# DropGNN: Random Dropouts Increase the Expressiveness of Graph Neural Networks

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

This paper studies Dropout Graph Neural Networks (DropGNNs), a new approach that aims to overcome the limitations of standard GNN frameworks. In DropGNNs, we execute multiple runs of a GNN on the input graph, with some of the nodes randomly and independently dropped in each of these runs. Then, we combine the results of these runs to obtain the final result. We prove that DropGNNs can distinguish various graph neighborhoods that cannot be separated by message passing GNNs. We derive theoretical bounds for the number of runs required to ensure a reliable distribution of dropouts, and we prove several properties regarding the expressive capabilities and limits of DropGNNs. We experimentally validate our theoretical findings on expressiveness. Furthermore, we show that DropGNNs perform competitively on established GNN benchmarks.

## 1 Introduction

Neural networks have been successful in handling various forms of data. Since some of the world's most interesting data is represented by graphs, Graph Neural Networks (GNNs) have achieved state-of-the-art performance in various fields such as quantum chemistry, physics or social networks [9; 19; 12]. On the other hand, GNNs are also known to have severe limitations and are sometimes unable to recognize even simple graph structures.

In this paper, we present a new approach to increase the expressiveness of GNNs, called Dropout Graph Neural Networks (DropGNNs). Our main idea is to execute not one but *multiple* different runs of the GNN. We then aggregate the results from these different runs into a final result.

In each of these runs, we remove ("drop out") each node in the graph with a small probability $p$. As such, the different runs of an episode will allow us to not only observe the actual extended neighborhood of a node for some number of layers $d$, but rather to observe various slightly perturbed versions of this $d$-hop neighborhood. We emphasize that this notion of dropouts is very different from the popular dropout regularization method; in particular, DropGNNs remove nodes during *both training and testing*, since their goal is to observe a similar distribution of dropout patterns during training and testing.

This dropout technique increases the expressive power of our GNNs dramatically: even when two distinct $d$-hop neighborhoods cannot be distinguished by a standard GNN, their dropout variants (with a few nodes removed) are already separable by GNNs in most cases. Thus by learning to identify the dropout patterns where the two $d$-hop neighborhoods differ, DropGNNs can also distinguish a wide variety of cases that are beyond the theoretical limits of standard GNNs.

**Our contributions.** We begin by showing several example graphs that are not distinguishable in the regular GNN setting but can be easily separated by DropGNNs. We then analyze the theoretical

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

properties of DropGNNs in detail. We first show that executing $\widetilde{O}(\gamma)$ different runs is often already sufficient to ensure that we observe a reasonable distribution of dropouts in a neighborhood of size $\gamma$. We then discuss the theoretical capabilities and limitations of DropGNNs in general, as well as the limits of the dropout approach when combined with specific aggregation methods.

We validate our theoretical findings on established problems that are impossible to solve for standard GNNs. We find that DropGNNs clearly outperform the competition on these datasets. We further show that DropGNNs have a competitive performance on several established graph benchmarks.

## 2 Related Work

GNNs apply deep learning to graph-structured data [22]. In GNNs, every node has an embedding that is shared over multiple iterations with its neighbors. This way nodes can gather their neighbors' features. In recent years, many different models have been proposed to realize how the information between nodes is shared [25]. Some approaches take inspiration from convolution [17; 6; 10], others from graph spectra [12; 4], others from attention [24], and others extend previous ideas of established concepts such as skip connections [26].

Principally, GNNs are limited in their expressiveness by the *Weisfeiler-Lehman test* (WL-test) [27], a heuristic to the graph isomorphism problem. [27] propose a new architecture, *Graph Isomoprhism Networks* (GIN), that is proven to be exactly as powerful as the WL test. However, even GINs cannot distinguish certain different graphs, namely those that the WL-test cannot distinguish. This finding [8] motivated more expressive GNN architectures. These improvements follow two main paths.

The first approach augments the features of nodes or edges by additional information to make nodes with similar neighborhoods distinguishable. Several kinds of information have been used: inspired from distributed computing are port numbers on edges [20], unique IDs for nodes [14], or random features on nodes [21; 1], another idea are angles between edges [13] from chemistry (where edges are electron bonds. All of these approaches have shortcomings: for ports and angles we know of graphs that these extensions cannot distinguish [8] and adding IDs or random features helps during training, but the learned models do not generalize. These additional features tend to overfit the training set, while DropGNNs also generalize to test set graphs.

