualise a thin slice through the 25 dimension space for both models and show just the first two dimensions (all other dimensions are restricted to be close to zero), the result of which is shown in Figure 3. We also indicate two randomly selected positive pairs of examples for each model.

The contrastive embedding produces a vastly more clustered space relative to the VAE. Strongly similar points (those with small height subtree replacements) are placed in individual isolated clusters, whilst less similar points are placed in nearby clusters. Using the same test set and a corresponding one for the SMILES chemicals, we also calculate the average cosine similarity between positive pairs using both models, as in Table 1. The contrastive model displays an improved similarity between structurally similar points. This difference is understandable given the explicit cosine similarity requirement of the contrastive loss, but it clearly shows how contrastive learning produces an embedding where structurally similar points are more tightly bound together than in a VAE.

[1] `http://opensmiles.org/opensmiles.html`

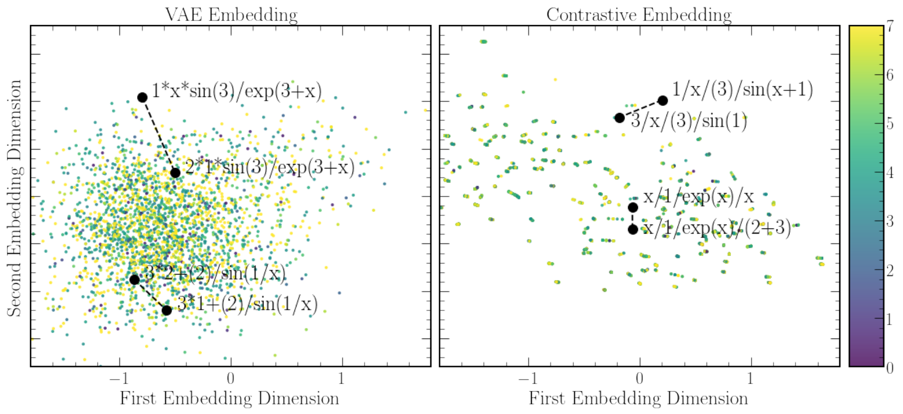

Figure 3: A slice through the learnt embedding for both the VAE (left) and contrastive model (right) trained on the arithmetic expression data. For each slice, two positive pairs of examples are shown. The contrastive embedding contains dense clusters of very similar points, while the VAE produces a more uniform space. When comparing the positive pairs, it can be seen that even for multiple augmentations with non-trivial subtree replacements, pairs remain close in the contrastive embedding, whilst even pairs with minor differences can be placed relatively far apart in the VAE.

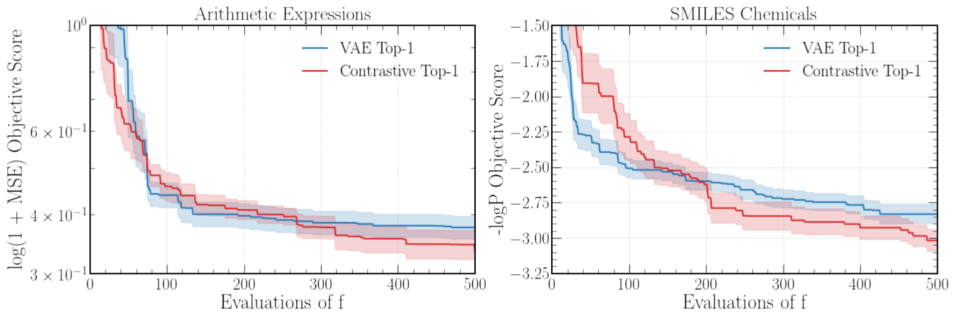

Figure 4: Top-1 score performance during BO for both the arithmetic expression fitting task and the SMILES molecules drug likeness task using both a VAE and contrastive embedding. The contrastive embedding is observed to perform better in both cases. The shaded regions correspond to the standard deviation divided by the square root of the number of trials (standard error on the mean).

4.2 Bayesian Optimisation

We perform BO in the trained embeddings for both the arithmetic expressions and SMILES chemicals. Previous work (2; 1; 25) approached the BO task by first training a sparse Gaussian process model on a large number of data points before starting the BO procedure. We instead choose to begin with no known initial objective scores. Furthermore, as the contrastive model does not naturally contain a decoding element, we instead consider the task of finding the optimal sample from a set of candidates. Although previous work with VAEs could decode candidate points back to the original space, and therefore, was not limited to a candidate set, we don't believe this is a limitation in practice. This is because all possible candidate points (particularly those of interest) can be determined upfront using the context-free grammar.

