Self-Supervised Learning for Molecular Property Prediction

Laurent Dillard
Elix, Inc.
Tokyo, Japan.
laurent.dillard@elix-inc.com

Abstract

Predicting molecular properties remains a challenging task with numerous potential applications, notably in drug discovery. Recently, the development of deep learning, combined with rising amounts of data, has provided powerful tools to build predictive models. Since molecules can be encoded as graphs, Graph Neural Networks (GNNs) have emerged as a popular choice of architecture to tackle this task. Training GNNs to predict molecular properties however faces the challenge of collecting annotated data which is a costly and time consuming process. On the other hand, it is easy to access large databases of molecules without annotations. In this setting, self-supervised learning can efficiently leverage large amounts of non-annotated data to compensate for the lack of annotated ones. In this work, we introduce a self-supervised framework for GNNs tailored specifically for molecular property prediction. Our framework uses multiple pretext tasks focusing on different scales of molecules (atoms, fragments and entire molecules). We evaluate our method on a representative set of GNN architectures and datasets and also consider the impact of the choice of input features. Our results show that our framework can successfully improve performance compared to training from scratch, especially in low data regimes. The improvement varies depending on the dataset, model architecture and, importantly, on the choice of input feature representation.

1 Introduction

In recent years, deep learning has made considerable progress in fields like computer vision and natural language processing. Consequently, it gained popularity in a number of other domains, including drug discovery. Predicting molecular properties is critically important to efficiently screen large datasets of compounds and select promising candidates to consider for further validation. In light of their success in processing graph structured data, Graph Neural Networks (GNNs) have become a popular approach to handle molecules.

Like most deep learning applications, training GNNs requires large amounts of annotated data and is therefore faced with the challenge of data scarcity. Especially, for molecular properties, collecting annotations is a costly process as it involves running chemical experiments. On the flip side, many large datasets of molecules have been made publicly available [12, 5, 16] such that it is easy to access large quantities of molecules without annotations. In this context, self-supervised learning has been demonstrated to effectively improve predictive performance [18, 14, 11, 25, 31]. One drawback of recent methods however is that they rely on pretext tasks that focus mostly on the atom level only. This is in contrast with molecular properties that are defined on the molecule level. Additionally, although the choice of input features can vary drastically from one method to another [18, 11], the impact of this choice has not been examined in previous studies as far as we are aware. Based on those observations, we introduce in this work a new self-supervised learning framework for GNNs

Preprint. Under review.

tailored specifically for molecular property prediction and analyze how the choice of input features impacts its performance.

2 Related Work

2.1 Graph Neural Networks

Graph Neural Networks (GNNs) have emerged as a powerful architecture to process graph-structured data. GNNs can be simply described using the message passing framework [6]. Each node in a graph sends and receives messages from its neighbors. An update function is used to update each node vector based on the messages it received.

Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ denote an undirected graph with $\mathbf{X}$ the input feature matrix encoding the nodes vectors and $\mathbf{E}$ the input feature matrix encoding the edges vectors, $\mathcal{N}(v)$ denoting the neighborhood of node v and n denoting the layer index. The GNN message passing is formulated as follows:

$$h_v^0 = \mathbf{X}_v \tag{1}$$

$$m_v^{n+1} = \sum_{w \in \mathcal{N}(v)} M_n(h_v^n, h_w^n, E_{vw}) \tag{2}$$

$$h_v^{n+1} = U_n(h_v^n, m_v^{n+1}) \tag{3}$$

The choice of of message and update function at each layer M_n, U_n depends on the GNN architecture. After N layers, a permutation invariant readout function R can be applied to obtain a single feature vector for the whole graph.

$$g(\mathcal{G}) = R(\{h_v^N | v \in \mathcal{G}\}) \tag{4}$$

This graph representation can then be processed by a multi-layer perceptron (MLP) to generate the desired output. A wide variety of GNNs architectures have been developed [13, 6, 30, 29, 23], achieving state-of-the-art performance in many applications, including molecular property prediction.