The second approach exploits the fact that running the WL-test on tuples, triples, or generally $k-$tuples keeps increasing its expressiveness. Thus a GNN operating on tuples of nodes has higher expressiveness than a standard GNN [16; 15]. However, the downside of this approach is that even building a second-order graph blows up the graph quadratically. The computational cost quickly becomes a problem that needs to be to addressed, for example with sampling [16]. Furthermore, second-order graph creation is a global operation of the graph that destroys the local semantics induced by the edges. In contrast to this, DropGNN can reason about graphs beyond the WL-test with only a linear overhead (through run repetition), while also keeping the local graph structure intact.

Finally, we note that removing nodes is a common tool for regularization in deep neural networks, which has also seen use in GNNs [18; 7]. However, as mentioned before, this is a different dropout concept where nodes are only removed during training to reduce the co-dependence of nodes.

## 3 DropGNN

### 3.1 About GNNs

Almost all GNN architectures [24; 12; 27; 6; 25; 10; 26] follow the message passing framework [9; 3]. Every node starts with an embedding given by its initial features. One round of message passing has three steps. In the first MESSAGE step, nodes create a message based on their embedding and send this message to all neighbors. Second, nodes AGGREGATE all messages they receive. Third, every node UPDATEs its embedding based on its old embedding and the aggregated messages. One such round corresponds to one GNN layer. Usually, a GNN performs some number $d$ rounds of message passing. Thus, the node's embedding in a GNN reflects its features and the information of its $d-$hop neighborhood. Finally, a READOUT method translates these final embeddings into predictions. Usually, MESSAGE, AGGREGATE, UPDATE and READOUT are functions with learnable parameters, for instance linear layers with activation functions.

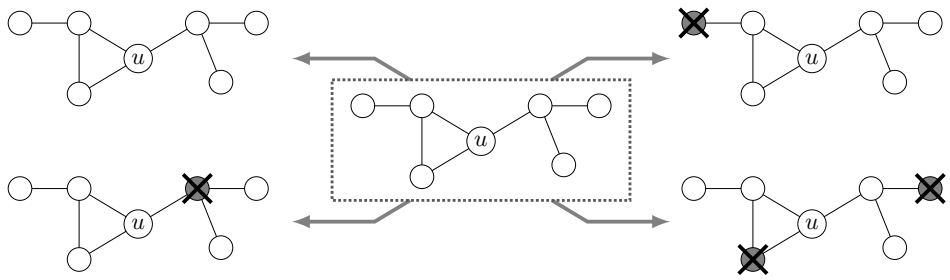



Figure 1: Illustration of $4$ possible dropout combinations from an example 2-hop neighborhood around $u$: a 0-dropout, two different 1-dropouts and a 2-dropout.

This GNN paradigm is closely related to the WL-test for a pair of graphs, which is an iterative color refinement procedure. In rounds $1, ..., d$, each node looks at its own color and the multiset of colors of its direct neighbors and uses a hash function to select a new color based on this information. As such, if the WL-test cannot distinguish two graphs, then a standard GNN cannot distinguish them either: intuitively, the nodes in these graphs receive the same messages and create the same embedding in each round, and thus they always arrive at the same final result.

### 3.2 Idea and motivation

The main idea of DropGNNs is to execute multiple independent runs of the GNN during both training and testing. In each run, every node of the GNN is removed with probability $p$, independently from all other nodes. If a node $v$ is removed during a run, then $v$ does not send or receive any messages to/from its neighbors and does not affect the remaining nodes in any way. Essentially, the GNN behaves as if $v$ (and its incident edges) were not present in the graph in the specific run, and no embedding is computed for $v$ in this run (see Figure 1 for an illustration).

Over the course of multiple runs, dropouts allow us to not only observe the $d$-hop neighborhood around any node $u$, but also several slightly perturbed variants of this $d$-hop neighborhood. In the different runs, the embedding computed for $u$ might also slightly vary, depending on which node(s) are missing from the $d$-hop neighborhood in a specific run. This increases the expressive power of GNNs significantly: even when two different $d$-hop neighborhoods cannot be distinguished by standard GNNs, the neighborhood variants observed when removing some of the nodes are usually still remarkably different. In Section 3.4, we discuss multiple examples for this improved expressiveness.

