



UNIVERSITY OF
CAMBRIDGE

Department of Computer
Science and Technology

Task-Agnostic Graph Neural Network Evaluation via Adversarial Collaboration

Xiangyu Zhao
Trinity College

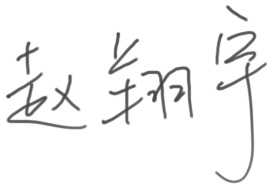
27 May 2022

*A dissertation submitted to the University of Cambridge
in partial fulfilment of the requirements for the
Computer Science Tripos, Part III*

Declaration of Originality

I, Xiangyu Zhao of Trinity College, being a candidate for Computer Science Tripos, Part III, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed:

Handwritten signature in Chinese characters: 赵翔宇

Date: 27 May 2022

Total page count: 58

Main chapters (excluding front-matter, references and appendix): 39 pages (pp. 9–47)

Main chapters word count: 11185

Methodology used to generate that word count:

```
$ make wordcount
texcount -1 -sum report-submission.tex
11185
```

Acknowledgements

This project would not have been possible without the invaluable support from Prof Pietro Liò, Dr Dominique Beaini and Hannes Stärk, including holding weekly meetings to provide me detailed guidance on the project and the dissertation, promptly answering my questions whenever I was in doubt, and kindly encouraging me when I experienced setbacks. I would especially like to thank Hannes Stärk for sharing his 3D Infomax repository with me, and arranging one-to-one coding sessions to help me familiarise its source code. I would like to thank Dr Petar Veličković and Prof Pietro Liò again, for delivering the wonderful Representation Learning on Graphs and Networks (L45) course, which significantly deepened my understandings towards GNNs and self-supervised learning on graphs. I would also like to thank Chaitanya Joshi and Cristian Bodnar, for supervising my L45 mini-project, which also provided me valuable insights towards this project. I am also very grateful to Dr Sean Holden, my Director of Studies and supervisor of my Part II project, who helped me turn my Part II project into a publication and taught me lots of useful academic writing tips.

This work was performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) operated by the University of Cambridge Research Computing Service (www.csd3.cam.ac.uk), provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/T022159/1), and DiRAC funding from the Science and Technology Facilities Council (www.dirac.ac.uk). I would like to thank Malcolm Scott, the IT Infrastructure Specialist at the Department of Computer Science and Technology, for providing me the CSD3 resources, and the Cambridge High Performance Computing Service (HPCS) support team for resolving my questions regarding the CSD3 usage.

Finally, I would like to thank my parents, Haiming Zhao and Xianli Sun, for their wholehearted support, both financially and emotionally, throughout my entire study at Cambridge. I can always feel their enduring love and care even though we are more than 5,000 miles apart. I would also like to give my special thanks to my dear girlfriend, Jing Zeng, for her warmest love and support that accompanied me through all sorts of difficulties.

Abstract

Graph Neural Networks (GNNs) have experienced rapid growth over the last decade, and have been successful in many real-world applications. In order to cope with the rapid growth of this field, it is increasingly demanding to develop reliable GNN evaluation methods to facilitate GNN research and quantify their progress. Current GNN benchmarking methods all focus on comparing the GNNs with respect to their training performances on some node/graph classification/regression tasks in certain datasets, but there has not been any principled, task-agnostic method to directly compare the two GNNs.

Furthermore, learning informative representations of graph-structured data using self-supervised learning (SSL) is becoming crucial in many real-world tasks nowadays, when labelled data are expensive and limited. Most of the existing graph SSL works incorporate handcrafted augmentations to the graph, which has several severe difficulties due to the unique characteristics of graph-structured data. Therefore, it is highly needed to develop a principled SSL framework across various types of graphs, that does not require handcrafted augmentations.

In this project, I tackled both questions above, and developed GraphAC (Graph Adversarial Collaboration), a conceptually novel, principled, task-agnostic, and stable framework for evaluating GNNs through contrastive self-supervision. It consists of two different GNNs directly competing against each other, with the more expressive GNN wins by producing more informative graph representations. I built two frameworks for GraphAC, and designed a novel objective function that enables stable and effective training of two different GNNs, inspired by Barlow Twins.

The experimental results show that GraphAC succeeds in distinguishing GNNs of different expressivity across various aspects including the number of layers, hidden dimensionality, aggregators, GNN architecture and edge features, and always allow more expressive GNNs to win with statistically significant difference. GraphAC proved to be a principled and reliable GNN evaluation method, and enables stable SSL without needing handcrafted augmentations.

Contents

1	Introduction	9
1.1	Motivation	9
1.2	Contributions	11
2	Background	12
2.1	Deep Learning	12
2.1.1	Multi-Layer Perceptrons	12
2.1.2	Training	13
2.2	Graph Neural Networks	14
2.2.1	Permutation Invariance and Equivariance	15
2.2.2	Message Passing Neural Networks	16
2.2.3	Expressivity of Graph Neural Networks	16
2.2.4	Graph Convolutional Networks	18
2.2.5	Graph Isomorphism Networks	18
2.2.6	Principal Neighbourhood Aggregation	19
2.3	Self-Supervised Learning	20
2.3.1	Contrastive Learning	22
3	Related work	24
3.1	General Advances in Contrastive Self-Supervised Learning	24
3.1.1	Barlow Twins: SSL via Redundancy Reduction	25
3.1.2	VICReg: SSL via Variance-Invariance-Covariance Regularisation	26
3.2	Contrastive Self-Supervised Learning on Graphs	28
4	Design and Implementation	29
4.1	Proposed Frameworks	29
4.1.1	Contestant-Judge Framework: GNNs + MLPs	29
4.1.2	Competitive Barlow Twins	31
4.2	Datasets	34
4.3	Implementation	35
4.3.1	Starting Point	35
4.3.2	Contestant-Judge Framework	36

4.3.3	Competitive Barlow Twins	36
5	Experiments and Evaluation	37
5.1	Experimental Setup	37
5.2	Hyperparameter Tuning	37
5.2.1	Contestant-Judge Framework	38
5.2.2	Competitive Barlow Twins	39
5.3	Experiments	40
5.4	Evaluation Metrics	40
5.5	Results	41
5.5.1	Different Numbers of GNN Layers	41
5.5.2	Different Hidden Dimensions	42
5.5.3	Different Aggregators	43
5.5.4	Different GNN Architectures	43
5.5.5	Inclusion of Edge Features	44
5.5.6	Fine-tuning	45
6	Summary and Conclusions	46
6.1	Accomplishments	46
6.2	Future Work	46
	Bibliography	47
	A Notations	54
	B Algorithms	56

List of Figures

- 2.1 Example structures of a perceptron and an MLP 13
- 2.2 Example of execution of the 1-WL test on two isomorphic graphs 17
- 2.3 Example of two non-isomorphic graphs on which the 1-WL test fails . . 17
- 2.4 Paradigms for self-supervised learning 21
- 2.5 Comparison between contrastive learning and predictive learning 22

- 4.1 Architecture of the contestant-judge framework 30
- 4.2 Architecture of the competitive Barlow-Twins framework 33

- 5.1 Loss differences and PCA explained variance of contestant-judge 38
- 5.2 Loss differences and PCA explained variance of competitive Barlow Twins 39

List of Tables

4.1	Statistics of the OGB graph property prediction datasets	35
5.1	Hyperparameters searched for the competitive Barlow Twins framework	39
5.2	Loss differences of PNAs with different numbers of layers	41
5.3	Loss differences of PNAs with different hidden dimensions	42
5.4	Loss differences of PNAs with different aggregators	43
5.5	Loss differences of PNA, GIN and GCN	44
5.6	Loss differences of PNAs with and without edge features	44
5.7	Fine-tuning results of GraphAC on the ZINC dataset	45

Chapter 1

Introduction

1.1 Motivation

Graph Neural Networks (GNNs) have attracted an enormous amount of research interest over the last few years, and have made significant advancements that are successfully applied to a wide diversity of domains beyond computer science, such as chemistry (Gilmer et al., 2017), biology (Stokes et al., 2020), social science (Monti et al., 2019) and e-commerce (Ying et al., 2018). As this field rapidly grows, it becomes crucial to develop reliable benchmark datasets to facilitate GNN research and quantify their performances. The graph datasets built in the early stage, such as Cora (McCallum et al., 2000), CiteSeer (Getoor, 2005) and PubMed (Sen et al., 2008), have soon become deprecated as newly developed GNNs can easily overfit those small-scale datasets, and there is no guarantee that the current widely-accepted GNN benchmarks (Dwivedi et al., 2020; Hu et al., 2020) can be everlasting. While there is an ongoing need to build powerful graph datasets that meet the requirements of the latest progress on GNNs, it is still highly desired to develop GNN evaluation methods that can fully exploit the current datasets, in order to extend their lifespan.

There have been some work concerning the evaluation of GNNs' capacities against theoretical tests such as the Weisfeiler-Lehman (1-WL) graph isomorphism test (Weisfeiler and Leman, 1968; Xu et al., 2019; Dwivedi et al., 2020), but they are limited in the information that they provide. Toy benchmarks were also developed to measure a GNN's ability to detect patterns, substructures and clusters (Dwivedi et al., 2020), or graph properties such as diameter, eccentricity, and spectral radius (Corso et al., 2020). However, they cannot be used consistently, since certain GNN types can use this information as positional encodings, and thus directly cheating the task (Kreuzer et al., 2021; Bodnar et al., 2021; Dwivedi et al., 2022). Therefore, there is a need for a task-agnostic evaluation of the GNNs' expressivity.

Another challenge for GNNs is designing principled self-supervised learning (SSL) methods on graphs. As labelled data can be expensive, limited or even unavailable in many real-world scenarios, it has become increasingly demanding to develop powerful SSL methods on graphs. A lot of successful SSL works on graphs (Veličković et al., 2019; You et al., 2020, 2021; Xu et al., 2021) tend to adapt the success of SSL on images (Chen et al., 2020) to the graph domain, by applying handcrafted augmentations to the graphs and then trying to maximise the mutual information between positive pairs. However, there are several key difficulties in applying augmentations to graphs. Firstly, there exists no universal augmentation that works across all types of graphs. Secondly, graphs are not invariant to augmentations like images: applying filters or rotating an image still preserves its essential invariances, but even a tiny augmentation on a graph can significantly change its topological structure or intrinsic properties. There is another class of SSL methods on graphs that do not require augmentations (Stärk et al., 2021), but they rely on exploiting the mathematical/physical properties of some specific types of graphs, and cannot be generalised to other graph types. Therefore, it is highly desired to develop a principled and generalisable SSL framework that does not require handcrafted augmentations.

In this project, I tackled all the above questions, and developed GraphAC (Graph Adversarial Collaboration), a conceptually novel, principled, and task-agnostic framework for evaluating GNNs in a self-supervised, adversarial collaboration manner, without the need of handcrafted augmentations. In the GraphAC framework, two GNNs directly compete against each other on the same unlabelled graphs, in which the more expressive GNN produces more complex and informative graph embeddings and wins the game.

The analogical reasoning behind GraphAC is as follows: imagine a graph dataset as an arena, where each graph is an animal with a bounty, and a GNN as a warrior. In the conventional GNN benchmarking methods, only one warrior enters the area at each time, and the warriors are ranked by the amount of bounties they claim within a certain period of time, by catching as many animals as they can. However, these methods do not compare the warriors directly against each other, and such ranking methods can fail, if the animals are not strong enough, or the bounties are poorly designed, or one warrior discovers a trick to catch many animals without genuinely making efforts. In GraphAC, the warriors are ranked by directly fighting against each other in the arena, which can produce unquestionable winners based on the warriors' actual skills. Further more, the warriors can also improve their skills through fighting against their opponents, and actively practice to catch more animals in order to claim more bounties, which can help them in defeating their opponents. This also provides an insight of GraphAC to the SSL aspect.

1.2 Contributions

GraphAC has made the following contributions to the graph SSL community:

- Introduces a completely novel principle of evaluating GNNs, by having them directly compete against each other in a self-supervised manner, rather than comparing them using a scoreboard of training performances on some datasets;
- inspired by the novel principle, proposes a novel architecture and an original modification to the existing Barlow Twins loss (Zbontar et al., 2021) that enables the GNNs to stably compete against each other, while ensuring that more expressive GNNs can always win;
- develops a principled SSL framework without needing any handcrafted augmentations, which is also generalisable to various types of GNNs.

Chapter 2

Background

2.1 Deep Learning

2.1.1 Multi-Layer Perceptrons

Deep learning is a family of ML methods that aims at learning to approximate functions using artificial neural networks (ANNs). Deep neural networks (DNNs) are a class of ANNs with multiple layers between the input and output layers. Multilayer perceptrons (MLPs) are the most basic subset of DNNs: they are feedforward DNNs composed of fully connected layers. Each layer contains multiple perceptrons, which serve as elementary units in DNNs. A perceptron receives an input feature vector \mathbf{x} and applies it to a linear function defined by a weight vector \mathbf{w} and a bias b . The result is then modified by a non-linear activation function σ , as described by the following formula:

$$a = \sigma(z) = \sigma(\mathbf{w}^T \mathbf{x} + b) = \sigma \left(b + \sum_{i=1}^d w_i x_i \right) \quad (2.1)$$

Figure 2.1a illustrates the structure of a perceptron. The non-linear activation function is used to allow DNNs to compute non-trivial problems using only a small number of perceptrons or layers of perceptrons, as otherwise the composition of two linear functions is still linear. Some common activation functions include the logistic function (sigmoid), hyperbolic tangent (tanh), and rectified linear unit (ReLU).