2.2 Molecular Property Prediction

Traditional approaches to molecular property prediction involve the use of Morgan fingerprints (ECFP) [17] combined with traditional machine learning models like support vector machines [9], random forest [1] or multi-layer perceptrons [8]. Another line of work makes use of deep architectures like Recurrent Neural Networks (RNNs) [10] and Transformers [22] applied directly to SMILES strings [26]. The main drawback of SMILES representation is that it does not explicitly encode structural information about the molecule. SMILES strings also dot not imply a one to one relation with molecular structures: different SMILES strings can represent the same molecule while many SMILES string are invalid and do not have corresponding molecular structures. Additionally, a minor modification in the string can lead to a large change in said molecular structures. Representing molecules as graphs where nodes (atoms) are connected by edges (bonds) therefore appears as a more natural approach. In this setting, GNN architectures can be interpreted as way to generate learnable molecular fingerprints. These models have been successfully applied to molecular property prediction, achieving state-of-the-art performance on a wide range of datasets [30, 28, 20].

2.3 Self-supervised learning

Self-supervised learning falls in the realm of unsupervised learning methods. It aims at leveraging large quantities of unlabeled data to learn general feature representation using pretext tasks. Those pretext tasks are designed such that the labels can be easily generated from the data and should push the network to learn meaningful features. After self-supervision, the trained network can be fine tuned on downstream tasks. This transfer learning setting proves efficient especially in cases where the number of samples available for the downstream tasks is limited. Self-supervised learning has been widely successful in computer vision [3] and natural language processing [15]. More recently, it has been increasingly applied to graph structured data and molecular property prediction. Recent works on self supervised learning for GNNs include several different approaches. Some work use techniques borrowed from computer vision like contrastive learning [25, 31–33]. For those methods,

the main challenge lies in finding graph augmentations that preserve molecular properties. Other techniques focus on on graph structured data in general and are inspired mostly by graph theory [11, 21]. Finally some techniques more specifically target molecular graphs and make use of pretext tasks designed for molecules only [18].

3 Method

In this section, we detail our self-supervised learning framework and the pretext tasks associated with it.

3.1 Intuition

Our goal was to design a framework specifically tailored for GNNs and molecular property prediction.

Many molecular properties of interest, like ADME properties [2], largely depend on global molecular level characteristics. However, most of the published self-supervised learning methods focus on node level pre-training, therefore only consider atom level pretext tasks. This can be explained because GNNs mainly work on the node level: the core of GNNs is to produce a vector representation for each node using the message passing scheme. In the context of molecular property prediction however, it is necessary to also take into account graph level representations and design pretext tasks associated with them.

Pre-training on the node level is also important as the final graph representation is obtained from the nodes. Generating meaningful node level representation is a necessary condition to obtain useful graph representations.

For some molecular properties such as toxicity, the property can be related to the presence or not of certain specific functional groups and is best understood on the molecular fragment level: it is neither spread across the entire molecule nor is it related specifically to a single atom. Enforcing the GNN to learn useful representation for arbitrary sized molecular fragments should therefore also be beneficial in the context of molecular property prediction.

3.2 Pretext tasks

We introduce the three distinct pretext tasks used in our self-supervised learning framework which focus each on a different scale of molecules: atom, fragment and entire molecule. A representation of our framework can be found in Figure 1

3.2.1 Atom level

A useful pretext task should depend on labels that can be generated easily and push the network to learn useful feature representation about the atom context. In order to comply to those two principles, we defined the atom level pretext task as a classification problem to recognize which fragment each atoms belong to.

More precisely, we used the list of fragments introduced for Synthetic Accessibility Score [4] (SAScore) computation as our available classes. This list of fragments is based on a structural analysis of roughly 1 million representative molecules from the PubChem database [16]. Each fragment is associated with its frequency of appearance. Considering that the total number of resulting fragments is 605,864 and that most fragments are extremely rare (51% appear only once), we restricted ourselves to only the top 2000 most frequent fragments which covers fragments that appear at least 1000 times in the database. This leads to a classification problem with 2000 classes. For each atom, we consider whether it belongs to any of the 2000 fragments. If it does, we assign its label to be the largest fragment it belongs to. If it belongs to none of the fragment then no loss will be computed on this atom during training. Considering that we use the most frequent fragments, it is rare that an atom is not assigned any label. On our pre-training dataset, this only happened for 1.16% of all atoms. We also considered using more fragments by training on 4000 classes but empirically

found it had no impact on the framework. This is consistent with the fact that additional classes only represent very rare fragments from which little useful knowledge can be gained.

The loss to be minimized for this task for each batch can be written as follows:

$$\mathcal{L}_{atom} = \frac{1}{|A|} \sum_{h_v^N, y_v \in A} \mathcal{L}(y_v, f_a(h_v^N)) \tag{5}$$

Where A denotes the set of atoms in the batch (excluding those that were not assigned any fragment), h_v^N denotes the feature representation of node v obtained following equation (3), y_v denotes the fragment label associated to node v, $\mathcal{L}$ denotes the cross entropy loss and f_a denotes a MLP head used to obtain the atom classification output.