Our randomized approach means that in different runs, we will have different nodes dropping out of the GNN. As such, the GNN is only guaranteed to produce the same node embeddings in two runs if we have exactly the same subset of nodes dropping out. Given the $d$-hop neighborhood of a node $u$, we will refer to a specific subset of nodes dropping out as a *dropout combination*, or more concretely as a $k$-dropout in case the subset has size $k$.

In order to analyze the $d$-hop neighborhood of $u$, the reasonable strategy is to use a relatively small dropout probability $p$: this ensures that in each run, only a few nodes are removed (or none at all), and thus the GNN will operate on a $d$-hop neighborhood that is similar to the original neighborhood of $u$. As a result, 1-dropouts will be frequent, while for a larger $k$, observing a $k$-dropout will be unlikely.

To reduce the effect of randomization on the final outcome, we have to execute multiple independent runs of our GNN; we denote this number of runs by $r$. For a successful application of the dropout idea, we have to select $r$ large enough to ensure that the set of observed dropout combinations is already reasonably close to the actual probability distribution of dropouts. In practice, this will not be feasible for $k$-dropouts with large $k$ that occur very rarely, but we can already ensure for a reasonably small $r$ that e.g. the frequency of each 1-dropout is relatively close to its expected value.

### 3.3 Run aggregation

Recall that standard GNNs first compute a final embedding for each node through $d$ layers, and then they use a READOUT method to transform this into a prediction. In DropGNNs, we also need to introduce an extra phase between these two steps, called *run aggregation*.

In particular, we execute $r$ independent runs of the $d$-layer GNN (with different dropouts), which altogether produces $r$ distinct final embeddings for a node $u$. Hence we also need an extra step to merge these $r$ distinct embeddings into a single final embedding of $u$, which then acts as the input for the READOUT function. This run aggregation method has to transform a multiset of embeddings into a single embedding; furthermore, it has to be a *permutation-invariant* function (similarly to neighborhood aggregation), since the ordering of different runs carries no meaning.

We note that simply applying a popular permutation-invariant function for run aggregation, like `sum` or `max`, is often not sufficient. For example, in a simple case of $d = 1$, `sum` aggregation essentially provides the same `sum` as a standard GNN, only scaled by a factor $p$. A `max` aggregation, on the other hand, ignores most of the runs, and thus it is rather wasteful with resources.

Instead, one natural solution to extract more information from the distribution of runs is to first apply a simple transformation on each node feature, and only execute `sum` aggregation afterward. The most straightforward choice is to consider a transformation $x \to \sigma(Wx + b)$, with $\sigma$ denoting a basic non-linearity such as a sigmoid or step function. Such a simple transformation is already sufficient for almost all of our examples and theoretical results in the paper.

### 3.4 Motivational examples

We discuss several examples to demonstrate how DropGNNs are more expressive than standard GNNs. We only outline the intuitive ideas behind the behavior of the DropGNNs here; however, in Appendix A, we also describe the concrete functions that can separate each pair of graphs.

**Example 1.** Figure 2a shows a fundamental example of two different graphs that cannot be distinguished by the 1-WL test, consisting of cycles of different length. This example is known to be hard for extended GNNs variants: the two cases cannot even be distinguished if we also use port numbers or angles between the edges [8].

The simplest solution here is to consider a GNN with $d = 2$ layers; this already provides a very different distribution of dropouts in the two graphs. For example, the 8-cycle has 2 distinct 1-dropouts where $u$ retains both of its direct neighbors, but it only has 1 neighbor at distance 2; such a situation is not possible in the 4-cycle at all. Alternatively, the 4-cycle has a 1-dropout case with probability $p \cdot (1-p)^2$ where $u$ has 2 direct neighbors, but no distance 2 neighbors at all; this only happens for a 2-dropout in the 8-cycle, i.e. with a probability of only $p^2 \cdot (1-p)^2$. With appropriate weights, a GNN can learn to recognize these situations, and thus distinguish the two cases.