In an MLP, data flow in only one direction: from the input perceptrons, through the perceptrons in the hidden layers, and finally to the output perceptrons. This process is also called forward propagation. There are no cycles in an MLP, and hence the data never go backward. The forward propagation can be expressed by the following equation:

$$\mathbf{h}^{(l+1)} = \sigma^{(l)} (\mathbf{W}^{(l)} \mathbf{h}^{(l)} + \mathbf{b}^{(l)}) \quad (2.2)$$

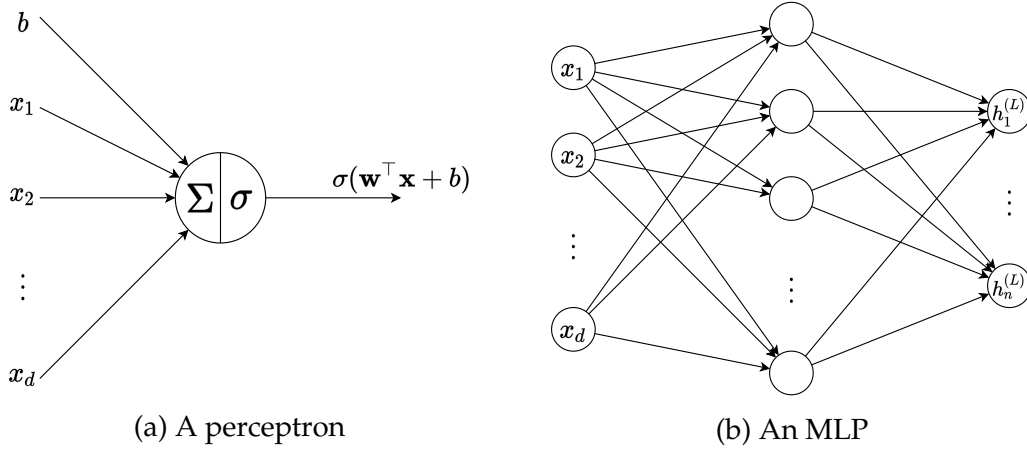


Figure 2.1: Example structures of a perceptron and an MLP

where $\mathbf{W}^{(l)} = [\mathbf{w}_1^{(l)} \ \dots \ \mathbf{w}_n^{(l)}]^\top$ and $\mathbf{b}^{(l)} = [b_1^{(l)} \ \dots \ b_n^{(l)}]^\top$ are the weight matrix and bias vector of the l -th layer, and $\mathbf{h}^{(l)}$ is the latent input of the l -th layer. An example MLP structure is illustrated in Figure 2.1b.

The universal approximation theorem (Hornik et al., 1989; Hornik, 1991) states that MLPs with a sufficient number of weights and at least one hidden layer layers are capable of approximating any continuous function. This also provides theoretical grounds for using MLPs as building blocks for GNNs.

2.1.2 Training

The training process of a DNN is done by iteratively adjusting the weights of the DNN in order to minimise a differentiable loss function \mathcal{L} . The DNN can do so by using stochastic gradient descent: starting with an appropriately initialised \mathbf{w}_0 , a mini-batch of m training examples is selected and fed into the DNN at each iteration; then, the DNN computes the output and the loss of the mini-batch via forward propagation, and the weights of the DNN are iteratively updated by a small amount in the direction of the negative gradient of \mathcal{L} , as described by the following formula:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \left. \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \right|_{\mathbf{w}_t} \quad (2.3)$$

where η is a small positive value known as the learning rate. The learning rate must be chosen carefully to obtain the best training outcome.

The gradient of the loss function with respect to the weights can be computed using the backpropagation algorithm: the gradients are propagated reversely, from the output perceptrons to the input perceptrons, through the perceptrons in the hidden layers, based on the chain rule:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial w} \quad (2.4)$$

The loss function must be chosen appropriately for each ML task, with sufficient mathematical grounds, in order to properly measure the DNN’s performance on the task. For example, for regression tasks, mean absolute error (MAE) or mean squared error (MSE) can be used as the loss function; for classification tasks, categorical cross-entropy can be used as the loss function; for an unsupervised learning task modelled by a variational autoencoder, a combination of reconstruction error and Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951) is chosen as the loss function (Kingma and Welling, 2014).

Various regularisation and optimisation techniques, including but not limited to dropout (Srivastava et al., 2014), batch normalisation (Ioffe and Szegedy, 2015) and adaptive learning rates such as Adam (Kingma and Ba, 2015), can be used in order to improve training performance.

2.2 Graph Neural Networks

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a collection of nodes \mathcal{V} and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ between pairs of nodes. Two graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ are isomorphic, denoted $\mathcal{G}_1 \cong \mathcal{G}_2$, if and only if there exists an edge-preserving bijective mapping $f : \mathcal{V}_1 \rightarrow \mathcal{V}_2$ between them, i.e.,

$$\forall u, v \in \mathcal{V}_1 . (u, v) \in \mathcal{E}_1 \iff (f(u), f(v)) \in \mathcal{E}_2 \quad (2.5)$$

In graph representation learning, a graph can be represented by a tuple of features $(\mathbf{X}, \mathbf{E}, \mathbf{A})$, where

- $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the node feature matrix of the graph, with each row $\mathbf{x}_u \in \mathbb{R}^d$ being the d -dimensional features of node u ;
- $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d_e}$ is the edge feature matrix of the graph, with each row $\mathbf{e}_{uv} \in \mathbb{R}^{d_e}$ being the d_e -dimensional features of edge (u, v) ; and
- $\mathbf{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ is the adjacency matrix of the graph, with each entry A_{uv} representing an edge between nodes u and v : $A_{uv} = 1$ if there exists an edge between nodes u and v (i.e., $(u, v) \in \mathcal{E}$), and 0 otherwise.

GNNs are a class of DNNs designed to operate on graph-structured data. The key idea of the GNNs is to generate representations of nodes and graphs that actually depend on the structures of the graphs, as well as their feature information (Hamilton, 2020).

2.2.1 Permutation Invariance and Equivariance

Since the key structural property of graphs is that the nodes in \mathcal{V} are usually not assumed to be provided in any particular order, any operations performed on graphs should not depend on the ordering of nodes. This makes the usual DNN architectures unsuitable for graphs, such as the convolutional neural networks (CNNs) designed for grid-structured inputs (for example, images), and the recurrent neural networks (RNNs) designed for sequences (for example, text). In order to design a GNN that is independent of the ordering of the nodes, while respecting individual node-wise transformation, the GNN model and its layers should maintain the following properties (Bronstein et al., 2021):

- Permutation invariance: a (graph-level) function f is permutation invariant if, for any permutation matrix \mathbf{P} ,

$$f(\mathbf{PX}, \mathbf{PAP}^\top) = f(\mathbf{X}, \mathbf{A}) \quad (2.6)$$

This means that permuting the nodes does not modify the results, and implies that for any two isomorphic graphs, the outcomes of f are identical.

- Permutation equivariance: a (node-level) function \mathbf{F} is permutation equivariant if, for any permutation matrix \mathbf{P} ,

$$\mathbf{F}(\mathbf{PX}, \mathbf{PAP}^\top) = \mathbf{PF}(\mathbf{X}, \mathbf{A}) \quad (2.7)$$

This means that the order of the rows of \mathbf{F} 's output is tied to the order of the rows of \mathbf{X} , so that each output node representation can consistently correspond to each input node.

An important constraint when designing deep learning models is locality, i.e., it is desirable to have the transformation stable under slight deformations (for example, shifts and distortions) of the domain. This can be enforced on graphs in the context of neighbourhoods: let the (1-hop) neighbourhood of node u , denoted \mathcal{N}_u , be defined as

$$\mathcal{N}_u = \{v \mid (u, v) \in \mathcal{E} \vee (v, u) \in \mathcal{E}\} \quad (2.8)$$

and the neighbourhood features of node u as the multiset

$$\mathbf{X}_{\mathcal{N}_u} = \{\{\mathbf{x}_v \mid v \in \mathcal{N}_u\}\} \quad (2.9)$$

Then, a permutation equivariant function \mathbf{F} enforcing locality on graphs can be constructed by applying a permutation invariant local function $\phi(\mathbf{x}_u, \mathbf{X}_{\mathcal{N}_u})$ to every node's neighbourhood in isolation.

2.2.2 Message Passing Neural Networks

Almost all existing GNNs can be abstracted as variants of Message Passing Neural Networks (MPNNs, Gilmer et al., 2017). In a generic MPNN, the message passing operation iteratively updates the features in every node from layer l to layer $l + 1$ via propagating messages through neighbouring nodes, obtaining the set of latent node features $\mathbf{h}_u^{(l)} \in \mathbb{R}^d$ for $u \in \mathcal{V}$ and $1 \leq l \leq L$. This can be formalised by the following equation:

$$\mathbf{h}_u^{(l+1)} = \text{UPDATE} \left(\mathbf{h}_u^{(l)}, \bigoplus_{v \in \mathcal{N}_u} \text{MESSAGE} \left(\mathbf{h}_u^{(l)}, \mathbf{h}_v^{(l)}, \mathbf{e}_{uv} \right) \right) \quad (2.10)$$

where

- MESSAGE and UPDATE are learnable functions, such as MLPs;
- \mathcal{N}_u is the neighbourhood of node u , as defined by Equation (2.8); and
- \bigoplus is a permutation invariant local neighbourhood function, such as sum, mean or max.

After L layers of message passing, the final node embeddings $\mathbf{h}_u^{(L)}$ are obtained. The graph embedding $\mathbf{h}_G \in \mathbb{R}^d$ can be computed from the node embeddings via a READ-OUT function:

$$\mathbf{h}_G = \text{READOUT} \left(\{ \{ \mathbf{h}_u^{(L)} \mid u \in \mathcal{V} \} \} \right) \quad (2.11)$$

The node and graph embeddings can then be used for any relevant downstream tasks. MPNNs draw an analogy with CNNs by considering its update operation as a convolutional aggregation across neighbourhoods of nodes. Examples of convolutional GNNs include Graph Convolutional Network (GCN, Kipf and Welling, 2017), Graph Isomorphism Network (GIN, Xu et al., 2019) and Principal Neighbourhood Aggregation (PNA, Corso et al., 2020), which are further described in Sections 2.2.4, 2.2.5 and 2.2.6.

2.2.3 Expressivity of Graph Neural Networks

The expressivity of a GNN can be defined as the ability to distinguish non-isomorphic graphs. Formally, *a GNN model A is strictly more expressive than another GNN model B, if A can distinguish all the pairs of attributed graphs that B can distinguish, and there there exists at least a pair of attributed graphs that A can distinguish but B cannot.* The expressivity of GNNs can be analysed by comparing to the Weisfeiler-Lehman (1-WL) graph isomorphism test (Weisfeiler and Leman, 1968). The 1-WL test is a simple heuristic for distinguishing most of the pairs of non-isomorphic graphs. Similar to GNNs, the 1-WL test iteratively updates the node labels (colours) of a graph by neighbourhood aggregation: for each node $u \in \mathcal{V}$ in a graph, an initial node colour $C_u^{(0)}$ is assigned, and is iteratively updated using random hashes of sums:

$$C_u^{(t+1)} = \text{HASH} \left(\sum_{v \in \mathcal{N}_u} C_v^{(t)} \right) \quad (2.12)$$

The 1-WL test terminates when stable node colouring of the graph is reached, and outputs a histogram of colours. Two graphs with different colour histograms are guaranteed to be non-isomorphic, and two graphs with the same colour histograms are possibly, but not necessarily, isomorphic. An example of execution of the 1-WL test on two isomorphic graphs is shown in Figure 2.2, and an example of non-isomorphic graphs that the 1-WL test fails to distinguish is shown in Figure 2.3.

Xu et al. (2019) have proved that any aggregation-based GNNs can only be as expressive as the 1-WL test. Although a GNN that share the same architecture as another GNN but with more parameters (for example, more hidden layer or larger hidden dimensions) is not mathematically strictly more expressive, it tend to be practically more expressive as it can capture more information of the graphs by having more parameters, and is therefore more capable of distinguishing non-isomorphic graphs, provided that it does not overfit the data.

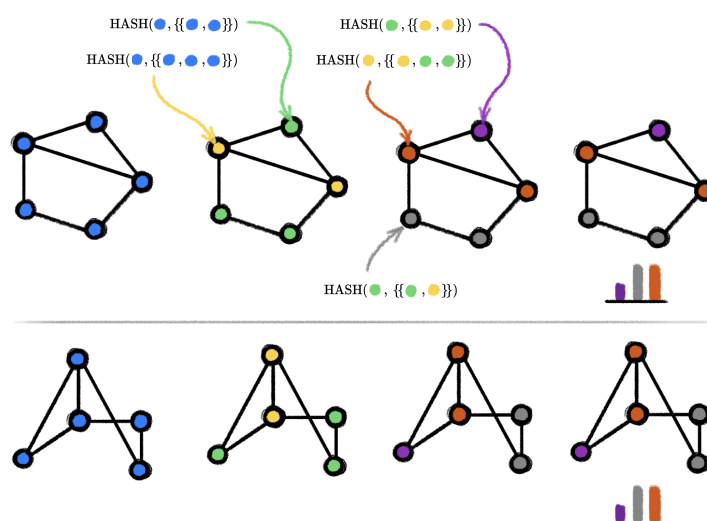


Figure 2.2: Example of execution of the 1-WL test on two isomorphic graphs. The algorithm stops after the colouring does not change and produces an output (histogram of colours). Equal outputs for the two graphs suggest that they are possibly isomorphic. Figure adapted from (Bronstein, 2020).

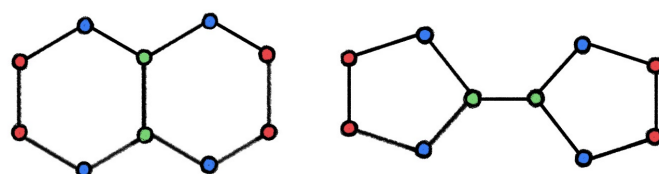


Figure 2.3: Two non-isomorphic graphs on which the 1-WL test fails, as evident from the identical colouring it produces. In chemistry, these graphs represent the molecular structure of two different compounds, decalin (left) and bicyclopentyl (right). Figure adapted from (Sato, 2020).

2.2.4 Graph Convolutional Networks

GCN is a simple and classic GNN architecture developed in the early stage, and is used as a baseline model in the experiments of this project. In a vanilla GCN, the layer-wise graph convolution operation on all node features is defined as

$$\mathbf{H}^{(l+1)} = \sigma \left(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(l)} \mathbf{W}^{(l)} \right) \quad (2.13)$$

where

- $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix of the graph with added self-connections;
- $\tilde{\mathbf{D}}$ is the corresponding degree matrix of $\tilde{\mathbf{A}}$: $\tilde{D}_{uu} = \sum_{v \in \mathcal{V}} \tilde{A}_{uv}$, and 0s elsewhere;
- $\mathbf{W}^{(l)} \in \mathbb{R}^{d \times d}$ is a layer-specific trainable weight matrix; and
- σ denotes a non-linear activation function.

This yields the corresponding node-wise propagation rule:

$$\mathbf{h}_u^{(l+1)} = \sigma \left(\sum_{v \in \mathcal{N}_u \cup \{u\}} c_{uv} \mathbf{W}^{(l)} \mathbf{h}_v^{(l)} \right) \quad (2.14)$$

where c_{uv} is a constant for each pair of nodes i and j :

$$c_{uv} = \sqrt{\frac{1}{(\deg(u) + 1)(\deg(v) + 1)}} \quad (2.15)$$

GCN is proved to be less expressive than the 1-WL test (Xu et al., 2019).