3.2.2 Fragment level

For the fragment level pretext task, the goal is to decompose each molecules into fragments and teach the network to determine which fragments originate from the same molecule.

To do so, the molecular graphs are first partitioned into subsets of nodes, each subset corresponding to a fragment, then the edges between distinct subsets are removed. To allow for more diversity, the fragment sizes are uniformly distributed between a minimum and maximum size.
The network is trained to predict if two fragments belong to the same molecule. For a given pair of fragments, the output is obtained by considering the dot product between the fragment feature vectors. The fragment feature vectors are obtained as the average of all node vectors belonging to the fragment:

$$x_{\mathcal{F}} = \frac{1}{|\mathcal{F}|} \sum_{v \in \mathcal{F}} h_v^N \tag{6}$$

Where h_v^N denotes the node feature vector obtained after the last graph convolution of the GNN.

The fragment features are then projected onto a lower dimensional space by a simple two layers MLP. This is done to facilitate the pairwise dot product computation and allow decoupling with the other pretext tasks. Additionally, using a projection head can ensure a richer feature representation [3]. This is because the pretext task can force the representation to be invariant to features that could still be useful for certain downstream tasks. By adding the projection head we ensure only the vectors obtained after the projection head will get rid of those extra features while the feature representation generated by the GNN can still preserve them.

Once all fragment features have been generated, the pairwise dot products are computed and the model is trained with a binary classification loss where positive pairs consist of fragments from the same molecules while negative pairs consist of fragments from different molecules. Considering that the batch size is usually relatively large (128,256,512...) the problem is very imbalanced with much more negative than positive pairs. To overcome that issue, we enforce equal proportions between positive and negative pairs at each batch by considering all positive pairs and randomly sampling an equal number of negative pairs from the much larger number of them. The loss is then computed only on those selected positive and negative pairs to prevent imbalance.

The loss to be minimized for this task for each batch can be written as follows:

$$\mathcal{L}_{fragment} = \frac{1}{|N| + |P|} \Big(\sum_{\substack{\mathcal{F}, \mathcal{F}' \in P \\ \overline{\mathcal{F}}, \overline{\mathcal{F}'} \in N}} \mathcal{L}(1, \sigma(f_p(x_{\mathcal{F}}) \cdot f_p(x_{\mathcal{F}'}))) + \mathcal{L}(0, \sigma(f_p(x_{\overline{\mathcal{F}}}) \cdot f_p(x_{\overline{\mathcal{F}'}}))) \tag{7}$$

Where P and N denote respectively the set of positive and selected negative pairs in the batch, σ denotes the sigmoïd activation function, f_p denotes the fragment projection head, $\cdot$ denotes the dot product operation and $\mathcal{L}$ denotes the binary cross entropy loss.

3.2.3 Molecule level

For the molecule level task, we want to push the network to learn feature representations that encode global properties of the graph. To achieve this, we use a multi label classification task where we use the same fragments introduced for the atom level task 3.2.1 and predict for each molecule which fragments it contains.

The loss to be minimized for this task for each batch can be written as follows:

$$\mathcal{L}_{molecule} = \frac{1}{|B|} \sum_{\mathcal{G}, y \in B} \mathcal{L}(y, f(g(\mathcal{G}))) \tag{8}$$

Where B denotes the batch, g denotes the GNN function detailed in equation (4), f denotes a MLP head to generate the multi-label classification output and $\mathcal{L}$ denotes the binary cross entropy loss function.

The three losses are then combined such that the final loss minimized at each batch is:

$$\mathcal{L}_{final} = \lambda_{atom} * \mathcal{L}_{atom} + \lambda_{fragment} * \mathcal{L}_{fragment} + \lambda_{molecule} * \mathcal{L}_{molecule} \tag{9}$$