We first choose a random candidate set of 100,000 and 50,000 samples from outside the training data for the arithmetic expressions and SMILES chemicals. We start our BO procedure by fitting a Gaussian process to a random selection of 50 and 200 candidate points from the arithmetic expressions and SMILES chemical candidate sets, respectively. We then perform 500 iterations of BO using the expected improvement acquisition function computed over all candidate points at each step. Figure 4 shows the average Top-1 score (the top scoring point at each iteration) over 20 trials of BO for each of the tasks and each model. For both the arithmetic expression and SMILES chemical tasks, the BO is found to perform better in the embedding space of the contrastive model.

5 Limitations and Future Work

The work presented in this paper is only a first look at how contrastive learning can be applied to structured spaces defined by a context-free grammar. Therefore, a different formulation of similarity or a different choice of contrastive loss not studied here could provide significant performance improvements. This is especially the case when we have not explicitly considered the interplay between the structural similarity of points and their actual objectives scores, which could prove incredibly beneficial compared to the VAE approach.

Additionally, the graphical nature of the input data may be better exploited by employing a graph neural network as the encoder. This change could also allow positive pairs to be generated on the fly as the bottleneck of parsing sequences to trees could be removed.

6 Conclusion

We have shown that contrastive learning can be used to learn an effective embedding within which to perform BO. We have shown that a simple subtree replacement strategy can generate positive pairs of samples from the structured input space described by a context-free-grammar to perform the contrastive learning. The resulting embedding is shown to embed the structural similarity of points more accurately than a VAE, resulting in improved BO performance. We believe there is considerable scope to apply contrastive learning in other such structured domains.

Acknowledgments

The work of Josh Tingey was supported by the UCL Centre for Doctoral Training in Data Intensive Science, STFC grant ST/P006736/1.

References

[1] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *International Conference on Machine Learning*, pp. 1945–1954, PMLR, 2017.

[2] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS central science*, vol. 4, no. 2, pp. 268–276, 2018.

[3] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *International Conference on Machine Learning*, pp. 2323–2332, PMLR, 2018.

[4] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song, "Syntax-directed variational autoencoder for structured data," *arXiv preprint arXiv:1802.08786*, 2018.

[5] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," *arXiv preprint arXiv:1808.07233*, 2018.

[6] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-vae: A variational autoencoder for directed acyclic graphs," *arXiv preprint arXiv:1904.11088*, 2019.

[7] X. Lu, J. Gonzalez, Z. Dai, and N. D. Lawrence, "Structured variationally auto-encoded optimization," in *International conference on machine learning*, pp. 3267–3275, PMLR, 2018.

[8] P. H. Le-Khac, G. Healy, and A. F. Smeaton, "Contrastive representation learning: A framework and review," *IEEE Access*, vol. 8, p. 193907–193934, 2020.

[9] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, pp. 1597–1607, PMLR, 2020.

[10] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton, "Big self-supervised models are strong semi-supervised learners," *arXiv preprint arXiv:2006.10029*, 2020.

[11] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, “Momentum contrast for unsupervised visual representation learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.

[12] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, “Unsupervised learning of visual features by contrasting cluster assignments,” *arXiv preprint arXiv:2006.09882*, 2020.

[13] A. Saeed, D. Grangier, and N. Zeghidour, “Contrastive learning of general-purpose audio representations,” in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3875–3879, IEEE, 2021.

[14] A. v. d. Oord, Y. Li, and O. Vinyals, “Representation learning with contrastive predictive coding,” *arXiv preprint arXiv:1807.03748*, 2018.

[15] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, “Gcc: Graph contrastive coding for graph neural network pre-training,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1150–1160, 2020.

[16] K. Hassani and A. H. Khasahmadi, “Contrastive multi-view representation learning on graphs,” in *International Conference on Machine Learning*, pp. 4116–4126, PMLR, 2020.

[17] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, “Graph contrastive learning with augmentations,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[18] J. E. Hopcroft, R. Motwani, and J. D. Ullman, “Introduction to automata theory, languages, and computation, 2nd edition,” *SIGACT News*, vol. 32, p. 60–65, Mar. 2001.

[19] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

[20] D. J. Rezende, S. Mohamed, and D. Wierstra, “Stochastic backpropagation and approximate inference in deep generative models,” in *International conference on machine learning*, pp. 1278–1286, PMLR, 2014.

[21] C. E. Rasmussen, “Gaussian processes in machine learning,” in *Summer school on machine learning*, pp. 63–71, Springer, 2003.

[22] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, “Taking the human out of the loop: A review of bayesian optimization,” *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[23] Y. Wang, J. Wang, Z. Cao, and A. B. Farimani, “Molclr: Molecular contrastive learning of representations via graph neural networks,” *arXiv preprint arXiv:2102.10056*, 2021.