2.2.5 Graph Isomorphism Networks

GIN is the first attempt at developing a maximally-expressive GNN under the 1-WL limit, and serves as another baseline model in the experiments of this project. The update operation in a GIN layer is defined as

$$\mathbf{h}_u^{(l+1)} = \phi^{(l)} \left(\left(1 + \epsilon^{(l)}\right) \mathbf{h}_u^{(l)} + \sum_{v \in \mathcal{N}_u} \mathbf{h}_v^{(l)} \right) \quad (2.16)$$

where $\phi^{(l)}$ is an MLP, and $\epsilon^{(l)}$ is a learnable scalar. Instead of only using the final node embeddings for the graph-level representation, GIN computes the graph embedding by concatenating the node features across all intermediate layers (Xu et al., 2018):

$$\mathbf{h}_{\mathcal{G}} = \text{CONCAT} \left(\sum_{u \in \mathcal{V}} \mathbf{h}_v^{(l)} \mid l = 1, \dots, L \right) \quad (2.17)$$

GIN is provably as expressive as the 1-WL test, which makes it one of the maximally-expressive GNNs, according to Xu et al. (2019) and Corso et al. (2020).

2.2.6 Principal Neighbourhood Aggregation

PNA is a general and flexible architecture that enables the use of multiple aggregators concurrently, instead of using only a single aggregation method. PNA is used as the core model in all experiments of this project. Examples of PNA's aggregators include:

- Mean and sum aggregations:

$$\text{mean}_u(\mathbf{H}^{(l)}) = \frac{1}{\text{deg}(u)} \sum_{v \in \mathcal{N}_u} \mathbf{h}_v^{(l)} \quad \text{sum}_u(\mathbf{H}^{(l)}) = \sum_{v \in \mathcal{N}_u} \mathbf{h}_v^{(l)} \quad (2.18)$$

- Maximum and minimum aggregations:

$$\text{max}_u(\mathbf{H}^{(l)}) = \max_{v \in \mathcal{N}_u} \mathbf{h}_v^{(l)} \quad \text{min}_u(\mathbf{H}^{(l)}) = \min_{v \in \mathcal{N}_u} \mathbf{h}_v^{(l)} \quad (2.19)$$

- Standard deviation aggregation:

$$\text{std}_u(\mathbf{H}^{(l)}) = \sqrt{\text{ReLU}\left(\text{mean}_u\left((\mathbf{H}^{(l)})^2\right) - \text{mean}_u(\mathbf{H}^{(l)})^2\right) + \epsilon} \quad (2.20)$$

where ReLU is the rectified linear unit used to avoid negative values caused by numerical errors, and ϵ is a small positive value to ensure that the standard deviation is differentiable.

In addition, once the messages are aggregated, they are multiplied by multiple scaler functions to perform amplifications or attenuations of the incoming messages. Corso et al. (2020) propose the logarithm scaler for PNA, which can be generalised by the following function, with positive exponent α for amplification, negative for attenuation and zero for no scaling, and δ being a normalisation parameter computed over the training set:

$$\delta = \frac{1}{|\mathcal{G}_{\text{train}}|} \sum_{g \in \mathcal{G}_{\text{train}}} \sum_{u \in \mathcal{V}_g} \log(\text{deg}(u) + 1) \quad (2.21)$$

$$S(u, \alpha) = \left(\frac{\log(\text{deg}(u) + 1)}{\delta} \right)^\alpha \quad (-1 \leq \alpha \leq 1)$$

In the original PNA paper, the authors defined the overall aggregation function \oplus for PNA as four neighbourhood-aggregators with three degree-scalers each, as shown in

the following equation:

$$\bigoplus = \underbrace{\begin{bmatrix} \text{identity} \\ \text{amplification} \\ \text{attenuation} \end{bmatrix}}_{\text{scalers}} \otimes \underbrace{\begin{bmatrix} \text{mean} \\ \text{max} \\ \text{min} \\ \text{std} \end{bmatrix}}_{\text{aggregators}} = \underbrace{\begin{bmatrix} S(\mathcal{V}, \alpha = 0) \\ S(\mathcal{V}, \alpha = 1) \\ S(\mathcal{V}, \alpha = -1) \end{bmatrix}}_{\text{scalers}} \otimes \underbrace{\begin{bmatrix} \text{mean} \\ \text{max} \\ \text{min} \\ \text{std} \end{bmatrix}}_{\text{aggregators}} \quad (2.22)$$

where \otimes denotes the tensor product. The PNA operator can then be inserted into the standard MPNN framework, obtaining the following PNA layer:

$$\mathbf{h}_u^{(l+1)} = \phi^{(l)} \left(\mathbf{h}_u^{(l)}, \bigoplus_{v \in \mathcal{N}_u} \psi^{(l)}(\mathbf{h}_u^{(l)}, \mathbf{h}_v^{(l)}, \mathbf{e}_{uv}) \right) \quad (2.23)$$

where ψ, ϕ are MLPs. An optional residual or gated recurrent unit (GRU, Cho et al., 2014) connection can be added after each PNA layer, and Set2Set (Vinyals et al., 2016) is used as the readout function for obtaining the final graph embeddings.

PNA is practically more expressive than GIN by including more aggregators and therefore increasing the probability that at least one of the aggregators can distinguish different graphs, according to Corso et al. (2020).

2.3 Self-Supervised Learning

SSL aims to learn useful representations of the input data without relying on annotated labels. Typical training paradigms to apply self-supervision include unsupervised representation learning, unsupervised pre-training, and auxiliary learning (Xie et al., 2022):

- In unsupervised representation learning, only the unlabelled data is available for the entire training process. The unsupervised representation learning paradigm on graph-structured data $(\mathbf{X}, \mathbf{E}, \mathbf{A})$ can be formulated as

$$\begin{aligned} f_{\theta}^* &= \arg \min_{\theta} \mathcal{L}_{\text{SSL}}(\theta) \\ \mathbf{h}^* &= f_{\theta}^*(\mathbf{X}, \mathbf{E}, \mathbf{A}) \end{aligned} \quad (2.24)$$

The learned representations \mathbf{h}^* can then be used in further downstream tasks.

- In unsupervised pre-training, an encoder f_{θ} is trained first with unlabelled data. Then, the parameters θ of the pre-trained encoder is used as the initialisation of the encoder in a supervised fine-tuning task, together with a prediction head θ' .

Formally,

$$f_{\theta_{\text{init}}} = \arg \min_{\theta} \mathcal{L}_{\text{SSL}}(\theta) \quad (2.25)$$

$$f_{\theta, \theta'}^* = \arg \min_{\theta, \theta'} \mathcal{L}_{\text{SL}}(\theta, \theta')$$

For semi-supervised learning using this paradigm, the labelled graphs in the fine-tuning dataset are a subset of the pre-training dataset. For transfer learning, the pre-training and fine-tuning datasets are from different domains.

- In auxiliary learning, an auxiliary task under self-supervision is included to contribute to the main supervised learning task. The encoder is trained through both the main task and the auxiliary task simultaneously. This can be formulated as

$$f_{\theta, \theta'}^* = \arg \min_{\theta, \theta'} (\mathcal{L}_{\text{SL}}(\theta, \theta') + \lambda \mathcal{L}_{\text{SSL}}(\theta)) \quad (2.26)$$

where λ is a positive scalar weight that balances the two terms in the loss.

The three SSL training paradigms are illustrated in Figure 2.4.

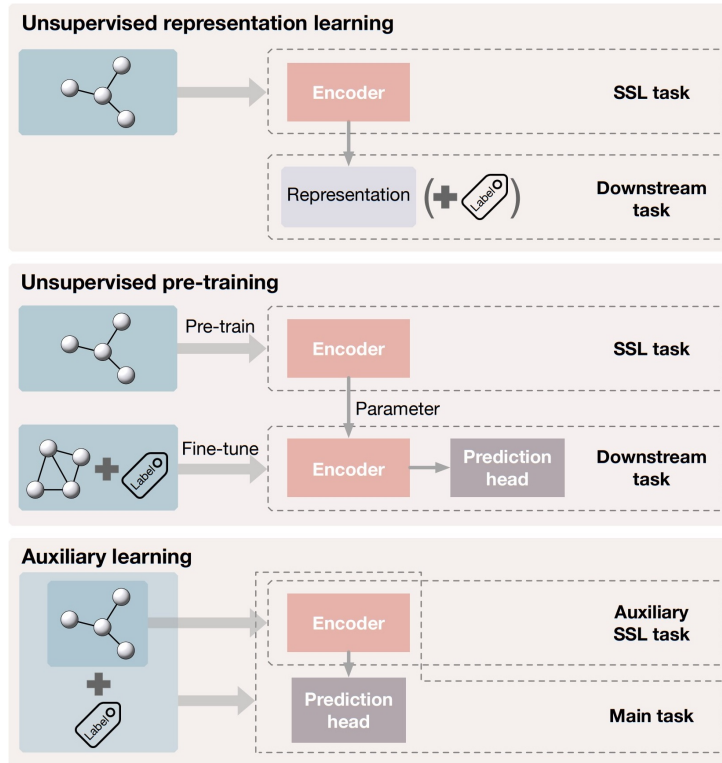


Figure 2.4: Paradigms for self-supervised learning. Top: in unsupervised representation learning, only the unlabelled are used to train the encoder through the SSL task. The learned representations are fixed and used in downstream tasks. Middle: unsupervised pre-training trains the encoder with unlabelled data by the SSL task. The pre-trained encoder’s parameters are then used as the initialization of the encoder for supervised fine-tuning in downstream tasks. Bottom: in auxiliary learning, an auxiliary SSL task is included to help learn the supervised main task. The encoder is trained through both the main task and the auxiliary task simultaneously. Figure adopted from (Xie et al., 2022).

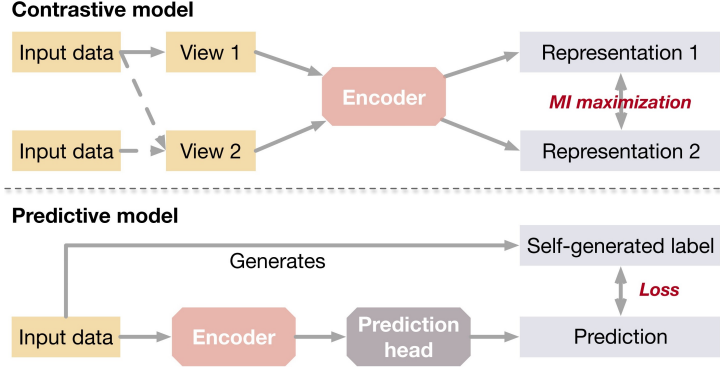


Figure 2.5: Comparison between contrastive learning and predictive learning. Figure adopted from (Xie et al., 2022).

2.3.1 Contrastive Learning

Based on how the training task is designed, SSL methods can be divided into two main categories: contrastive learning and predictive learning (Xie et al., 2022). The major difference between the two categories is that contrastive learning requires data-data pairs for training, whereas predictive learning adapts a more supervised fashion and requires data-label pairs, where the labels are self-generated from the data. The comparison between contrastive learning and predictive learning is illustrated in Figure 2.5. In the context of this project, only the contrastive learning framework is relevant.

In contrastive learning, given a set of unlabelled training data, one or more encoders are trained such that representations of similar data instances agree with each other, and representations of dissimilar data instances differ from each other. In general, given a graph-structured data $(\mathbf{X}, \mathbf{E}, \mathbf{A})$, multiple transformations T_1, \dots, T_K are applied to obtain different views $(\tilde{\mathbf{X}}_1, \tilde{\mathbf{E}}_1, \tilde{\mathbf{A}}_1), \dots, (\tilde{\mathbf{X}}_K, \tilde{\mathbf{E}}_K, \tilde{\mathbf{A}}_K)$ of the graph. Then a set of encoders f_1, \dots, f_K take corresponding views as their inputs and output the representations $\mathbf{h}_1, \dots, \mathbf{h}_K$ of the graph from the views, as summarised by the following equations: for $k = 1, \dots, K$,

$$\begin{aligned} (\tilde{\mathbf{X}}_k, \tilde{\mathbf{E}}_k, \tilde{\mathbf{A}}_k) &= T_k(\mathbf{X}, \mathbf{E}, \mathbf{A}) \\ \mathbf{h}_k &= f_k(\tilde{\mathbf{X}}_k, \tilde{\mathbf{E}}_k, \tilde{\mathbf{A}}_k) \end{aligned} \quad (2.27)$$

During training, the objective of contrastive learning is to discriminate jointly sampled view pairs (for example, two views belonging to the same instance) from independently sampled view pairs (for example, two views belonging to different instances). This can be abstracted as to maximise the agreement between representations computed from jointly sampled view pairs. The agreement is usually measured by the mutual information $I(\mathbf{h}_i; \mathbf{h}_j)$ between a pair of representations \mathbf{h}_i and \mathbf{h}_j , which is defined as

$$I(\mathbf{h}_i; \mathbf{h}_j) = D_{\text{KL}}(p(\mathbf{h}_i, \mathbf{h}_j) \parallel p(\mathbf{h}_i)p(\mathbf{h}_j)) = \mathbb{E}_{p(\mathbf{h}_i, \mathbf{h}_j)} \left[\log \frac{p(\mathbf{h}_i, \mathbf{h}_j)}{p(\mathbf{h}_i)p(\mathbf{h}_j)} \right] \quad (2.28)$$

where $D_{\text{KL}}(p||q)$ denotes the KL divergence from distribution p to distribution q . In order to computationally estimate the mutual information, various lower-bounds to the mutual information are derived, including the Donsker-Varadhan representation \hat{I}_{DV} (Donsker and Varadhan, 1983), the Jensen-Shannon estimator \hat{I}_{JS} (Nowozin et al., 2016), and the noise-contrastive estimation \hat{I}_{NCE} (Gutmann and Hyvärinen, 2010).

Chapter 3

Related work

3.1 General Advances in Contrastive Self-Supervised Learning

As far as I am aware, there has not been any published attempt to develop a method for evaluating DNN models by directly competing two DNNs in a contrastive self-supervised environment, no matter in the general domain or the graph domain. To my knowledge, all works that develop new deep learning methods tend to compete against other methods by comparing their performances on one or several datasets, without setting up an environment for the methods to directly compete against each other. However, the state-of-the-arts in contrastive SSL, both in the non-graph domain and the graph domain, are still relevant to this project, as their successes in building contrastive learning architectures can serve as valuable references. It should be emphasised that, since this project aims at task-agnostic evaluations on GNNs rather than optimising downstream tasks, the performances of the state-of-the-arts in contrastive SSL on downstream tasks are not relevant to this project.