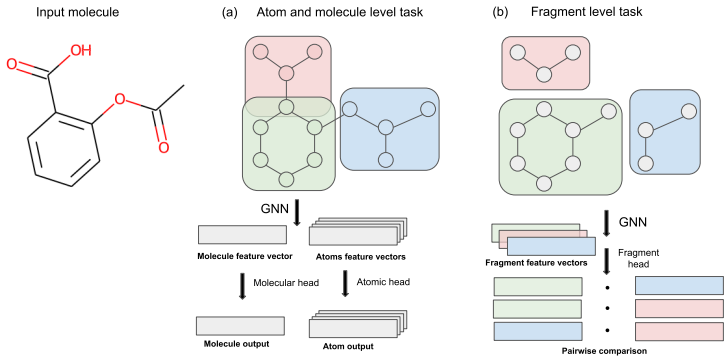

Figure 1: Illustration of our framework. **(a)** For the atom and molecule level tasks, the molecular graph is processed to return atom and molecule features each passed to parallel heads to provide classification outputs. The molecule task is a multi-label classification problem (one molecule can contain multiple fragments) while the atom task is multi-class one (one atom assigned to only one fragment). **(b)** For the fragment task, the molecular graph is decomposed into sub-graphs and processed by the GNN. A fragment head produces fragment features that are then compared to each other via dot product.

4 Experiments

We evaluate our framework on several standard molecular property benchmark datasets using different popular GNN architectures.

4.1 Datasets

The self-supervised learning framework implies having both an unlabeled dataset to perform pre-training and a target dataset, containing labels, to fine tune the pre-trained network and evaluate its performance.

4.1.1 Pre-training dataset

We used a subset [7] of ZINC database [12] containing 250,000 samples as the pre-training dataset. This subset covers a wide range of LogP and molecular weight values to ensure diversity.

4.1.2 Target datasets

For the target datasets, we used a subset of the MoleculeNet [27] benchmark datasets. Namely we used the following 6 classification datasets:

- **BACE**: This dataset contains 1,522 samples with binary labels on binding results with human β-secretase 1 (BACE-1).
- **BBBP**: This dataset contains 2,053 samples with binary labels about permeability of the compounds with the blood-brain barrier. Blood-brain barrier penetration is relevant for the design of drugs impacting central nervous system.
- **ClinTox**: This dataset contains 1,491 samples with two distinct binary labels associated. The first label refers to clinical trial toxicity and the second one to the FDA approval status.
- **SIDER**: This dataset contains 1,427 samples with binary labels for 27 different drug side-effects categories.
- **ToxCast**: This dataset contains 8,615 samples with 617 binary labels based on the results of in vitro toxicology experiments.
- **Tox21**: This dataset contains 8,014 samples with 12 binary labels based on toxicity measurements.

MoleculeNet contains more datasets but we chose to focus mainly on datasets with a lower number of samples since this better reflects practical applications of self-supervised learning for drug discovery.

4.2 Model architectures

To ensure that our framework generalizes to multiple GNN architectures, we selected 3 representative GNN architectures that are standardly used: Graph Convolution Network (GCN) [13], Graph Isomorphism Network (GIN) [29] and Directed Message Passing Neural Network (DMPNN) [30].

For GCN, we used 3 graph layers with a feature dimension of 512 and concatenation of max and sum as readout function. For GIN we used 5 graph layers with a feature dimension of 512 and concatenation of max and sum as readout function. For DMPNN we used a single graph layer with 4 steps with a feature dimension of 512 and set2set [24] as readout function. All architectures used a 2 layers MLP head on top of the obtained graph feature representations to produce the final output.

For the featurization of the graphs, we used 74 dimensional vectors for the atom features and 12 dimensional vectors for the bond features. The breakdown of these input features is detailed in Table 1.

Category	Property	Dimension
atom	One hot encoding of atom type	43
	One hot encoding of atom degree	11
	One hot encoding of atom valence	7
	Atom formal charge	1
	Number of radical electrons	1
	One hot encoding of atom hybridization	5
	Binary encoding is atom aromatic	1
	One hot encoding of total number of hydrogens	5
bond	One hot encoding of bond type	4
	Binary encoding is bond conjugated	1
	Binary encoding is bond in ring	1
	One hot encoding of bond stereochemistry	6

Table 1: Summary of all properties encoded in the atom and bond features used as input features for the graphs.