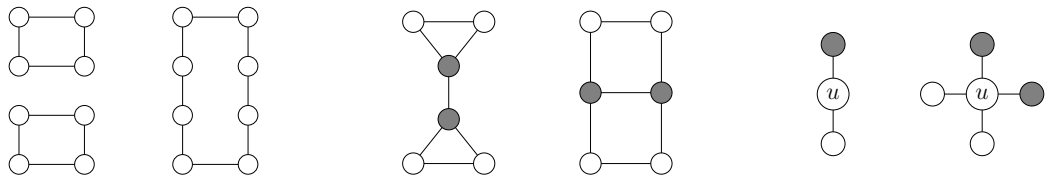
**Example 2.** Figure 2b shows another example of two graphs that cannot be separated by a WL test; note that node features simply correspond to the degrees of the nodes. From an algorithmic perspective, it is not hard to distinguish the two graphs from specific 1-dropout cases. Let $u$ and $v$ denote the two gray nodes in the graphs, and consider the process from $u$'s perspective. In both graphs, $u$ can recognize if $v$ is removed in a run since $u$ does not receive a "gray" message in the first round. However, the dropout of $v$ has a different effect in the two graphs later in the process: in the left-hand graph, it means that there is no gray neighbor at a 3-hop distance from $u$, while in the right-hand graph, $u$ will still see a gray node (itself) in a 3-hop distance.

Thus by identifying the 1-dropout of $v$, an algorithm can distinguish the two graphs: if we observe runs where $u$ receives no gray message in the first round, but it receives an (aggregated) gray message in the third round, then $u$ has the right-hand neighborhood. This also means that a sufficiently powerful GNN which is equivalent to the 1-WL test can also separate the two cases.

**Example 3.** Note that using a `sum` function for neighborhood aggregation is often considered a superior choice to `mean`, since $u$ cannot separate e.g. the two cases shown in Figure 2c with `mean` aggregation [27]. However, the `mean` aggregation of neighbors also has some advantages over `sum`; most notably, it does not grow with the degree of the node.

We show that dropouts also increase the expressive power of GNNs with `mean` aggregation, thus possibly making `mean` aggregation a better choice in some applications. In particular, a DropGNN with `mean` aggregation is still able to separate the two cases on Figure 2c.

Assume that the two colors in the figure correspond to feature values of 0 and 1, and let $p = \frac{1}{4}$. In the left-hand graph, there is a 1-dropout where $u$ ends up with a single neighbor of value 1; hence




(a) (b) (c)

Figure 2: Several example graphs which show that DropGNNs are more expressive than standard GNNs in various cases. Different node colors correspond to different node features.

mean aggregation yields a value of $1$ with probability $\frac{1}{4} \cdot \frac{3}{4} \approx 0.19$ in each run. However, in the right-hand graph, the only way to obtain a mean of $1$ is through a 2-dropout or some 3-dropouts; the total probability of these is only $0.06$ (see Appendix A for calculations). If we first transforms all other values to $0$ (e.g. with $\sigma(x - 0.5)$, where $\sigma$ is a step function), then we can use a run aggregation with `mean` (or `sum`) afterwards. The final embeddings in the two graphs will converge to $0.19$ and $0.06$ (multiplied by $r$ in case of `sum`), which allows us to distinguish the two cases.

# 4 Theoretical analysis

## 4.1 Required number of runs

We analyze DropGNNs with respect to the *neighborhood of interest* around a node $u$, denoted by $\Gamma$. That is, we select a specific region around $u$, and we want to ensure that the distribution of dropout combinations in this region is reasonably close to the actual probabilities. This choice of $\Gamma$ then determines the ideal choice of $p$ and $r$ in our DropGNN.

One natural choice is to select $\Gamma$ as the entire $d$-hop neighborhood of $u$, since a GNN will always compute its final values based on this region of the graph. Note that even for this largest possible $\Gamma$, the size of this neighborhood $\gamma := |\Gamma|$ does not necessarily scale with the entire graph. That is, even when our graph is large, $\gamma$ can still be reasonably small in practice: e.g. if both the degree of nodes and the number of layers $d$ is a small constant, then $\Gamma$ essentially has constant size.