[24] Z. Wang, M. Liu, Y. Luo, Z. Xu, Y. Xie, L. Wang, L. Cai, and S. Ji, “Moleculekit: Machine learning methods for molecular property prediction and drug discovery,” *arXiv preprint arXiv:2012.01981*, 2020.

[25] A. Tripp, E. Daxberger, and J. M. Hernández-Lobato, “Sample-efficient optimization in the latent space of deep generative models via weighted retraining,” in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 11259–11272, Curran Associates, Inc., 2020.

[26] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld, “Quantum chemistry structures and properties of 134 kilo molecules,” *Scientific data*, vol. 1, no. 1, pp. 1–7, 2014.

[27] D. Weininger, “Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules,” *Journal of Chemical Information and Computer Sciences*, vol. 28, no. 1, pp. 31–36, 1988.

[28] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

A Additional Experimental Details

A.1 Context-free Grammars and Tasks

For the single variable arithmetic expression task, we used the same context-free grammar definition as in (1) shown below. S and T denote non-terminal symbols, whilst all symbols in quotation marks indicate terminals. We generated sequences using this grammar up to a maximum parse tree height of six and 15 total productions to produce the training data. Unlike previous work (1; 25) which used the first 100,000 expressions that are algorithmically generated using the grammar, we chose to select the 100,000 training samples from the approximately 1.3 million possibilities randomly. The 100,000 Bayesian optimisation candidate set was also randomly selected in the same way.

$$\begin{aligned}
&\mathrm{S} \rightarrow \mathrm{S}\ `+\text{'}\ \mathrm{T} \quad | \quad \mathrm{S}\ `*\text{'}\ \mathrm{T} \quad | \quad \mathrm{S}\ `/\text{'}\ \mathrm{T} \quad | \quad \mathrm{T}\\
&T \rightarrow `(\text{'}\ \mathrm{S}\ `)\text{'} \quad | \quad `\sin(\text{'}\ \mathrm{S}\ `)\text{'} \quad | \quad `\exp(\text{'}\ \mathrm{S}\ `)\text{'} \quad | \quad `\mathrm{x}\text{'} \quad | \quad `1\text{'} \quad | \quad `2\text{'} \quad | \quad `3\text{'}
\end{aligned}$$

The arithmetic expression objective score is given by

$$\text{Objective score}(\mathbf{x}, \mathbf{x}^*) = \min\left(\log(1 + \mathrm{MSE}(\mathbf{x}, \mathbf{x}^*)),\ 7.0 \right),$$

where $\mathbf{x}$ is the sequence to be scored and $\mathbf{x}^*$ is the target sequence 1/3*x*sin(x*x). A maximum score of 7.0 is used to prevent large values from affecting the Bayesian optimisation procedure.

For the SMILES chemicals task, we used the context-free grammar definition given by the openSMILES specification[2] and shown below. In our case, we substituted the start symbol (smiles) for S to be consistent with the arithmetic expression grammar. Using the QM9 dataset of molecules with fewer than nine heavy atoms (26) we encoded the SMILES sequences using a one-hot encoding up to 75 productions in size. We used the negative form of the penalised water-octanol partition coefficient for the SMILES chemicals objective score as used in (25) and with a maximum score of 10.0 to prevent large values from affecting the Bayesian optimisation procedure.

[2] `http://opensmiles.org/opensmiles.html`

S → chain

atom → bracket_atom | aliphatic_organic | aromatic_organic

aliphatic_organic → ‘ B ’ | ‘ C ’ | ‘ N ’ | ‘ O ’ | ‘ S ’ | ‘ P ’ | ‘ F ’ | ‘ I ’ | ‘ Cl ’ | ‘ Br ’

aromatic_organic → ‘ c ’ | ‘ n ’ | ‘ o ’ | ‘ s ’

bracket_atom → ‘ [’ BAI ‘] ’