Dataset	**BACE (1522)**	**BBBP (2053)**	**ClinTox (1491)**	**SIDER (1427)**	**Toxcast (8615)**	**Tox21 (8014)**	**Average**	**Average Gain**
GCN	0.831(0.027)	0.898(0.033)	0.909(0.038)	0.605(0.027)	0.651(0.014)	0.769(0.013)	0.777(0.025)	
GCN (SSL)	0.858(0.023)	0.904(0.035)	0.927(0.026)	0.615(0.017)	0.653(0.012)	0.761(0.024)	0.786(0.023)	+0.009
GIN	0.844(0.028)	0.885(0.038)	0.902(0.045)	0.602(0.023)	0.625(0.013)	0.773(0.012)	0.772(0.027)	
GIN (SSL)	0.854(0.025)	0.891(0.032)	0.904(0.033)	0.614(0.017)	0.630(0.013)	0.773(0.018)	0.778(0.023)	+0.006
DMPNN	0.800(0.034)	0.894(0.038)	0.908(0.029)	0.620(0.021)	0.637(0.012)	0.762(0.018)	0.770(0.025)	
DMPNN (SSL)	0.855(0.027)	0.898(0.034)	0.910(0.040)	0.605(0.025)	0.618(0.013)	0.757(0.020)	0.774(0.026)	+ 0.004
GROVER [18]	**0.894(0.028)**	**0.940(0.019)**	**0.944(0.021)**	**0.658(0.023)**	**0.737(0.010)**	**0.831(0.025)**	**0.834(0.021)**	
GROVER-GIN [18]	0.862(0.020)	0.925(0.036)		0.648(0.015)				
Hu et al. [11]	0.851(0.027)	0.915(0.040)	0.762(0.058)	0.614(0.006)	0.714(0.019)	0.811(0.015)	0.778(0.028)	

Table 2: Comparison of standard training and our self-supervised framework along with two SOTA methods on MoleculeNet benchmark datasets. For GCN, GIN and DMPNN results are reported, both for standard training and with our self-supervised training framework denoted as (SSL). Shaded cells indicate best results between baseline (training from scratch) and SSL while **bold** results indicates best performance overall. For GROVER, GROVER-GIN and Hu et al. results were taken from [18].

4.3 Training and evaluation

During pre-training, we used the loss defined in equation (9) with λ_{atom}, $\lambda_{fragment}$, $\lambda_{molecule}$ all set to 1. Each model was trained for 40 epochs with a batch size of 512 using AdaBelief optimizer [34] with OneCycle learning rate policy [19] with a learning rate and weight decay of 1e-4 and 1e-5 respectively. On the target datasets, regardless of using self-supervised weights or not, each model was trained for 40 epochs with a batch size of 128 using AdaBelief and OneCycle policy. The learning rate and weight decay were set to 1e-3 and 1e-5 for all target datasets except BACE and BBBP which used a learning rate of 3e-4 and 5e-4 respectively.

Once the model were pre-trained, they were fine tuned and evaluated on each of the 6 MoleculeNet benchmark datasets. As done in other studies [18, 14], and standardly used in chemistry applications, we used scaffold splitting to generate the train, validation and test splits containing respectively 80%, 10% and 10% of the data. Scaffold splitting ensures that different scaffolds appear in the training validation and test split and is therefore more challenging than a random split. This is to better reflect the practical use case where predictive models are expected to generalize on new domains of the chemical space. Recent studies [18, 14] average the results across 3 runs using different random seeds. We still observed important variance based on the seeds therefore we increased the number of runs to 10 to improve the robustness of the results.

For each dataset and model architecture, the model was trained using binary cross entropy loss and saved after each epoch. The best checkpoint was selected using the validation set and finally evaluated on the test set. We compared two different settings: training from scratch from randomly initialized weights or fine tuning the weights obtained from self-supervision. In both settings we used the same training hyper parameters as we found little to no benefits in fine tuning them for each setting.

4.4 Results

To evaluate the models, we measured the ROC-AUC on the test set and report both the mean value and standard deviation measured across 10 runs. The results obtained for each dataset and model architecture are presented in Table 2. We observe that the impact of our self-supervised learning framework largely depends both on the dataset and model architecture. For the same dataset, improvement can vary from +0.01 to +0.055 ROC-AUC depending on the model used. Conversely, for the same model the impact of self-supervised learning can vary from an improvement of +0.055 to a decrease of -0.019 ROC-AUC depending on the dataset. Some datasets appear to consistently benefit from self-supervised learning across architectures as can be observed on BACE, BBBP and ClinTox while others vary on the model architecture like SIDER and Toxcast. Finally self-supervised learning either has no effect or decreases performance on Tox21 for all 3 architectures. Based on averaging the performance across all datasets, our self-supervised learning consistently improves performance across all architectures with the best average performance gain obtained for the GCN architecture. Interestingly, the datasets with a lower number of samples seem to

benefit more from self-supervised learning although sample size is not the only parameter of influence.