Generative adversarial networks (GANs, Goodfellow et al., 2014) are perhaps one of the most classic contrastive SSL framework. GANs are based on a game in which a generator network produces synthetic samples, and a discriminator network acts as its adversary and attempts to distinguish between samples drawn from the training data and samples produced by the generator network. The core idea of GANs is to train the generator network to fool the discriminator. Unfortunately, in general, simultaneous gradient descent on two networks' losses is not guaranteed to reach an equilibrium. This causes many variants of GANs to suffer from non-convergence and high sensitivity to the hyperparameter selections (Goodfellow, 2015; Arjovsky et al., 2017), and stabilising GANs learning still remains an open problem.

After GANs, another class of contrastive SSL methods have emerged, based on the principle of maximising the mutual information between two representations of the same data, in possibly different views, by two separate DNNs. Oord et al. (2018) designed the noise contrastive estimation of mutual information loss called InfoNCE, which maximises a lower bound of the mutual information, based on the noise-contrastive estimation (Gutmann and Hyvärinen, 2010). InfoNCE is a popular objective for contrastive SSL, and is used by many famous works in computer vision such as Deep InfoMax (DIM, Hjelm et al., 2019), Momentum Contrast (MoCo, He et al., 2020) and SimCLR (Chen et al., 2020). All those works generate positive pairs (i.e., similar datapoints) by manually applying small augmentations to the datapoints, especially with SimCLR showing the importance of using appropriate augmentations, large batch sizes and non-trivial negative pairs (for example, images that look very similar but have different labels) in building successful contrastive SSL methods.

The above contrastive SSL works are prone to information collapse, in which the two networks ignore the input data and only produce identical and constant output vectors. In order to prevent information collapse, those works rely on large batch sizes or memory banks, and extensive searches of appropriate augmentations and mining functions for retrieving negative pairs, and tend to be very costly. Recently, there emerged an alternative class of collapse prevention methods by maximising the information contents within the representations, including Barlow Twins (Zbontar et al., 2021) and Variance-Invariance-Covariance Regularisation (VICReg, Bardes et al., 2022). These works provide more principled collapse prevention by decorrelating the features within the embedding vectors, and forcing the representations of the samples within a batch to be different. By doing so, they implicitly maximise the information content within the representation vectors. Barlow Twins and VICReg are the core references for this project, and will be described in detail in the following subsections.

However, applying handcrafted augmentations to the training data introduces arbitrary human knowledge not provided by the training data, and deviates the data from real-world distributions. Therefore, it is highly desired to build a principled contrastive SSL framework without the need for handcrafted augmentations, which is tackled by this project.

3.1.1 Barlow Twins: Self-Supervised Learning via Redundancy Reduction

In Barlow Twins, for a given input batch \mathbf{X} of size N_b , two batches of distorted views $\tilde{\mathbf{X}}_A$ and $\tilde{\mathbf{X}}_B$ of \mathbf{X} are obtained via data augmentation. The two batches of distorted views $\tilde{\mathbf{X}}_A$ and $\tilde{\mathbf{X}}_B$ are then fed into two separate DNNs, producing batches of d -dimensional embeddings \mathbf{H}_A and \mathbf{H}_B respectively. To simplify notations, the features in both \mathbf{H}_A and \mathbf{H}_B are assumed to have zero mean over the batch. Barlow Twins then computes

the cross-correlation matrix between \mathbf{H}_A and \mathbf{H}_B along the batch dimension:

$$C_{ij} = \frac{\sum_b^{N_b} (H_A)_{bi} (H_B)_{bj}}{\sqrt{\sum_b^{N_b} ((H_A)_{bi})^2} \sqrt{\sum_b^{N_b} ((H_B)_{bj})^2}} \quad (3.1)$$

where b indexes batch samples and i, j index the features of the embeddings. This is equivalent to the following matrix operation:

$$\mathbf{C} = \left(\frac{\mathbf{H}_A}{\sqrt{(\mathbf{H}_A)^2}} \right)^\top \left(\frac{\mathbf{H}_B}{\sqrt{(\mathbf{H}_B)^2}} \right) = \frac{1}{N_b} \left(\frac{\mathbf{H}_A}{\text{std}(\mathbf{H}_A)} \right)^\top \left(\frac{\mathbf{H}_B}{\text{std}(\mathbf{H}_B)} \right) \quad (3.2)$$

The cross-correlation matrix \mathbf{C} is a square matrix with the same dimensionality as the output embeddings, and values between -1 (perfect anti-correlation) and 1 (perfect correlation). Barlow Twins then applies the following loss function on the cross-correlation matrix:

$$\mathcal{L}_{\text{BT}} = \underbrace{\sum_i^d (1 - C_{ii})^2}_{\text{invariance term}} + \lambda \underbrace{\sum_i^d \sum_{j \neq i}^d C_{ij}^2}_{\text{redundancy reduction term}} \quad (3.3)$$

The invariance term of the Barlow Twins loss enforces the two output embeddings to be similar, by pushing the on-diagonal elements of the cross-correlation matrix towards 1. The redundancy reduction term tries to make the off-diagonal elements of the cross-correlation matrix closer to 0, and hence decorrelates the different features of the embeddings, so that the embeddings contain non-redundant information about the data.

Usually, the two DNNs in Barlow Twins are identical. However, in this project, two different DNNs are used. Although the authors of Barlow Twins notice a decrease in performance when introducing asymmetries into Barlow Twins, this project can still successfully discriminate different GNNs by their expressivity, though a modified objective function from Barlow Twins (details discussed in later chapters).

3.1.2 VICReg: Self-Supervised Learning via Variance-Invariance-Covariance Regularisation

Similar to Barlow Twins, VICReg is built based on the principle of preserving the information content of the representations. The architecture of VICReg is the same as Barlow Twins, except that it uses three regularisation terms in its objective function (using the same notations as in the previous subsection):

- Variance regularisation: a hinge loss to maintain the standard deviation of the embeddings along the batch dimension close to 1:

$$\text{Var}(\mathbf{H}) = \frac{1}{d} \sum_i^d \max(0, 1 - \text{std}(\mathbf{H}_{:,i})) \quad (3.4)$$

This term forces the output embeddings within a batch to be different.

- Invariance regularisation: the mean square Euclidean distance between the output embeddings:

$$\text{Inv}(\mathbf{H}_A, \mathbf{H}_B) = \frac{1}{N_b} \sum_b^{N_b} \|(\mathbf{h}_A)_b - (\mathbf{h}_B)_b\|_2^2 \quad (3.5)$$

- Covariance regularisation: a term that attracts the covariances between every pair of features of the embeddings over a batch towards 0: define the covariance matrix of mean-centred embeddings \mathbf{H} as

$$\mathbf{C}(\mathbf{H}) = \frac{1}{N_b - 1} \mathbf{H}^\top \mathbf{H} \quad (3.6)$$

Then, inspired by Barlow Twins, the covariance regularisation term can be defined as the sum of the squared off-diagonal elements of the covariance matrix, with a factor $\frac{1}{d}$ to scale the term as a function of the feature dimension:

$$\text{Cov}(\mathbf{H}) = \frac{1}{d} \sum_i^d \sum_{j \neq i}^d C(\mathbf{H})_{ij}^2 \quad (3.7)$$

This term decorrelates the different features of the embeddings, and thus prevents them from encoding similar information.

The overall loss function for VICReg is a weighted sum of the invariance, variance and covariance regularisation terms:

$$\mathcal{L}_{\text{VICReg}} = \lambda \underbrace{\text{Inv}(\mathbf{H}_A, \mathbf{H}_B)}_{\mathcal{L}_{\text{Inv}}} + \mu \underbrace{(\text{Var}(\mathbf{H}_A) + \text{Var}(\mathbf{H}_B))}_{\mathcal{L}_{\text{Var}}} + \nu \underbrace{(\text{Cov}(\mathbf{H}_A) + \text{Cov}(\mathbf{H}_B))}_{\mathcal{L}_{\text{Cov}}} \quad (3.8)$$

where $\lambda, \mu, \nu > 0$ are hyperparameters controlling the importance of each term in the loss. In this project, the invariance regularisation term of VICReg has other substitutes, and the variance regularisation term is not needed, because increasing the variance of the embeddings over a batch can potentially make the training unstable. The covariance regularisation term is very valuable to this project, as it helps strengthen the decorrelation of different features by combining with the redundancy reduction term in Barlow Twins. Details of how VICReg is incorporated in this project are explained in future chapters.

3.2 Contrastive Self-Supervised Learning on Graphs

SSL on graphs has attracted a lot of attention in recent years, and there has been a trend to extend the success of SSL on non-graph tasks to graph-structured data. Xie et al. (2022) provided a complete review of SSL works for GNNs up to its publication date. Amongst the SSL works on graphs, only the contrastive learning methods are relevant to this project. Here I reiterate that, as the core target for this project is to build task-agnostic evaluation of GNNs using SSL, the performances of the existing SSL works on downstream tasks are not relevant to this project.

Veličković et al. (2019) developed Deep Graph Infomax (DGI), a node-level SSL method on graphs, by adapting the ideas from DIM to the graph domain. Sun et al. (2020) then extended DIM to graph-level representations and proposed InfoGraph. Their successes inspired many works on contrastive SSL on graphs by applying augmentations to graphs followed by mutual information maximisation, with varying augmentation strategies and mutual information estimators.

Applying augmentations to graphs can be much harder than on images, because there exists no universal augmentation that works across all types of graphs. Besides, graphs are not invariant to noise like images, and even a small alternation on a graph can significantly change its topological structure, especially for small graphs such as molecules. Graph Contrastive Learning (GraphCL, You et al., 2020) responds to this by conducting a complete evaluation of augmentations on graphs in a comprehensive framework. An extension work on GraphCL (You et al., 2021) proposes a unified bilevel optimization framework to automate the augmentation selection process when performing GraphCL on specific graph-structured data. The current best-performing contrastive SSL work on molecular graphs is GraphLoG (Xu et al., 2021), which introduces hierarchical prototypes to capture the global semantic clusters, in addition to preserving the local similarities between positive pairs.

However, although the above works have developed fine-grained augmentations on graphs, it is still almost impossible for them to apply augmentations while preserving the graph's intrinsic properties, such as the chemical properties of molecules. Therefore, it is even more desired in the graph domain to build a principled contrastive SSL framework without handcrafted augmentations. 3D Infomax (Stärk et al., 2021) proposes such a framework by maximising the mutual information between the embedding of a 2D molecular graph and the embedding capturing its 3D information, produced by two separate GNNs. However, 3D Infomax specifically relies on the physical properties of molecules, and cannot be generalised to other domains. This project is the first ever to develop a contrastive self-supervised framework that does not require augmentations across a diversity of graph types, which possesses a high novelty in this area.

Chapter 4

Design and Implementation

4.1 Proposed Frameworks

In this project, I designed and implemented GraphAC (Graph Adversarial Collaboration), a conceptually novel, principled, and task-agnostic framework for evaluating GNNs in a self-supervised, adversarial collaboration manner, without the need of handcrafted augmentations. The intuition behind GraphAC is to have different GNNs competing against each other on the same unlabelled graphs, and make the more expressive GNNs to produce more complex and informative graph embeddings. This can be measured by the ability to predict other GNNs' graph embeddings from a GNN's own graph embeddings: if a GNN can predict another GNN's graph embeddings from its own graph embeddings better than the other way round, then its graph embeddings can be deemed more complex and informative than the other GNN's graph embeddings, and therefore, it can also be deemed more expressive than the other GNN. The main challenge for this project is to design a learning framework that maximises the performance differences between different GNNs, while ensuring stable training at the same time.

4.1.1 Contestant-Judge Framework: GNNs + MLPs

Initially, I designed GraphAC as a contestant-judge framework: it consists of two GNN-MLP pairs, in which the GNNs act as the contestants that produce embeddings for unlabelled graphs, and the MLPs act as the judges that evaluate the produced graph embeddings. During training, in each iteration, for the same unlabelled graph, each GNN produces a graph embedding. Then, its corresponding MLP in the GNN-MLP pair is trained to predict the other GNN's graph embedding from its own graph embedding, obtaining a loss value. The GNN then updates its weights and seeks to maximise the weighted difference between the opponent's prediction loss and its own

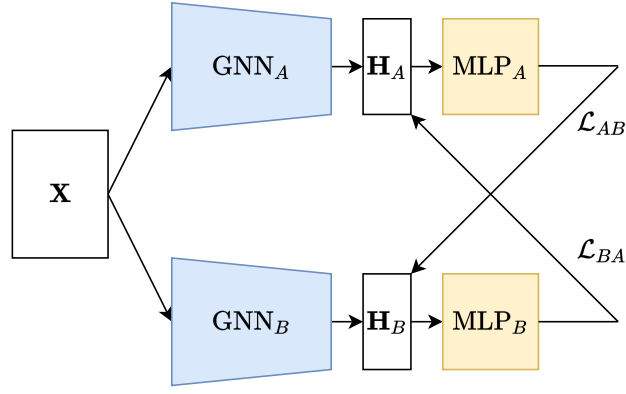


Figure 4.1: Architecture of the contestant-judge framework

prediction loss. At the end of training, the GNN-MLP pair with the lower prediction loss wins, and the GNN in that pair can be deemed more expressive.

Formally, let $(\text{GNN}_A, \text{MLP}_A)$ and $(\text{GNN}_B, \text{MLP}_B)$ be two GNN-MLP pairs. For each graph-structured input $(\mathbf{X}, \mathbf{E}, \mathbf{A})$, let

- \mathbf{h}_A be the output graph embedding by GNN_A ;
- \mathbf{h}_B be the output graph embedding by GNN_B ;
- $\hat{\mathbf{h}}_B$ be the prediction of \mathbf{h}_B by MLP_A from \mathbf{h}_A , with prediction loss \mathcal{L}_{AB} ; and
- $\hat{\mathbf{h}}_A$ be the prediction of \mathbf{h}_A by MLP_B from \mathbf{h}_B , with prediction loss \mathcal{L}_{BA} .