BAI → isotope symbol BAC | symbol BAC | isotope symbol | symbol

BAC → chiral BAH | BAH | chiral

BAH → hcount BACH | BACH | hcount

BACH → charge class | charge | class

symbol → aliphatic_organic | aromatic_organic

isotope → DIGIT | DIGIT DIGIT | DIGIT DIGIT DIGIT

DIGIT → ‘ 1 ’ | ‘ 2 ’ | ‘ 3 ’ | ‘ 4 ’ | ‘ 5 ’ | ‘ 6 ’ | ‘ 7 ’ | ‘ 8 ’ |

chiral → ‘ @ ’ | ‘ @@ ’

hcount → ‘ H ’ | ‘ H ’ DIGIT

charge → ‘ − ’ | ‘ − ’ DIGIT | ‘ − ’ DIGIT DIGIT | ‘ + ’ | ‘ + ’ DIGIT | ‘ + ’ DIGIT DIGIT

bond → ‘ − ’ | ‘ = ’ | ‘ / ’ | ‘ \ ’

ringbond → DIGIT | bond DIGIT

branched_atom → atom | atom RB | atom BB | atom RB BB

RB → RB ringbond | ringbond

BB → BB branch | branch

branch → ‘ (’ chain ‘) ’ | ‘ (’ bond chain ‘) ’

chain → branched_atom | chain branched_atom | chain bond branched_atom

A.2 Network Structure and training

We follow the same variational autoencoder model architecture for both the arithmetic expressions and SMILES chemicals tasks as in (1), updated to run using Tensorflow 2. For the contrastive models, we take the encoder of the variational autoencoder architecture and use the mean of the latent embedding distribution as input to the small three layer, 128 units per layer, projection head before the contrastive loss. All models are trained with a minibatch size of 1024 (1024 pairs of examples in the contrastive case) using the Adam optimiser (28) for 50 epochs.

A.3 Bayesian Optimisation

For our Bayesian optimisation implementation we re-implemented the code[3] used in (25) for our simpler candidate set use case. The code uses the modern (and fast) GPflow Gaussian process library built on top of Tensorflow 2.

A.4 Computing resources and run times

Generating 25 iterations of the input anchor samples took approximately 20 hours for the 100,000 arithmetic expressions and approximately 15 hours for the 25,000 SMILES QM9 chemicals, predominately using a single CPU core. For both model training and Bayesian optimisation, we used a single GPU (Nvidia GeForce RTX 2080), for which the approximate average run times are given in Table 2. In practice, we ran multiple experiments in parallel using multiple GPUs to reduce the combined run time. As the contrastive model does not require the training of a decoding element, the training time is much faster than that of the corresponding variational autoencoder, which may be of significance when applied to much larger datasets.

[3]`https://github.com/cambridge-mlg/weighted-retraining`

Table 2: Average run times in hours for model training and Bayesian Optimisation for both tasks using a single GPU.

Task	VAE Training	Contrastive Model Training	Bayesian Optimisation
Arithmetic Expressions	1:15	0:20	4:00
SMILES Chemicals	2:15	0:40	3:30

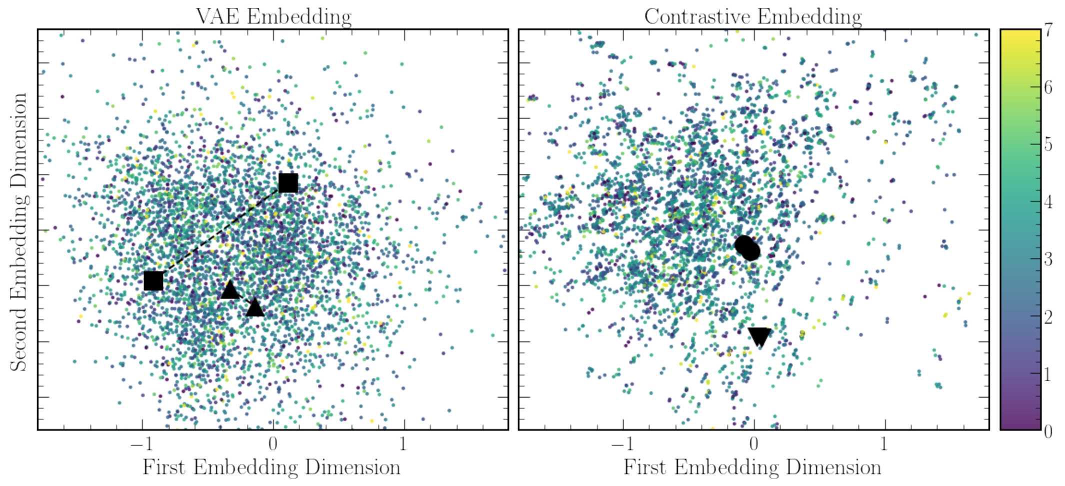

Figure 5: A slice through the learnt embedding for both the VAE (left) and contrastive model (right) trained on the SMILES chemicals QM9 data. For each slice, two positive pairs of examples are indicated, with the corresponding visualisations for these positive pairs shown in Figure 6. As in the arithmetic expression case, the contrastive embedding contains dense clusters of very similar points, while the VAE produces a more uniform space. When comparing the positive pairs, it can be seen that even for small augmentations, the VAE can place positive pairs relatively far apart, whilst pairs predominantly remain close in the contrastive embedding.