Comparing with other SOTA methods, GROVER [18] largely outperforms all other experiments. GROVER uses not only self-supervised learning but also introduces a new architecture, combining transformers and GNNs. In contrast, the performance significantly decreases when using GIN architecture. This shows that an important part of the performance gain in GROVER is not related to the self-supervised framework but to the new architecture used. However, even with GIN architecture, GROVER-GIN outperforms all other methods on the reported datasets. It should be noted that GROVER and GROVER-GIN performed self-supervised learning on a dataset of 11 millions molecules while we only used 250k for pre-training. Using our framework on a larger pre-training set might allow to further improve the performance but we leave these experiments for future work. Compared to Hu et al. our framework performs comparably while again using less data for the pre-training phase (250k vs 2 millions). Additionally our framework does not require pre-processing the entire pre-training dataset contrary to GROVER nor does it require an auxiliary GNN to be trained contrary to Hu et al..

4.5 Additional experiments

4.5.1 Ablation study

We conducted an ablation study to investigate the relative contribution of each of the 3 tasks of our self-supervised learning framework. To that end, we ran 3 different experiments where each model was trained on only one of the 3 tasks. Then we ran an additional experiment selecting the two tasks that performed best individually and compared all results to the complete framework that uses all 3 tasks. The results are presented in Table 3.

Here again the performance depends largely on the dataset with some task performing well individually on certain datasets (molecule only on BACE dataset) while leading to negative transfer learning on other datasets (molecule only on Toxcast and Tox21). On average, the atom level task reaches the best individual performance even though it is outperfomed by the other two on half the datasets (BACE, BBBP and Tox21). The combination of atom and fragment level tasks reaches a better average performance than any of the two individually and adding the third task brings and additional improvement in average performance. These results suggest that fine tuning the λ_{atom}, $\lambda_{fragment}$, $\lambda_{molecule}$ parameters depending on the dataset should be beneficial to reach optimal performance but using equal weighting gives a good baseline.

Dataset	**BACE (1522)**	**BBBP (2053)**	**ClinTox (1491)**	**SIDER (1427)**	**Toxcast (8615)**	**Tox21 (8014)**	**Average**	**Average Gain**
GCN	0.831(0.027)	0.898(0.033)	0.909(0.038)	0.605(0.027)	0.651(0.014)	**0.769(0.013)**	0.777(0.025)	
GCN (SSL)	0.858(0.023)	0.904(0.035)	**0.927(0.026)**	0.615(0.017)	**0.653(0.012)**	0.761(0.024)	**0.786(0.023)**	+ 0.009
GCN SSL atom only	0.848(0.019)	0.905(0.033)	0.894(0.042)	0.614(0.024)	0.651(0.008)	0.766(0.016)	0.780(0.024)	+ 0.003
GCN SSL fragment only	0.837(0.028)	**0.908(0.039)**	0.900(0.040)	0.611(0.028)	0.651(0.010)	**0.769(0.014)**	0.779(0.027)	+ 0.002
GCN SSL molecule only	**0.861(0.028)**	0.898(0.037)	0.905(0.041)	0.610(0.024)	0.635(0.008)	0.752(0.023)	0.777(0.027)	+ 0.000
GCN SSL atom + fragment	0.850(0.024)	0.904(0.037)	0.899(0.040)	**0.621(0.022)**	0.652(0.010)	**0.769(0.020)**	0.783(0.025)	+ 0.006

Table 3: Comparing the impact of each task of the self-supervised learning framework. 4 distinct settings are considered: using only the atom level task, using only the fragment level task, using only the molecule level task and using the combination of atom and fragment level tasks. Best performance overall is indicated in **bold**.

4.5.2 Testing different input features

We noticed that Hu et al. report an impressive average absolute improvement of 0.072 compared to their baseline while GROVER and GROVER-GIN only report reaching 0.038 and 0.027 respectively and our work only reached 0.009.
We conjectured that the choice of input features for the graphs is an important factor influencing how much a model can benefit from self-supervised learning. This came from the observation that the choice of input features was one of the main differences between the two previously mentioned works. Our hypothesis was that when using a poor feature representation as input, using self-supervised learning can bring more performance gain as it can effectively recover some of the missing input

features.
To test this hypothesis, we ran an additional set of experiments where we used a reduced set of input features, the same as those used by Hu et al.:

- The atom features are 2 dimensional vectors encoding the atom type and chirality.
- The bond features are 2 dimensional vectors encoding the bond type and direction.

We compared again the baseline setting where the model is trained from randomly initialized weights with using our self supervised learning framework. The results are presented in Table 4.