Then the loss functions of the models are defined as follows:

$$\begin{aligned}
 \mathcal{L}_{\text{MLP}_A} &= \mathcal{L}_{AB} = \text{MSE}(\mathbf{h}_B, \hat{\mathbf{h}}_B) \\
 \mathcal{L}_{\text{MLP}_B} &= \mathcal{L}_{BA} = \text{MSE}(\mathbf{h}_A, \hat{\mathbf{h}}_A) \\
 \mathcal{L}_{\text{GNN}_A} &= \mathcal{L}_{AB} - \lambda \mathcal{L}_{BA} \\
 \mathcal{L}_{\text{GNN}_B} &= \mathcal{L}_{BA} - \lambda \mathcal{L}_{AB}
 \end{aligned} \tag{4.1}$$

where $\lambda > 0$ is a weighting coefficient. If $\lambda = 1$, the GNNs are set to make equal effort to predict the other GNNs' graph embeddings (collaboration) and to prevent the other GNNs from predicting their own graph embeddings (competition). If $\lambda < 1$, the GNNs are set to prioritise on collaboration. If $\lambda > 1$, the GNNs are set to prioritise on competition. The value of λ must be carefully chosen to obtain meaningful training outcome: if λ is too small, the GNNs will only mimic other GNNs and may even aim at producing trivial and constant graph embeddings; if λ is too large, the GNNs will bypass the training target by producing random embeddings or randomly permuting the entries of its output graph embeddings. In both cases, the GNNs will not try to learn useful information from the graphs. The architecture of such a contestant-judge framework is illustrated in Figure 4.1. For fair competition, the MLPs should share the same structure, but not necessarily with the same weights, and both $\mathcal{L}_{\text{GNN}_A}$ and $\mathcal{L}_{\text{GNN}_B}$ should have the same λ value.

During the experiments, it is observed that the GNNs in this contestant-judge framework may try to bypass the training targets by blindly enlarging the entries of their output graph embeddings without learning from the graph information, causing unstable and fruitless training. In order to stabilise training, I attempted to add a weighted Barlow Twins term and a covariance regularisation term from VICReg in the above loss functions. I include Barlow Twins in my loss functions because its invariance term forces \mathbf{h}_A and \mathbf{h}_B to be similar, and within the context of this framework, its redundancy reduction term decorrelates different feature dimensions between the two output graph embeddings, and hence forces the two GNNs to capture the graph information independently, without copying from the other GNNs’ output. I include the covariance regularisation term from VICReg in my loss functions because it decorrelates different feature dimensions within each output graph embeddings, and forces the graph embeddings to be fully used to capture graph information. The invariance regularisation term from VICReg is not included in my loss functions, because the effect of the invariance regularisation term has already been achieved by the MSE loss and the invariance term from Barlow Twins. I also do not include the variance regularisation term from VICReg in my loss functions, because it forces the variance of the embeddings over a batch to be above a given threshold, which can potentially cause the training to be even more unstable. The detailed explanations of the effects of different terms in Barlow Twins and VICReg are described in Section 3.1. Adding the Barlow Twins and VICReg covariance regularisation terms yields the adjusted loss functions as follows:

$$\begin{aligned}
\mathcal{L}_{\text{MLP}_A} &= \mathcal{L}_{AB} = \text{MSE}(\mathbf{h}_B, \hat{\mathbf{h}}_B) \\
\mathcal{L}_{\text{MLP}_B} &= \mathcal{L}_{BA} = \text{MSE}(\mathbf{h}_A, \hat{\mathbf{h}}_A) \\
\mathcal{L}_{\text{GNN}_A} &= \alpha(\mathcal{L}_{AB} - \lambda\mathcal{L}_{BA}) + \beta\mathcal{L}_{\text{BT}} + \gamma\mathcal{L}_{\text{Cov}} \\
\mathcal{L}_{\text{GNN}_B} &= \alpha(\mathcal{L}_{BA} - \lambda\mathcal{L}_{AB}) + \beta\mathcal{L}_{\text{BT}} + \gamma\mathcal{L}_{\text{Cov}}
\end{aligned} \tag{4.2}$$

where $\alpha, \beta, \gamma > 0$ are weighting coefficients, and $\mathcal{L}_{\text{BT}}, \mathcal{L}_{\text{Cov}}$ are the Barlow Twins and VICReg covariance regularisation terms defined in Section 3.1. The loss functions defined by Equation (4.1), which do not contain the Barlow Twins and VICReg covariance regularisation terms, can also be expressed by the above loss functions by setting $\alpha = 1$ and $\beta = \gamma = 0$.

4.1.2 Competitive Barlow Twins

Despite that various optimisation methods have been tried during the experiments, it still shows that the contestant-judge framework described in the previous section suffers from unstable training (further details are described in Sections 4.3.2 and 5.2.1). Therefore, I designed another learning framework, which proved to be the final successful learning framework for GraphAC. The new framework does not need MLP judges to explicitly predict between the GNNs’ output graph embeddings. Instead, it uses a

novel pair of loss functions modified from the Barlow Twins described in Section 3.1.1, which I call the competitive Barlow Twins.

A deeper analysis of the Barlow Twins shows that, according to Equation (3.1) defined in Section 3.1.1,

$$C_{ij} = \frac{\sum_b^{N_b} (H_A)_{bi} (H_B)_{bj}}{\sqrt{\sum_b^{N_b} ((H_A)_{bi})^2} \sqrt{\sum_b^{N_b} ((H_B)_{bj})^2}} \quad (3.1)$$

the (i, j) -th entry C_{ij} of the cross-correlation matrix represents how much feature i of the first model's output embeddings \mathbf{h}_A correlates to feature j of the second model's output embeddings \mathbf{h}_B . Therefore, for output embeddings of dimensionality d , the row $\mathbf{C}_{i,[i+1:d]}$ at the upper-triangle of the cross-correlation matrix represents how much feature i of \mathbf{h}_A correlates to features $i + 1$ to d of \mathbf{h}_B . For i close to 1, the row $\mathbf{C}_{i,[i+1:d]}$ in the upper-triangle becomes much longer, and thus the i -th feature of \mathbf{h}_A represented by that piece of the row correlates to the majority of the features of \mathbf{h}_B . For i close to d , the row at the upper-triangle becomes much shorter, and thus the i -th feature of \mathbf{h}_A represented by that piece of the row correlates to very few features of \mathbf{h}_B . This means that the smaller-indexed features of the first model's output embeddings \mathbf{h}_A become the more important features, if monitored by the upper-triangle of the cross-correlation matrix. Similarly, in the lower-triangle of the cross-correlation matrix, the column $\mathbf{C}_{[j+1:d],j}$ represents how much feature j of \mathbf{h}_B correlates to features $j + 1$ to d of \mathbf{h}_A , making the smaller-indexed features of the second model's output embeddings also becoming the more important features. It can therefore be hypothesised that under this upper-lower-triangle setting, the first few features of both models' output embeddings are targeted at capturing the low frequency signals as they are easier to predict, and the later features are set to capture the high frequency signals, which are harder to predict.

Based on the above findings, if the two triangles of the cross-correlation matrix are summed, then the sum of each triangle is dominated by the first few rows/columns, as they contain the most entries. Therefore, the sum of the triangle provides a measure of how much a model's output features correlate to the other model's output features, weighted by importance, since there are more elements in the triangle corresponding to the more important features. Consequently, a larger sum implies a better correlation of a model's most important features in its output embeddings with the other model's output features, which implies a stronger ability to predict the other model's output embeddings from its own output embeddings. This naturally yields the definition of the competitive Barlow Twins loss, which preserves the invariance term in the original Barlow Twins, but replaces the off-diagonal sum with the difference between the upper-

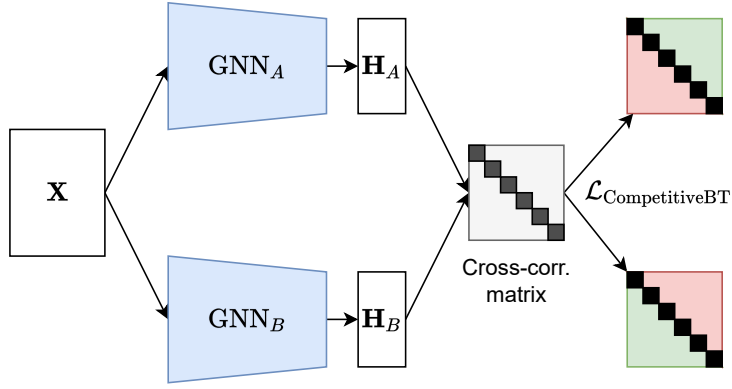


Figure 4.2: Architecture of the competitive Barlow-Twins framework

triangle and the lower-triangle of the cross-correlation matrix:

$$\begin{aligned}
 \mathcal{L}_{\text{CompetitiveBT}_A} &= \sum_i^d (1 - C_{ii})^2 + \lambda \left(\sum_i^d \sum_{j>i}^d C_{ij}^2 - \mu \sum_j^d \sum_{i>j}^d C_{ij}^2 \right) \\
 \mathcal{L}_{\text{CompetitiveBT}_B} &= \sum_i^d (1 - C_{ii})^2 + \lambda \left(\sum_j^d \sum_{i>j}^d C_{ij}^2 - \mu \sum_i^d \sum_{j>i}^d C_{ij}^2 \right)
 \end{aligned} \tag{4.3}$$

where $\lambda, \mu > 0$ are weighting coefficients, with λ inherited from the original Barlow Twins, and μ trading off the importance of correlating the opponent GNN's output features (collaboration) and preventing the opponent GNN from correlating the GNN's own output features (competition). Although the above reasoning discourages the use of different weights on the sums of triangles, which is also confirmed by the hyperparameter tuning results described in Section 5.2.2, I still include μ in my definition of the competitive Barlow Twins for the purpose of hyperparameter tuning.

Another important enhancement by the competitive Barlow Twins is that, since the triangles make the smaller-indexed features of both models' output embeddings the more important features, both models' output embeddings are ordered by feature importance. This ordering prevents the models from simply permuting the entries of their output embeddings to avoid being predicted by their opponent models, and makes the training much more stable.

The architecture of the competitive Barlow Twins framework is illustrated in Figure 4.2. For the similar reasons in the contestant-judge framework, I also include a covariance regularisation term from VICReg in the final loss functions for the GNNs, obtaining the following definitions:

$$\begin{aligned}
 \mathcal{L}_{\text{GNN}_A} &= \alpha \mathcal{L}_{\text{CompetitiveBT}_A} + \beta \mathcal{L}_{\text{Cov}} \\
 \mathcal{L}_{\text{GNN}_B} &= \alpha \mathcal{L}_{\text{CompetitiveBT}_B} + \beta \mathcal{L}_{\text{Cov}}
 \end{aligned} \tag{4.4}$$

where $\alpha, \beta > 0$ are weighting coefficients, and \mathcal{L}_{Cov} is the VICReg covariance regularisation term defined in Section 3.1.2. Although the competitive Barlow Twins losses

can enable stable training of the models, and can counter the instability caused by the VICReg variance regularisation term, I still do not include the variance regularisation term in the loss functions, because it is used to prevent the models from producing the same embedding vectors for samples within a batch, which did not occur in this framework.

4.2 Datasets

Wu et al. (2018), Dwivedi et al. (2020) and Hu et al. (2020) proposed a range of datasets across different artificial and real-world tasks, which have now become widely accepted benchmarks in the GNN community. They have demonstrated that the appropriate graph datasets that are able to statistically distinguish the performance of GNNs need to be representative, realistic and large-scale. This guides me to set the following requirements for the datasets to be used for GraphAC:

- the datasets should be application-oriented with real-world implications, in order for GraphAC to provide the most realistic evaluations of the GNNs;
- the datasets should be large-scale of high quality, in order for GraphAC to discriminate between GNNs with statistical significance;
- since GraphAC is designed for graph-level prediction, the datasets should also be constructed graph-level prediction, which means that the datasets should contain a large number of relatively small graphs, as if the sizes of the graphs are too large, it would be very computationally resource-heavy for the project, and require extensive GPU resources;
- the datasets should provide both node and edge features for the graphs, in order for GraphAC to support GNNs both with and without edge features, and allow it to study the effect for a GNN to include the edge features.

Based on the above requirements, the drug-like small molecular datasets are the most suitable datasets for GraphAC, with the following reasons:

- molecules can naturally be represented as graphs;
- molecular property prediction is a fundamental task within many important applications in chemistry;
- there is a vast variety of molecules in the world, and the drug-like small molecular graphs can be trained efficiently without requiring extensive GPU resources.

Therefore, the following widely acknowledged molecular datasets are selected for GraphAC:

OGB graph property prediction The Open Graph Benchmark (OGB, Hu et al., 2020) library provides the following datasets for graph-level prediction, as listed in Table 4.1:

Table 4.1: Statistics of the OGB graph property prediction datasets

Name	#Graphs	Avg. #nodes per graph	Avg #edges per graph	Avg. node degree	Graph type
ogbg-molhiv	41,127	25.5	27.5	2.2	Molecule
ogbg-molpcba	437,929	26.0	28.1	2.2	Molecule
ogbg-ppa	158,100	243.4	2266.1	18.3	Protein
ogbg-code2	452,741	125.2	124.2	2.0	Syntax tree

Among those datasets, the `ogbg-molpcba` dataset is the most suitable dataset for GraphAC, because it is the largest molecular dataset provided by OGB that is not a part of the large-scale challenge (OGB-LSC, Hu et al., 2021). Although the `ogbg-code2` dataset contains slightly more graphs than `ogbg-molpcba`, the graphs in the `ogbg-molpcba` dataset generally have a smaller size, which is already adequate for GraphAC evaluations, and can enable efficient training. The `ogbg-molpcba` dataset is used by GraphAC for all GNN evaluation experiments, which are elaborated in Chapter 5.

ZINC ZINC (Irwin et al., 2012) is one of the most popular real-world molecular dataset of 249,456 drug-like molecules with 28 atom (node) types and 4 bond (edge) types, and its graphs have sizes ranging from 6 to 38 nodes. The average number of nodes in the graphs of the ZINC dataset is 23.2, and the average number of edges is 49.8. The ZINC dataset is split into 220,011 train, 24,445 validation and 5,000 test graphs. The task of the ZINC dataset is to regress a molecular property known as the constrained solubility. The ZINC dataset is used by GraphAC for fine-tuning, as an optional extension for this project.

4.3 Implementation

4.3.1 Starting Point

The source code of GraphAC is developed on top of the 3D Infomax repository by Stärk et al. (2021)¹, which provides a comprehensive implementation of different GNNs, a flexible training environment with necessary utility functions such as losses and visualisations, and a thorough configuration setup that enables large-scale training in parallel. All frameworks and experiments are implemented using PyTorch (Paszke et al., 2019), and PyTorch Geometric (PyG, Fey and Lenssen, 2019) and Deep Graph Library (DGL,

¹<https://github.com/HannesStark/3DInfomax>

Wang et al., 2019) are used as the deep graph representation learning libraries for this project. The OGB datasets are directly adopted from the OGB library, and the implementation of the ZINC dataset is based on Dwivedi et al. (2020)’s source code repository.² All relevant licence requirements of the imported open-source libraries have been carefully observed. The source code for this project is publicly available at <https://github.com/VictorZXY/GraphAC>.