B Further Experimental Results

B.1 Latent Space Distribution for SMILES chemicals

Similarly to that conducted for the arithmetic expressions, we visualise how similar points are placed within the embedding space of the SMILES chemicals. We encode a test set of 50,000 positive pairs of SMILES chemicals examples generated in the same way as the training data to the embedding space for both the VAE and contrastive model. We then visualise a thin slice through the 56 dimension space for both models and show just the first two dimensions (all other dimensions are restricted to be close to zero), the result of which is shown in Figure 5. We also indicate two randomly selected positive pairs of examples for each model, which are shown as molecules in Figure 6.

B.2 Bayesian Optimisation for lower dimensional embedding

We also conducted the same Bayesian optimisation procedure using the embeddings of VAE and contrastive models trained with a reduced dimensionality embedding space. We restricted the embedding dimensionality to eight for both models and conducted Bayesian optimisation in the same way as previously described. Figure 7 shows the Bayesian optimisation Top-1 results for the arithmetic expressions and SMILES chemicals tasks using the reduced dimensionality. Again the contrastive model embedding provides improved Bayesian optimisation performance compared to the variational autoencoder embedding. Interestingly, both embeddings, particularly the contrastive one, see no dramatic change in performance compared to the default dimensionality embeddings. Future work may wish to explore how dimensionality affects latent space optimisation more rigorously. Although not presented in this work and not comprehensively studied, we also considered how a VAE trained on all examples used in the contrastive training procedure performed with an embedding

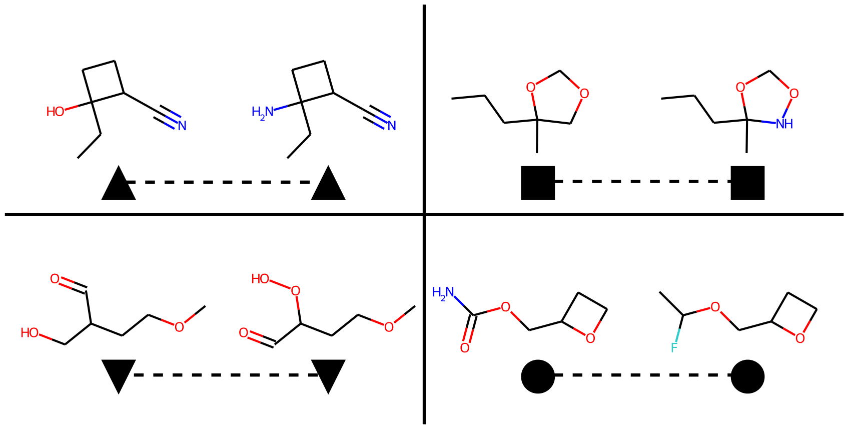

Figure 6: Molecule visualisations for the positive pairs indicated in Figure 5

.

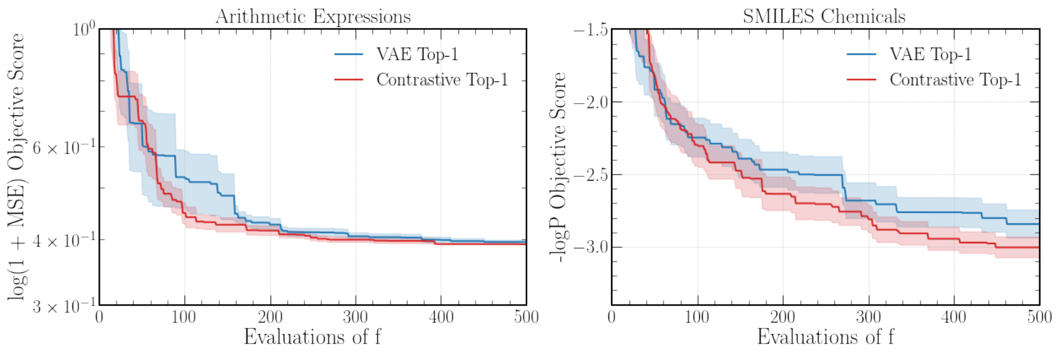

Figure 7: Top-1 score performance during Bayesian optimisation for both the arithmetic expression fitting task and the SMILES molecules drug likeness task using both a variation autoencoder and a contrastive embedding for an embedding dimensionality of eight. Again the contrastive embedding is found to provide advantageous performance. The shaded regions correspond to the standard deviation divided by the square root of the number of trials (standard error on the mean).

dimensionality to eight. The contrastive embedding still exhibited slightly improved performance compared to a VAE trained in this way.