We observe that when using the reduced set of features, the increase in average performance brought by using our self-supervised learning framework increased drastically from +0.009, +0.006 and +0.004 to +0.052, +0.024 and + 0.055 for GCN, GIN and DMPNN respectively. These results strongly suggest that the choice of input features has a critical importance in the performance gain that can be achieved using self supervised learning. Using a more extensive set of input features may lead to a lower improvement in performance. We interpret this phenomena the following way: if a lot of the information that the network can learn from the self-supervision is already contained in the input features then the performance gain is marginal. On the contrary, if a more limited set of input features is used, the network has more opportunities to learn features that are not already included in the input features during self-supervision, which in turn leads to larger gains in performance.

Dataset	**BACE (1522)**	**BBBP (2053)**	**ClinTox (1491)**	**SIDER (1427)**	**Toxcast (8615)**	**Tox21 (8014)**	**Average**	**Average Gain**
GCN †	0.717(0.048)	0.864(0.038)	0.630(0.121)	0.572(0.023)	0.624(0.009)	0.715(0.048)	0.687(0.042)	
GCN (SSL) †	**0.856(0.019)**	**0.896(0.037)**	**0.687(0.051)**	**0.592(0.019)**	**0.649(0.004)**	**0.755(0.017)**	**0.739(0.025)**	+ 0.052
GIN †	0.816(0.038)	0.893(0.032)	0.560(0.070)	0.576(0.023)	0.598(0.019)	0.740(0.022)	0.697(0.033)	
GIN (SSL) †	**0.849(0.032)**	**0.904(0.026)**	**0.605(0.148)**	**0.594(0.029)**	**0.623(0.011)**	**0.750(0.015)**	**0.721(0.043)**	+ 0.024
DMPNN †	0.676(0.037)	0.833(0.051)	0.509(0.108)	0.564(0.024)	**0.603(0.018)**	0.710(0.039)	0.649(0.046)	
DMPNN (SSL) †	**0.831(0.032)**	**0.877(0.041)**	**0.595(0.109)**	**0.594(0.014)**	0.598(0.033)	**0.733(0.016)**	**0.704(0.041)**	+ 0.055

Table 4: Performance of models when using a reduced set of input features. †indicates that the experiments used a reduced set of features. Results in **bold** indicate for each architecture which of the baseline or SSL experiment performed better.

5 Conclusion

In this work, we applied self-supervised learning methods for GNNs specifically in the context of molecular property prediction. We introduced a framework that uses a combination of 3 tasks focusing each on a different scale of molecules: atom, fragment and entire molecules. We evaluated our framework on 6 datasets from MoleculeNet using scaffold splits and 3 different GNN architectures. For each architecture, we compared our framework to a baseline of training the network from randomly initialized weights. Finally we analyzed the impact of the choice of input features on our framework.

Our results indicate that self-supervision can successfully improve the performance of GNNs for molecular property prediction, especially in low data regime. However, our framework was not able to improve the performance consistently across datasets and architectures. Another important finding highlighted in this work is the importance of the choice of input features for self-supervision. When using a very limited set of input features, the gain in performance obtained by applying our self-supervised framework increased significantly and was consistent across all datasets and GNN architectures tested. This finding is consistent with the results of previous studies, where GROVER [18], that used a rich set of input features, reported a lower improvement on their baseline than Hu et al. [11], that used a much more reduced set.

References

[1] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324. URL `http://dx.doi.org/10.1023/A%3A1010933404324`.

[2] Darko Butina, Matthew D Segall, and Katrina Frankcombe. Predicting adme properties in silico: methods and models. *Drug Discovery Today*, 7(11):S83–S88, 2002. ISSN 1359-6446. doi: https://doi.org/10.1016/S1359-6446(02)02288-2. URL https://www.sciencedirect.com/science/article/pii/S1359644602022882.

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020. URL https://arxiv.org/abs/2002.05709.

[4] Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *J. Cheminformatics*, 1:8, 2009. URL http://dblp.uni-trier.de/db/journals/jcheminf/jcheminf1.html#ErtlS09.

[5] Anna Gaulton, Anne Hersey, Michał Nowotka, A. Patrícia Bento, Jon Chambers, David Mendez, Prudence Mutowo, Francis Atkinson, Louisa J. Bellis, Elena Cibrián-Uhalte, Mark Davies, Nathan Dedman, Anneli Karlsson, María Paula Magariños, John P. Overington, George Papadatos, Ines Smit, and Andrew R. Leach. The ChEMBL database in 2017. *Nucleic Acids Research*, 45(D1):D945–D954, 11 2016. ISSN 0305-1048. doi: 10.1093/nar/gkw1074. URL https://doi.org/10.1093/nar/gkw1074.