However, we point out that $\Gamma$ can be freely chosen as a neighborhood of any specific size. That is, even if a GNN aggregates information from all nodes within a distance of $d = 5$ layers, we can still select $\Gamma$ to denote, for example, only the 2-hop neighborhood of $u$. The resulting DropGNN still computes a final node feature that is dependent on the entire 5-hop neighborhood of $u$; however, our DropGNN will now only ensure that we can observe a reasonable distribution of dropout combinations in the 2-hop neighborhood of $u$.

In this sense, the size $\gamma$ is essentially a trade-off hyperparameter: while a smaller $\gamma$ will require a smaller number of runs $r$ until the distribution of dropout combinations stabilizes, a larger $\gamma$ allows us to observe more variations of the region around $u$.

**1-complete dropouts.** From a strictly theoretical perspective, choosing a sufficiently large $r$ always allows us to observe every possible dropout combination. However, since the number of combinations is exponential in $\gamma$, this approach is not viable in practice (see Appendix B for more details).

To reasonably limit the number of necessary runs, we focus on the so-called 1-*complete case*: we want to have enough runs to ensure that at least every 1-dropout is observed a few times. Indeed, if we can observe each variant of $\Gamma$ where a single node is removed, then this might already allow a sophisticated algorithm to reconstruct a range of useful properties of $\Gamma$. Note that in all of our examples, a specific 1-dropout was already sufficient to distinguish the two cases.

For any specific node $v \in \Gamma$, the probability of a 1-dropout for $v$ is $p \cdot (1 - p)^{\gamma}$ in a run (including the probability that $u$ is not dropped out). We apply the $p$ value that maximizes the probability of such a 1-dropout; a simple differentiation shows that this maximum is obtained at $p^* = \frac{1}{1+\gamma}$.

This choice of $p$ also implies that the probability of observing a specific 1-dropout in a run is

$$\frac{1}{1+\gamma} \cdot \left(\frac{\gamma}{1+\gamma}\right)^{\gamma} \geq \frac{1}{1+\gamma} \cdot \frac{1}{e} \,.$$

Hence if we execute $r \geq e \cdot (\gamma + 1) = \Omega(\gamma)$ runs, then the expected number of times we observe a specific 1-dropout (let us denote this by $\mathbb{E}_1$) is at least $\mathbb{E}_1 \geq r \cdot \frac{1}{e} \cdot \frac{1}{1+\gamma} \geq 1$.

Moreover, one can use a Chernoff bound to show that after $\Omega(\gamma \log \gamma)$ runs, the frequency of each 1-dropout is sharply concentrated around $\mathbb{E}_1$. This also implies that we indeed observe each 1-dropout at least once with high probability.

For a more formal statement, let us consider a constant $\delta \in [0, 1]$ and an error probability $\frac{1}{t} < 1$. Also, given a node $v \in \Gamma$ (or subset $S \subseteq \Gamma$), let $X_v$ (or $X_S$) denote the number of times this 1-dropout ($|S|$-dropout) occurs during our runs.

**Theorem 1** *If $r \geq \Omega\left(\gamma \log \gamma t\right)$, then with a probability of $1 - \frac{1}{t}$, it holds that for each $v \in \Gamma$, we have $X_v \in [\,(1-\delta) \cdot \mathbb{E}_1\,,\,(1+\delta) \cdot \mathbb{E}_1\,]$.*

With slightly more runs, we can even ensure that each $k$-dropout for $k \geq 2$ happens less frequently than 1-dropouts. In this case, it already becomes possible to distinguish 1-dropouts from multiple-dropout cases based on their frequency.

**Theorem 2** *If $r \geq \Omega\left(\gamma^2 + \gamma \log \gamma t\right)$, then with a probability of $1 - \frac{1}{t}$ it holds that*

- *for each $v \in \Gamma$, we have $X_v \in [\,(1-\delta) \cdot \mathbb{E}_1\,,\,(1+\delta) \cdot \mathbb{E}_1\,]$,*
- *for each $S \subseteq \Gamma$ with $|S| \geq 2$, we have $X_S < (1-\delta) \cdot \mathbb{E}_1$.*

Since the number of all dropout combinations is in the magnitude of $2^{\gamma}$, proving this bound is slightly more technical. We discuss the proofs of these theorems in Appendix B.