4.3.2 Contestant-Judge Framework

The core pseudocode in a PyTorch style for the contestant-judge framework is shown in Algorithm 2. The experiments show that if both GNNs are updated every iteration, they would both try to update their output graph embeddings to respond to the opponents’ graph embeddings produced in the immediate preceding iteration, causing the training to be unstable. In order to stabilise training, in each iteration, I only update one of the GNNs so that it can fit its opponent’s output graph embeddings without disturbance, and switch the GNNs every fixed iterations. This is inspired by the deep Q-network (DQN, Mnih et al., 2015) which includes a target network that is only periodically updated to improve stability. The weights of the MLPs are not frozen, because they need to update their weights in accordance with the new output graph embeddings of their corresponding GNNs. The PyTorch-style pseudocode for this periodic training paradigm is shown in Algorithm 1 in Appendix B.1. Various additional techniques have also been used in the actual implementation to stabilise training, such as normalising the output embeddings by both the GNNs and MLPs, and gradient clipping. These techniques are not included in the pseudocode, as they are not a part of the core learning algorithm.

4.3.3 Competitive Barlow Twins

The PyTorch-style pseudocode for the competitive Barlow Twins framework is shown in Algorithm 3 in Appendix B.2. Unlike the contestant-judge framework, the GNNs in the competitive Barlow Twins framework are not required to explicitly predict the other GNNs’ output graph embeddings. Therefore, there is no need to freeze one of the GNNs and switch between GNNs every fixed intervals like in the contestant-judge framework. Besides, since the pair of competitive Barlow Twins losses involve subtractions between the triangle elements on opposite sides, periodically freezing one of the GNNs during training is inappropriate for the competitive Barlow Twins setup as this would break the symmetry. Gradient clipping can still be included as a technique for stabilising training, but there is no need to normalise the GNNs’ output graph embeddings, as they are already normalised in the computations of competitive Barlow Twins and VICReg covariance regularisation.

²<https://github.com/graphdeeplearning/benchmarking-gnns>

Chapter 5

Experiments and Evaluation

5.1 Experimental Setup

The experiments and evaluations of GraphAC were conducted on two different machines. Developments, unit testing and small-scale trial experiments were carried out on my personal laptop, which is a Microsoft Surface Book 2 that has an Intel Core i7-8650U 4-core CPU @ 1.90GHz, 16GB RAM, 256GB SSD, and an NVIDIA GeForce GTX 1060 GPU with 6GB graphics memory. It runs Windows 11 with Ubuntu 20.04 on Windows Subsystem for Linux (WSL). Full-scale training and experiments were carried out on the Wilkes3 clusters of the Cambridge Service for Data Driven Discovery (CSD3), which contains two AMD EPYC 7763 64-core CPUs @ 1.8GHz, 1000GB RAM, 1.04TB disk quota, four NVIDIA A100 SXM GPUs each with 80GB graphics memory in each node, but only one CPU core and one GPU were used for each experiment. Scientific Linux 7 is run on CSD3. CUDA 11.4 is installed on both my personal laptop and CSD3. All reports of training time in this chapter refer to the experiments on CSD3.

5.2 Hyperparameter Tuning

In order to validate the two proposed frameworks for GraphAC and find the optimal hyperparameter settings for them, I conducted hyperparameter tuning experiments in a grid-search manner on each framework. All experiments were trained on the `ogbg-molpcba` dataset for 50 epochs. After a few preliminary experiments, I decided to use a batch size of 512 for training, which works much better than smaller batch sizes such as 256 or 128. Adam was used as the optimiser for all experiments. On both sets of hyperparameter tuning experiments, a 10-layer PNA with 256 hidden dimensions, and a 10-layer PNA with 128 hidden dimensions, were used as the pair of competing GNNs. Both PNAs use [max, mean, sum] as their aggregators, [identity, amplification, attenuation] as their scalers, and their message passing functions are parametrised by 2-layer MLPs. The output dimensionality is set to 256 for all experiments.

5.2.1 Contestant-Judge Framework

Hyperparameter tuning of the contestant-judge framework was focused on the weighting coefficients of different loss terms. For simplicity, the Barlow Twins (\mathcal{L}_{BT}) and VICReg covariance regularisation (\mathcal{L}_{Cov}) terms are set to share the same weights, and the weight of the MLP prediction loss (\mathcal{L}_{MLP_A} and \mathcal{L}_{MLP_B}) term is set to 1, which means $\alpha = 1$ and $\beta = \gamma$ in Equation (4.2). Since the trial experiments have shown that the value of the MLP prediction loss term is greater than the Barlow Twins and VICReg covariance regularisation terms by an order of 10^3 , the candidate values of $\beta = \gamma$ are set to be 0 (i.e., do not include \mathcal{L}_{BT} and \mathcal{L}_{Cov}), 1, 10, 10^2 , 10^3 and 10^4 . In order to trade-off between collaboration (i.e., predicting the other GNNs' graph embeddings) and competition (i.e., preventing the other GNNs from predicting the GNNs' own graph embeddings), the λ in Equation (4.2) is set to take value from 0.1, 0.2, 0.5, 1 and 2. Both MLP judges are 2-layer MLPs with dimensionality of 256 in each layer, and all models are trained with a learning rate of 8×10^{-5} .

The results show that $\lambda = 0.5$ is the optimal value for trading-off between collaboration and competition, and increasing the weight for Barlow Twins and VICReg covariance regularisation terms can indeed help stabilising training. However, even with the largest weighting coefficient value (i.e., $\beta = \gamma = 10^4$), the contestant-judge framework still suffers from unstable training. This is most likely due to the non-convergence in simultaneous gradient descent on two pairs of models. This claim is supported by the learning curve of the two GNNs' loss difference shown in Figure 5.1. The plot of the explained variance in the principal component analysis (PCA) of the stronger GNN's output embeddings in Figure 5.1 also indicates that information collapse has occurred during training, since the entire output embeddings can be explained by less than 4 variables. Therefore, the contestant-judge framework is not satisfactory enough for the goal of this project, and requires more delicate design for its architecture and losses. This justifies the need for the competitive Barlow Twins framework.

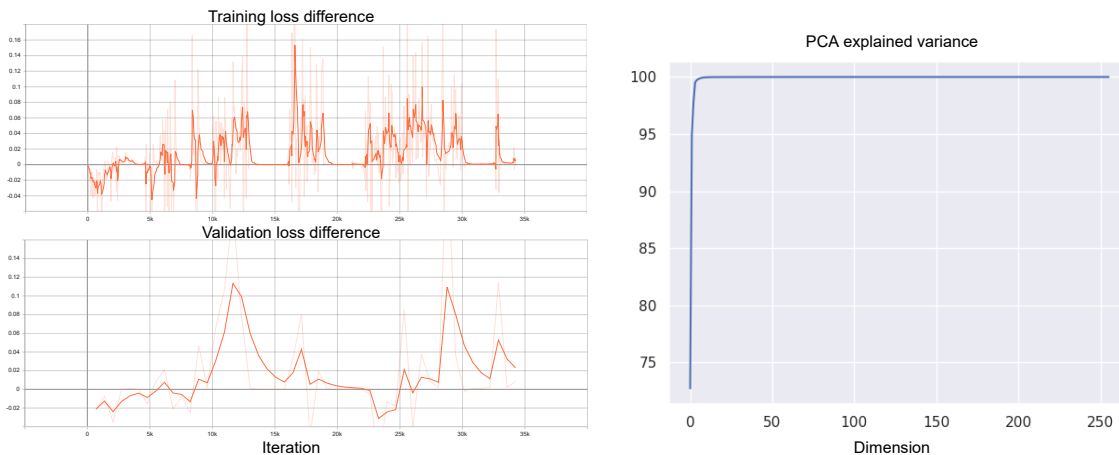


Figure 5.1: Learning curves of $\mathcal{L}_{GNN_A} - \mathcal{L}_{GNN_B}$ (left) and PCA explained variance of the stronger GNN's output embeddings (right) under the contestant-judge framework. The hyperparameter settings in this example is $\lambda = 0.5$ and $\beta = \gamma = 0$.

5.2.2 Competitive Barlow Twins

Hyperparameter tuning of the competitive Barlow Twins framework was focused on the weighting coefficients of the sums of the two triangles, and the learning rates. Again, for simplicity, the competitive Barlow Twins and VICReg covariance regularisation terms are set to share the same weights (i.e., $\alpha = \beta = 1$ in Equation (4.4)), and the value of λ in Equation (4.3) adopts the hyperparameter tuning results in the original Barlow Twins paper (Zbontar et al., 2021), which is 5×10^{-3} . The hyperparameter search space and the final values selected for the competitive Barlow Twins framework are specified in Table 5.1:

Table 5.1: Hyperparameters searched for the competitive Barlow Twins framework. **Bold** indicate the final selections.

Hyperparameter	Search space
Weighting coefficient of the triangle (μ)	[0.1, 0.2, 0.5, 1 , 2]
Learning rate	$[1 \times 10^{-5}, \mathbf{5 \times 10^{-5}}, 2 \times 10^{-4}]$

The results show that the competitive Barlow Twins framework can indeed distinguish the two GNNs with stable training, and can ensure that the more expressive GNN always has a lower loss. I also noted that hyperparameter tuning was much less critical for the competitive Barlow Twins framework compared to the contestant-judge framework, since the training was generally much more stable. An example learning curve of the loss difference of the two GNNs is shown in Figure 5.2. The PCA explained variance plot in Figure 5.2 also suggests that the competitive Barlow Twins framework can successfully avoid information collapse. Therefore, the competitive Barlow Twins framework satisfies the goal of this project, and can be used in GraphAC for the following evaluation experiments.

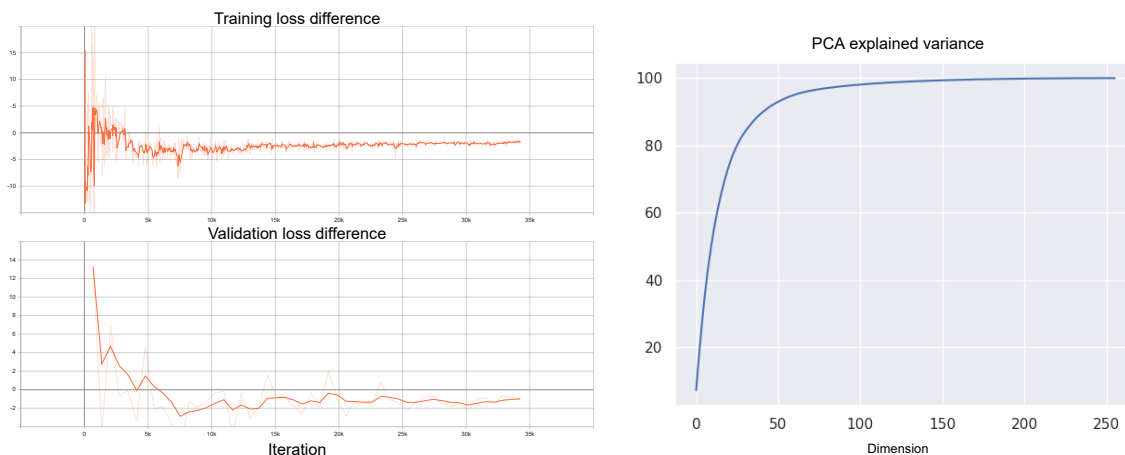


Figure 5.2: Learning curves of the loss differences (left) and PCA explained variance of the stronger GNN’s output embeddings (right) under the competitive Barlow Twins framework. The hyperparameter settings in this example is $\mu = 1$ and $lr = 5 \times 10^{-5}$.

5.3 Experiments

After the hyperparameter tuning experiments, the competitive Barlow Twins is chosen as the framework for GraphAC, and a series of experiments was conducted to examine GraphAC’s performance in distinguishing different GNNs. In order to fairly compare the GNNs as well as to evaluate GraphAC’s ability in distinguishing different components of a GNN, the experiments were split into five groups of controlled experiments. In each group, one component of the GNN is varied, while all other components of the GNN are fixed. The five aspects of a GNN evaluated by GraphAC are as follows:

- Number of GNN layers: in this group of experiments, PNAs with different numbers of layers are inserted into GraphAC for competitions. All other hyperparameters of the PNAs, including hidden dimensions and aggregators, are kept identical.
- Hidden dimensions: in this group of experiments, PNAs with different hidden dimensions are inserted into GraphAC for competitions. All other hyperparameters of the PNAs, including the number of layers and aggregators, are kept identical.
- Aggregators: in this group of experiments, PNAs with different aggregators are inserted into GraphAC for competitions. All other hyperparameters of the PNAs, including the number of layers and hidden dimensions, are kept identical.
- GNN architectures: in this group of experiments, PNA, GIN and GCN are inserted into GraphAC for competitions. The number of layers and hidden dimensions of the three types of GNNs are kept identical, in order to make their structures as similar as possible.
- Edge features: in this group of experiments, PNAs with the same structure, but one including the edge features and the other without the edge features, are inserted into GraphAC for competitions. This experiment is repeated with varying numbers of layers and hidden dimensions of the PNAs.

As an optional extension, I also selected some of the winning GNNs in the experiments, and fine-tuned them on the ZINC dataset, to test whether GraphAC can even make the GNNs produce representations that are useful for the downstream tasks, in addition to evaluating different GNNs. It should be noted that, since GraphAC is built for task-agnostic evaluations of the GNNs, the performance of GraphAC on fine-tuning is not a part of the core goal of this project.

5.4 Evaluation Metrics

Same as the hyperparameter tuning experiments, all experiments were trained on the ogbg-molpcba dataset, with a batch size of 512, for 50 epochs. For every experimental

setup listed in the previous section, I repeated the experiments three times with different random seeds, to obtain a more robust measure of the performance of GraphAC, and to ensure that the results are not due to random factors. The differences of the losses of the pairs of GNNs on the validation dataset, averaged over the last 10 epochs across the three random seeds, are used as the evaluation metrics of GraphAC.