[6] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017. URL http://arxiv.org/abs/1704.01212.

[7] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, Feb 2018. ISSN 2374-7943. doi: 10.1021/acscentsci.7b00572. URL https://doi.org/10.1021/acscentsci.7b00572.

[8] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[9] M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998. doi: 10.1109/5254.708428.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9 (8):1735–1780, 1997.

[11] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. Pre-training graph neural networks. *CoRR*, abs/1905.12265, 2019. URL http://arxiv.org/abs/1905.12265.

[12] Shoichet Brian K Irwin John J. Zinc–a free database of commercially available compounds for virtual screening. *Journal of chemical information and modeling*, 2005. doi: 10.1021/ci049714. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1360656/.

[13] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL http://arxiv.org/abs/1609.02907.

[14] Pengyong Li, Jun Wang, Yixuan Qiao, Hao Chen, Yihuan Yu, Xiaojun Yao, Peng Gao, Guotong Xie, and Sen Song. Learn molecular representations from large-scale unlabeled molecules for drug discovery. *CoRR*, abs/2012.11175, 2020. URL https://arxiv.org/abs/2012.11175.

[15] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[16] PubChem. PubChem. URL https://pubchem.ncbi.nlm.nih.gov/.

[17] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, May 2010. ISSN 1549-9596. doi: 10.1021/ci100050t. URL https://doi.org/10.1021/ci100050t.

[18] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. 2020.

[19] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120, 2017. URL http://arxiv.org/abs/1708.07120.

[20] Ying Song, Shuangjia Zheng, Zhangming Niu, Zhang-hua Fu, Yutong Lu, and Yuedong Yang. Communicative representation learning on attributed molecular graphs. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2831–2838. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/392. URL https://doi.org/10.24963/ijcai.2020/392. Main track.

[21] Fan-Yun Sun, Jordan Hoffmann, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *CoRR*, abs/1908.01000, 2019. URL http://arxiv.org/abs/1908.01000.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

[23] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. 2018.

[24] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets, 2016.

[25] Yuyang Wang, Jianren Wang, Zhonglin Cao, and Amir Barati Farimani. Molclr: Molecular contrastive learning of representations via graph neural networks. *CoRR*, abs/2102.10056, 2021. URL https://arxiv.org/abs/2102.10056.

[26] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.*, 28(1):31–36, February 1988. ISSN 0095-2338. doi: 10.1021/ci00057a005. URL https://doi.org/10.1021/ci00057a005.

[27] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay S. Pande. Moleculenet: A benchmark for molecular machine learning. *CoRR*, abs/1703.00564, 2017. URL http://arxiv.org/abs/1703.00564.

[28] Zhaoping Xiong, Dingyan Wang, Xiaohong Liu, Feisheng Zhong, Xiaozhe Wan, Xutong Li, Zhaojun Li, Xiaomin Luo, Kaixian Chen, Hualiang Jiang, and Mingyue Zheng. Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism. *Journal of Medicinal Chemistry*, 63(16):8749–8760, Aug 2020. ISSN 0022-2623. doi: 10.1021/acs.jmedchem.9b00959. URL https://doi.org/10.1021/acs.jmedchem.9b00959.

[29] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018. URL http://arxiv.org/abs/1810.00826.

[30] Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, Andrew Palmer, Volker Settels, Tommi Jaakkola, Klavs Jensen, and Regina Barzilay. Analyzing learned molecular representations for property prediction. *Journal of Chemical Information and Modeling*, 59(8):3370–3388, Aug 2019. ISSN 1549-9596. doi: 10.1021/acs.jcim.9b00237. URL https://doi.org/10.1021/acs.jcim.9b00237.

[31] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *CoRR*, abs/2010.13902, 2020. URL https://arxiv.org/abs/2010.13902.

[32] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. *CoRR*, abs/2106.07594, 2021. URL https://arxiv.org/abs/2106.07594.

[33] Shichang Zhang, Ziniu Hu, Arjun Subramonian, and Yizhou Sun. Motif-driven contrastive learning of graph representations. *CoRR*, abs/2012.12533, 2020. URL https://arxiv.org/abs/2012.12533.

[34] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha C. Dvornek, Xenophon Papademetris, and James S. Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *CoRR*, abs/2010.07468, 2020. URL https://arxiv.org/abs/2010.07468.