GraphAC is deemed successful if *for most of the experiments, GraphAC can distinguish GNNs of different expressivity, by making the more expressive GNNs to have lower final loss values than their opponent GNNs, with statistically significant differences.*

For the optional extension fine-tuning, the GNNs pre-trained by GraphAC are compared against the baseline GNNs, which have the same structures of the pre-trained GNNs, but with all weights randomly initialised. The fine-tuning experiments were conducted on the ZINC dataset, with a batch size of 256, for 50 epochs. The final MAE is used as the evaluation metric. There is no requirement for the GNNs pre-trained by GraphAC to outperform the baseline GNNs whatsoever.

5.5 Results

5.5.1 Different Numbers of GNN Layers

In this group of experiments, PNAs with 2, 4, 6, 8, 10 layers compete in the GraphAC framework on a double round-robin basis, with one extra experiment performed for each model to compete against itself. All PNAs have a fixed hidden dimensionality of 256, and use [max, mean, sum] as their aggregators. The training took from 1.7 hours (2-layer PNA vs. 2-layer PNA) to 4.2 hours (10-layer PNA vs. 10-layer PNA). The results of the experiments, recorded as $\mathcal{L}_{\text{GNN}_A} - \mathcal{L}_{\text{GNN}_B}$, are reported in Table 5.2:

Table 5.2: Loss differences of PNAs (hidden dim = 256, aggregators = [max, mean, sum]) with different numbers of layers, recorded as $\mathcal{L}_{\text{GNN}_A} - \mathcal{L}_{\text{GNN}_B}$. Negative value means GNN_A wins the game, and positive value means GNN_B wins the game. Greater absolute value indicates larger gap in expressivity determined by GraphAC. The gradient from bottom left to upper right clearly indicates that more layers lead to more expressive representations.

		#Layers in GNN_B				
		2	4	6	8	10
#Layers in GNN_A	2	-0.094	1.402	1.749	1.883	1.977
	4	-1.350	0.078	0.722	0.939	1.345
	6	-1.638	-0.914	0.035	0.701	0.845
	8	-1.837	-1.411	-0.542	0.010	0.516
	10	-1.870	-1.532	-1.177	-0.434	0.063

The results clearly show that GraphAC can always successfully distinguish GNNs of different depths, and ensure that deeper GNNs, which are inherently more expressive, can win the games with lower losses. Furthermore, there is no distinction between GNN_A or GNN_B , meaning that for all pairs of experiments, even if the order is switched, the absolute loss differences still remain roughly equal but only with the sign flipped. These observations suggest that GraphAC can genuinely distinguish different GNNs, regardless of their ordering in the framework. In addition, for the experiments with the same GNNs competing, GraphAC produced loss differences close to 0, which means that it can also allow GNNs with the same expressivity to tie, instead of falsely deciding a winner. Another delightful observation is that, for every three GNNs GNN_A , GNN_B and GNN_C in the experiments, if their expressivity can be ordered as $GNN_A > GNN_B > GNN_C$, then there is also $|\mathcal{L}_{GNN_A} - \mathcal{L}_{GNN_C}| > |\mathcal{L}_{GNN_A} - \mathcal{L}_{GNN_B}|$ produced by GraphAC. This shows that GraphAC is highly likely to be able to produce a total ordering of all GNNs, which further enhances its credibility in evaluating GNNs.

5.5.2 Different Hidden Dimensions

In this group of experiments, 4-layer PNAs with hidden dimensionalities of 16, 32, 64, 128, 256 compete in the GraphAC framework on a double round-robin basis, also with one extra experiment performed for each model to compete against itself. All PNAs use [max, mean, sum] as their aggregators. The results of those experiments, recorded again as $\mathcal{L}_{GNN_A} - \mathcal{L}_{GNN_B}$, are reported in Table 5.3. The results agree with all observations in Section 5.5.1: more expressive GNNs always win the game, regardless of their orders; same GNNs always tie, with negligible loss differences; if a pair of GNNs has a wider gap in expressivity than another pair, then their loss difference is also greater in magnitude. This suggests that GraphAC can genuinely distinguish

Table 5.3: Loss differences of PNAs (num layers = 4, aggregators = [max, mean, sum]) with different hidden dimensions, recorded as $\mathcal{L}_{GNN_A} - \mathcal{L}_{GNN_B}$. Negative value means GNN_A wins the game, and positive value means GNN_B wins the game. Greater absolute value indicates larger gap in expressivity determined by GraphAC. The gradient from bottom left to upper right shows a strong correlation between the hidden dimension of a GNN and its ability to win the game.

		#Hidden dims in GNN_B				
		16	32	64	128	256
#Hidden dims in GNN_A	16	0.007	1.229	1.964	2.348	2.436
	32	-1.249	-0.016	0.985	1.545	2.102
	64	-2.034	-0.922	-0.019	1.156	1.870
	128	-2.400	-1.671	-1.036	0.092	1.655
	256	-2.560	-2.221	-1.931	-1.443	-0.037

GNNs of different expressivity with respect to the hidden dimensionality, and cross-validates GraphAC’s reliability.

5.5.3 Different Aggregators

In this group of experiments, four 4-layer PNAs with 64 hidden dimensions, and [max], [mean], [sum], [max, mean, sum] as their aggregators respectively, are set to compete in the GraphAC framework on a double round-robin basis, again with one extra experiment performed for each model to compete against itself. The final loss differences of those experiments are reported in Table 5.4. Again, the results agree with all observations in both Sections 5.5.1 and 5.5.2, showing that GraphAC is also capable of distinguishing GNNs with different expressivity caused by different aggregators, and confirms Xu et al. (2019)’s findings that $\text{sum} > \text{mean} > \text{max}$ in terms of expressivity, and Corso et al. (2020)’s findings that combining multiple aggregators can improve GNN’s expressivity, compared to using only a single aggregator. However, it should be noted that the loss differences are not as significant as in the previous experiments. This is possibly because the effect on expressivity by the aggregators is less significant than the number of parameters (i.e., number of layers and hidden dimensions).

Table 5.4: Loss differences of PNAs (num layers = 4, hidden dims = 64) with different aggregators: {[max], [mean], [sum], [max, mean, sum] (Combined)}, recorded as $\mathcal{L}_{\text{GNN}_A} - \mathcal{L}_{\text{GNN}_B}$. Negative value means GNN_A wins the game, and positive value means GNN_B wins the game. Greater absolute value indicates larger gap in expressivity determined by GraphAC. The gradient from bottom left to upper right clearly indicates that [max, mean, sum] > sum > mean > max in terms of expressivity.

		Aggregators in GNN_B			
		[max]	[mean]	[sum]	Combined
Aggregators in GNN_A	[max]	-0.025	0.127	0.322	0.399
	[mean]	-0.154	-0.011	0.228	0.385
	[sum]	-0.329	-0.277	-0.034	0.175
	Combined	-0.342	-0.307	-0.239	-0.019

5.5.4 Different GNN Architectures

In this group of architectures, PNA, GIN and GCN, all with 4 layers and 64 hidden dimensions, and PNA with [max, mean, sum] as aggregators, are set to compete in the GraphAC framework on a double round-robin basis, again with one extra experiment performed for each model to compete against itself. The final loss differences of those experiments are reported in Table 5.5. Once more, the results agree with all observations in the previous sections, and further shows that GraphAC is generalisable to other GNN types than PNA. It also aligns with the proofs that $\text{PNA} > \text{GIN} > \text{GCN}$ in terms of expressivity (Xu et al., 2019; Corso et al., 2020).

It is observed that the differences between architectures are larger than simply changing the aggregators. This can be due to other subtle differences, such as message passing framework in PNA compared to convolutions in GCN and GIN, or the added ϵ term in GIN compared to GCN.

Table 5.5: Loss differences of PNA, GIN and GCN (num layers = 4, hidden dims = 64, PNA aggregator = [max, mean, sum]), recorded as $\mathcal{L}_{\text{GNN}_A} - \mathcal{L}_{\text{GNN}_B}$. Negative value means GNN_A wins the game, and positive value means GNN_B wins the game. Greater absolute value indicates larger gap in expressivity determined by GraphAC. The gradient from bottom left to upper right clearly indicates that sum > mean > max in terms of expressivity.

		GNN _B architecture		
		GCN	GIN	PNA
GNN _A architecture	GCN	-0.091	0.483	0.716
	GIN	-0.475	-0.053	0.515
	PNA	-0.652	-0.465	-0.019

5.5.5 Inclusion of Edge Features

In this group of experiments, PNAs with 4, 6, and 8 layers, and hidden dimensionalities ranging from 64, 128, and 256 are used. All PNAs use [max, mean, sum] as their aggregators. In each experiment, the same PNA with one including the edge features of the graphs, and the other removing the edge features, are set to compete in the GraphAC framework. The loss differences of the experiments, recorded as $\mathcal{L}_{\text{GNN}(\text{edge features})} - \mathcal{L}_{\text{GNN}(\text{no edge features})}$, is reported in Table 5.6. The results show that GraphAC can correctly reward the GNNs that include edge features. It is observed that when the number of layers and hidden dimensions are larger, the magnitude of the loss difference $\mathcal{L}_{\text{GNN}(\text{edge features})} - \mathcal{L}_{\text{GNN}(\text{no edge features})}$ becomes smaller. This is possibly because when a GNN is more complex, it can capture enough information from the graphs even without the edge features, and the gain in performance by including edge features becomes relatively smaller.

Table 5.6: Loss differences of PNAs with and without edge features, recorded as $\mathcal{L}_{\text{GNN}(\text{edge features})} - \mathcal{L}_{\text{GNN}(\text{no edge features})}$. The PNAs always use the aggregators = [max, mean, sum], while the num layers = {4, 6, 8} and hidden dims = {64, 128, 256} are varied. Negative value means the PNA with edge features wins the game in all experiments, and greater absolute value indicates larger gap in expressivity determined by GraphAC.

		#hidden dimensions		
		64	128	256
#Layers	4	-0.714	-0.750	-0.671
	6	-0.925	-0.558	-0.501
	8	-0.792	-0.309	-0.422

5.5.6 Fine-tuning

For the optional extension of GraphAC, I selected the most complex GNN trained against various opponent GNNs in the previous experiments, namely PNA with 10 layers and 256 hidden dimensions, and fine-tuned it on the ZINC dataset. I used a randomly initialised 10-layer PNA with 256 hidden dimensions as the baseline for comparison. In the first scenario, I used the weights of the pre-trained PNAs as the initial weights of the fine-tuning task. Since catastrophic forgetting and feature distortion (Kumar et al., 2022) may occur during fine-tuning task, I tried the same experiment but using linear probing. In this second scenario, I froze the weights of both the pre-trained PNAs and the baseline PNA, and only used their output graph embeddings, appended with a 2-layer MLP with 256 dimensions in each layer as a prediction head, for the fine-tuning task. Although this is limited in terms of maximal training performance, it is known to perform better in out-of-distribution settings or in low-data settings by avoiding to overfitting and catastrophic forgetting (Kumar et al., 2022).

The fine-tuning results, recorded as the MAE loss on ZINC, are reported in Table 5.7. Unfortunately, all the pre-trained PNAs perform worse than the baseline model on the fine-tuning task. Here it needs to be re-emphasised that performance on the fine-tuning tasks is not a part of the core goals for GraphAC. However, I can still harness useful observations from the the failure pattern: as the fine-tuning results tend to worsen when the pre-training opponent becomes more complex, I can hypothesise that when a GNN competes against a more complex GNN in GraphAC, it is pushed to learn more complex but non-essential information about the graph, and thus decreases its ability to capture more basic but useful information. This also inspires me into extending GraphAC into a multi-GNN framework instead of having only two GNNs, so that the GNNs with different expressivity can capture different levels of information of the graphs, and create a better overall understanding. In some sense, weaker GNNs would regularize the stronger ones to learn simpler and more useful information.

Table 5.7: Fine-tuning results of GraphAC on the ZINC dataset, measured by MAE.

Model	Opponent	MAE on ZINC	
		Fine-tuning	Linear-probing
PNA (10 layers)	N/A (baseline)	0.093	0.555
PNA (10 layers)	PNA (2 layers)	0.110	0.787
PNA (10 layers)	PNA (4 layers)	0.124	0.850
PNA (10 layers)	PNA (6 layers)	0.149	0.814
PNA (10 layers)	PNA (8 layers)	0.162	0.927
PNA (10 layers)	PNA (10 layers)	0.154	0.911

Chapter 6

Summary and Conclusions

6.1 Accomplishments

In this project, I designed and built GraphAC, a conceptually novel, principled, and task-agnostic framework for evaluating GNNs through contrastive self-supervision, without the need of handcrafted augmentations. I designed two different architectures for GraphAC: a contestant-judge framework that consists of two GNN + MLP pairs, and a competitive Barlow Twins framework that incorporates a novel objective function inspired by the Barlow Twins loss (Zbontar et al., 2021), which replaces its redundancy reduction term with a difference between the upper-triangle and lower-triangle of the cross-correlation matrix of the two GNN’s output embeddings. The competitive Barlow Twins framework successfully distinguishes GNNs of different expressivity in all experiments, across various aspects including the number of layers, hidden dimensionality, aggregators, GNN architecture and edge features, and ensures that more expressive GNNs can always win with a statistically significant difference. GraphAC is also able to estimate the degree of expressiveness of different GNNs, and produce a total ordering of all GNNs with its measurements. GraphAC has significantly exceeded its preset success criteria, and brings notable contributions to the graph SSL community by providing a novel principle of evaluating GNNs and a stable contrastive SSL framework without requiring handcrafted augmentations. Therefore, I can claim with full confidence that this project has been successful.

6.2 Future Work

Future work on GraphAC includes the following directions:

- In order to test GraphAC’s generalisability in other types of graphs, run GraphAC on datasets outside the molecular domain, or even test its generalisability to other DNNs, such as CNNs in computer vision;

- extend GraphAC's ability to producing informative and transferable representations for downstream tasks, by including multiple GNNs to simultaneously compete in the framework, so that the GNNs with different expressivity can capture different levels of information of the graphs, and create a better overall graph representations;
- although the contestant-judge framework was unsatisfactory, it still has a potential to be leveraged, by introducing more delicate design and objective functions, or even converting it into an actor-critic framework and incorporate reinforcement learning to it.

Bibliography

- Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, volume 70, pages 214–223. PMLR.
- Bardes, A., Ponce, J., and LeCun, Y. (2022). VICReg: Variance-invariance-covariance regularization for self-supervised learning. In *10th International Conference on Learning Representations (ICLR 2022)*. OpenReview.net.
- Bodnar, C., Frasca, F., Otter, N., Wang, Y., Liò, P., Montufar, G. F., and Bronstein, M. (2021). Weisfeiler and Lehman go cellular: CW networks. In *Advances in Neural Information Processing Systems (NeurIPS 2021)*, volume 34, pages 2625–2640. Curran Associates, Inc.
- Bronstein, M. M. (2020). Expressive power of graph neural networks and the Weisfeiler-Lehman test. *Towards Data Science*, <https://towardsdatascience.com/expressive-power-of-graph-neural-networks-and-the-weisfeiler-lehman-test-b883db3c7c49>.
- Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. (2021). Geometric deep learning: grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*.
- Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, volume 119, pages 1597–1607. PMLR.
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1724–1734. Association for Computational Linguistics.
- Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. (2020). Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems (NeurIPS 2020)*, volume 33, pages 13260–13271. Curran Associates, Inc.

- Donsker, M. D. and Varadhan, S. R. S. (1983). Asymptotic evaluation of certain Markov process expectations for large time. IV. *Communications on Pure and Applied Mathematics*, 36(2):183–212.
- Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. (2020). Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982v3*.
- Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. (2022). Graph neural networks with learnable structural and positional representations. In *10th International Conference on Learning Representations (ICLR 2022)*. OpenReview.net.
- Fey, M. and Lenses, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*.
- Getoor, L. (2005). Link-based classification. In *Advanced methods for knowledge discovery from complex data*, pages 189–207. Springer.
- Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, volume 70, pages 1263–1272. PMLR.
- Goodfellow, I. J. (2015). On distinguishability criteria for estimating generative models. In *3rd International Conference on Learning Representations (ICLR 2015), Workshop Track Proceedings*.
- Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS 2014)*, volume 27, pages 2672–2680. Curran Associates, Inc.
- Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 297–304. PMLR.
- Hamilton, W. L. (2020). Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159. Morgan & Claypool Publishers.
- He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2020)*.
- Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., and Bengio, Y. (2019). Learning deep representations by mutual information estimation and maximization. In *7th International Conference on Learning Representations (ICLR 2019)*. OpenReview.net.

- Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257.
- Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.
- Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., and Leskovec, J. (2021). OGB-LSC: A large-scale challenge for machine learning on graphs. In *35th Conference on Neural Information Processing Systems (NeurIPS 2021) Datasets and Benchmarks Track*.
- Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020). Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems (NeurIPS 2020)*, volume 33, pages 22118–22133. Curran Associates, Inc.
- Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, volume 37, pages 448–456. PMLR.
- Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S., and Coleman, R. G. (2012). ZINC: A free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768.
- Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR 2015)*.
- Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. In *2nd International Conference on Learning Representations (ICLR 2014)*.
- Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations (ICLR 2017)*. OpenReview.net.
- Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. (2021). Rethinking graph transformers with spectral attention. In *Advances in Neural Information Processing Systems (NeurIPS 2021)*, volume 34, pages 21618–21629. Curran Associates, Inc.
- Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.
- Kumar, A., Raghunathan, A., Jones, R. M., Ma, T., and Liang, P. (2022). Fine-tuning can distort pretrained features and underperform out-of-distribution. In *10th International Conference on Learning Representations (ICLR 2022)*. OpenReview.net.

- McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. (2000). Automating the construction of Internet portals with machine learning. *Information Retrieval*, 3(2):127–163.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.
- Monti, F., Frasca, F., Eynard, D., Mannion, D., and Bronstein, M. M. (2019). Fake news detection on social media using geometric deep learning. In *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*.
- Nowozin, S., Cseke, B., and Tomioka, R. (2016). f-GAN: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems (NIPS 2016)*, volume 29, pages 271–279. Curran Associates, Inc.
- Oord, A. v. d., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS 2019)*, volume 32, pages 8026–8037. Curran Associates, Inc.
- Sato, R. (2020). A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*.
- Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. (2008). Collective classification in network data. *AI Magazine*, 29(3):93.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., et al. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.
- Stärk, H., Beaini, D., Corso, G., Tossou, P., Dallago, C., Günnemann, S., and Liò, P. (2021). 3D Infomax improves GNNs for molecular property prediction. *arXiv preprint arXiv:2110.04126*.
- Stokes, J. M., Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N. M., MacNair, C. R., French, S., Carfrae, L. A., Bloom-Ackermann, Z., Tran, V. M., Chiappino-Pepe, A., Badran, A. H., Andrews, I. W., Chory, E. J., Church, G. M., Brown, E. D., Jaakkola,

- T. S., Barzilay, R., and Collins, J. J. (2020). A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702.
- Sun, F., Hoffmann, J., Verma, V., and Tang, J. (2020). InfoGraph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *8th International Conference on Learning Representations (ICLR 2020)*. OpenReview.net.
- Veličković, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. (2019). Deep graph infomax. In *7th International Conference on Learning Representations (ICLR 2019)*. OpenReview.net.
- Vinyals, O., Bengio, S., and Kudlur, M. (2016). Order matters: Sequence to sequence for sets. In *4th International Conference on Learning Representations (ICLR 2016)*.
- Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., and Zhang, Z. (2019). Deep Graph Library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*.
- Weisfeiler, B. and Leman, A. (1968). A reduction of a graph to canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16. English translation available at https://www.itl.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.
- Wu, Z., Ramsundar, B., Feinberg, E., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. (2018). MoleculeNet: a benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530.
- Xie, Y., Xu, Z., Zhang, J., Wang, Z., and Ji, S. (2022). Self-supervised learning of graph neural networks: A unified review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *7th International Conference on Learning Representations (ICLR 2019)*. OpenReview.net.
- Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. (2018). Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, volume 80, pages 5453–5462. PMLR.
- Xu, M., Wang, H., Ni, B., Guo, H., and Tang, J. (2021). Self-supervised graph-level representation learning with local and global structure. In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, volume 139, pages 11548–11558. PMLR.

- Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983.
- You, Y., Chen, T., Shen, Y., and Wang, Z. (2021). Graph contrastive learning automated. In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, volume 139, pages 12121–12132. PMLR.
- You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., and Shen, Y. (2020). Graph contrastive learning with augmentations. In *Advances in Neural Information Processing Systems (NeurIPS 2020)*, volume 33, pages 5812–5823. Curran Associates, Inc.
- Zbontar, J., Jing, L., Misra, I., LeCun, Y., and Deny, S. (2021). Barlow twins: Self-supervised learning via redundancy reduction. In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, volume 139, pages 12310–12320. PMLR.

Appendix A

Notations

A.1 Numbers and Arrays

a	A scalar (integer or real)
\mathbf{a}	A vector
\mathbf{A}	A matrix
\mathbf{I}	An identity matrix (dimensionality implied by context)
A	A scalar random variable

A.2 Indexing

a_i	Element i of vector \mathbf{a} , with indexing starting at 1
A_{ij}	Element i, j of matrix \mathbf{A}
$\mathbf{a}_{[i:j]}$	Sub-vector of \mathbf{a} sliced from element i to element j (both included)
$\mathbf{A}_{[i_1:i_2],[j_1:j_2]}$	Sub-matrix of \mathbf{A} sliced between rows i_1 to i_2 and columns j_1 to j_2
$A^{(t)}$	Random variable A at time t
$\mathbf{h}^{(l)}$	The output of the l -th layer

A.3 Calculus and Linear Algebra Operations

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\mathbf{x}}y$	Gradient of y with respect to \mathbf{x}
\mathbf{A}^\top	Transpose of matrix \mathbf{A}
$\mathbf{A} \otimes \mathbf{B}$	Tensor product of \mathbf{A} and \mathbf{B}

A.4 Sets and Functions

\mathcal{A} or $\{\cdot\}$	A set
$\{\{\cdot\}\}$	A multiset
\mathbb{R}	The set of real numbers
\mathcal{G}	A graph
\mathfrak{G}	A set of graphs
$ \mathcal{A} $	The cardinality of set \mathcal{A}
$f : \mathcal{A} \rightarrow \mathcal{B}$	The function f with domain \mathcal{A} and image \mathcal{B}
f_{θ} or $f(\cdot; \theta)$	A function parametrised by θ
\mathbf{F}	A matrix-valued function
\mathcal{L}	The loss function
ϕ, ψ, \dots	Learnable functions
σ	A non-linear activation function
\oplus	A permutation-invariant operator

Sometimes a function f whose argument is a scalar can be applied to a vector or matrix: $f(\mathbf{x})$ or $f(\mathbf{X})$. This denotes the application of f to the vector/matrix element-wise. For example, if $\mathbf{Y} = f(\mathbf{X})$, then $Y_{ij} = f(X_{ij})$ for all valid values of i and j .

A.5 Probability

$p(A)$	A probability distribution over a continuous variable
$\Pr(A)$	A probability distribution over a discrete variable
$\Pr_X(x)$	The likelihood of random variable X given outcome x
$\mathbb{E}_X[f(X)]$	Expectation of $f(X)$ with respect to X

Appendix B

Algorithms

B.1 Contestant-Judge Framework

Algorithm 1 PyTorch-style pseudocode for the periodic training paradigm

```
# optimiser_steps: optimiser step counter
# iter_per_model: number of iterations for training each GNN

for x in dataloader: # load a batch with N samples
    # compute the loss functions
    ...

    # optimisation step for the GNNs
    if (optim_steps // iter_per_model) % 2 == 0:
        loss_a.backward()
        optimiser_gnn_a.step()
        optimiser_gnn_a.zero_grad()
    else:
        loss_b.backward()
        optimiser_gnn_b.step()
        optimiser_gnn_b.zero_grad()

    # optimisation steps for the MLPs
    loss_ab.backward()
    loss_ba.backward()
    optimiser_mlp_a.step()
    optimiser_mlp_b.step()
    optimiser_mlp_a.zero_grad()
    optimiser_mlp_b.zero_grad()

    # update optimiser step counter
    optimiser_steps += 1
```

Algorithm 2 PyTorch-style pseudocode for the contestant-judge framework

```
# gnn_a, gnn_b: GNN contestant networks
# mlp_a, mlp_b: MLP judge networks
# alpha, beta, gamma, lambda, mu: coefficients of the loss terms
# N: batch size
# d: dimensionality of the embeddings
#
# mse_loss: mean squared error loss function
# diagonal: on-diagonal elements of a matrix
# off_diagonal: off-diagonal elements of a matrix

for x in dataloader: # load a batch with N samples
    # compute embeddings and predictions
    h_a_out = gnn_a(x) # N x d
    h_b_out = gnn_b(x) # N x d
    h_b_pred = mlp_a(x) # N x d
    h_a_pred = mlp_b(x) # N x d

    # prediction losses
    loss_ab = mse_loss(h_b_pred, h_b_out)
    loss_ba = mse_loss(h_a_pred, h_a_out)
    pred_loss_a = loss_ab - lambda * loss_ba
    pred_loss_b = loss_ba - lambda * loss_ab

    # normalize embeddings along the batch dimension (VICReg)
    h_a_out_norm = (h_a_out - h_a_out.mean(dim=0)) # N x d
    h_b_out_norm = (h_b_out - h_b_out.mean(dim=0)) # N x d

    # covariance matrices
    cov_a = (h_a_out_norm.T @ h_a_out_norm) / (N - 1) # d x d
    cov_b = (h_b_out_norm.T @ h_b_out_norm) / (N - 1) # d x d

    # covariance regularisation loss
    cov_loss = off_diagonal(cov_a).pow(2).sum() / d \
        + off_diagonal(cov_b).pow(2).sum() / d

    # normalize embeddings along the batch dimension (Barlow Twins)
    h_a_out_norm = h_a_out_norm / h_a_out.std(dim=0) # N x d
    h_b_out_norm = h_b_out_norm / h_b_out.std(dim=0) # N x d

    # cross-correlation matrix
    corr = (h_a_out_norm.T @ h_b_out_norm) / N # d x d

    # Barlow Twins loss
    bt_loss = (diagonal(corr) - 1).pow(2).sum() \
        + mu * off_diagonal(corr).pow(2).sum()

    # overall loss functions
    loss_a = alpha * pred_loss_a + beta * bt_loss + gamma * cov_loss
    loss_b = alpha * pred_loss_b + beta * bt_loss + gamma * cov_loss

    # optimisation steps
    ...
```

B.2 Competitive Barlow Twins

Algorithm 3 PyTorch-style pseudocode for the competitive Barlow Twins framework

```
# gnn_a, gnn_b: GNN encoder networks
# alpha, beta, lambda, mu: coefficients of the loss terms
# N: batch size
# d: dimensionality of the embeddings
#
# diagonal: on-diagonal elements of a matrix
# triu: upper-triangle elements of a matrix
# tril: lower-triangle elements of a matrix

for x in dataloader: # load a batch with N samples
    # compute embeddings and predictions
    h_a_out = gnn_a(x) # N x d
    h_b_out = gnn_b(x) # N x d
    h_b_pred = mlp_a(x) # N x d
    h_a_pred = mlp_b(x) # N x d

    # normalize embeddings along the batch dimension (VICReg)
    h_a_out_norm = (h_a_out - h_a_out.mean(dim=0)) # N x d
    h_b_out_norm = (h_b_out - h_b_out.mean(dim=0)) # N x d

    # covariance matrices
    cov_a = (h_a_out_norm.T @ h_a_out_norm) / (N - 1) # d x d
    cov_b = (h_b_out_norm.T @ h_b_out_norm) / (N - 1) # d x d

    # covariance regularisation loss
    cov_loss = off_diagonal(cov_a).pow(2).sum() / d \
        + off_diagonal(cov_b).pow(2).sum() / d

    # normalize embeddings along the batch dimension (Barlow Twins)
    h_a_out_norm = h_a_out_norm / h_a_out.std(dim=0) # N x d
    h_b_out_norm = h_b_out_norm / h_b_out.std(dim=0) # N x d

    # cross-correlation matrix
    corr = (h_a_out_norm.T @ h_b_out_norm) / N # d x d

    # competitive Barlow Twins loss components
    on_diagonal = (diagonal(corr) - 1).pow(2).sum()
    upper_triangle = triu(corr, diagonal=1).pow(2).sum()
    lower_triangle = tril(corr, diagonal=-1).pow(2).sum()

    # competitive Barlow Twins losses
    loss_a = alpha * (on_diagonal + lambda * (upper_triangle - mu * lower_triangle)) \
        + beta * cov_loss
    loss_b = alpha * (on_diagonal + lambda * (lower_triangle - mu * upper_triangle)) \
        + beta * cov_loss

    # optimisation steps
    loss_a.backward()
    loss_b.backward()
    optimiser_a.step()
    optimiser_b.step()
    optimiser_a.zero_grad()
    optimiser_b.zero_grad()
```