Note that these theorems only consider the dropout distribution around a specific node $u$. To ensure the same properties for all $n$ nodes in the graph simultaneously, we need to add a further factor of $n$ within the logarithm to the number of necessary runs in Theorems 1 and 2. However, while this is only a logarithmic dependence on $n$, it might still be undesired in practice.

### 4.2 Expressive power of DropGNNs

In Section 3.4, we have seen that DropGNNs often succeed when a WL-test fails. It is natural to wonder about the capabilities and limits of the dropout approach in general; we study this question for multiple neighborhood aggregation methods separately.

We consider neighborhood aggregation with `sum` and `mean` in more detail; the proofs of the corresponding claims are discussed in Appendices C and D, respectively. Another popular aggregation function is `max`, which does not combine well with the dropout approach in practice; we also discuss this briefly in Appendix D.

**Aggregation with `sum`.** As mentioned before, `sum` neighborhood aggregation allows for an injective GNN design that distinguishes any two neighborhoods around a node $u$ [27]. As such, in a theoretical sense, `sum` aggregation has the same expressive power as a general $d$-hop distributed algorithm that works with the same distribution of observed dropout combinations.

It is already non-trivial to find two distinct neighborhoods that cannot be distinguished already in the 1-complete case. However, such an example exists; one can even construct a pair of graphs that produce the exact same distribution of 1- and 2-dropouts.

**Theorem 3** *There exists a pair of neighborhoods that cannot be distinguished by 1- and 2-dropouts.*

We illustrate a simpler example for only 1-dropouts in Figure 3. For a construction that also covers the case of 2-dropouts, the analysis is more technical; we defer this to Appendix C.

We note that even these more difficult examples can be distinguished with our dropout approach, based on their $k$-dropouts for larger $k$ values. However, this requires an even higher number of runs: we need to ensure that we can observe a reliable distribution even for these many-node dropouts.

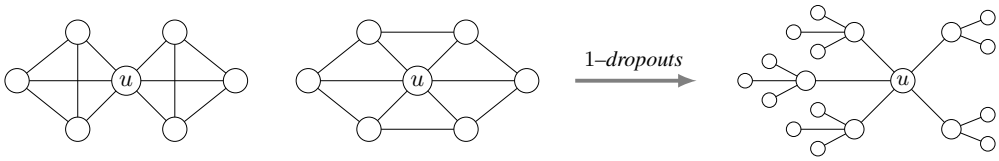



Figure 3: Example of two graphs not separable by 1-dropouts (left side). In both of the graphs, for any of the 1-dropouts, $u$ observes the same tree structure for $d = 2$, shown on the right side.

On the other hand, our dropout approach becomes even more powerful if we combine it e.g. with the extension by port numbers introduced in [20]. Intuitively speaking, port numbers allow an algorithm to determine all paths to the removed node in a 1-dropout, which in turn allows us to reconstruct the entire neighborhood of $u$. As such, in this case, 1-complete dropouts already allow us to distinguish any two neighborhoods.

**Theorem 4** *In the setting of Theorem 1, a DropGNN with port numbers can distinguish any two non-isomorphic $d$-hop neighborhoods.*

Finally, we note that the expressive power of DropGNNs in the 1-complete case is closely related to the *graph reconstruction problem*, which is a major open problem in theoretical computer science since the 1940s [11]. We discuss the differences between the two settings in Appendix C.

**Aggregation with `mean`.** We have seen in Section 3.4 that even with `mean` aggregation, DropGNNs can sometimes distinguish 1-hop neighborhoods (that is, multisets $S_1$ and $S_2$ of features) which look identical to a standard GNN. One can show that a similar separation is also possible in various other cases; e.g. we can distinguish any two multisets of the same size.

**Theorem 5** *Let $S_1 \neq S_2$ be two multisets of feature vectors with $|S_1| = |S_2|$. Then $S_1$ and $S_2$ can be distinguished by a DropGNN with `mean` neighborhood aggregation.*

However, in general, `mean` aggregation does not allow us to separate any two multisets based on the 1-complete case. In particular, we provide an example of distinct multisets with $S_1 \cap S_2 = \emptyset$ where the distribution of means obtained from $0-$ and $1-$dropouts is essentially identical in $S_1$ and $S_2$; thus the best we can hope for in this case is a more complex approach based on multiple-node dropouts.

# 5 Experiments

## 5.1 Datasets beyond WL

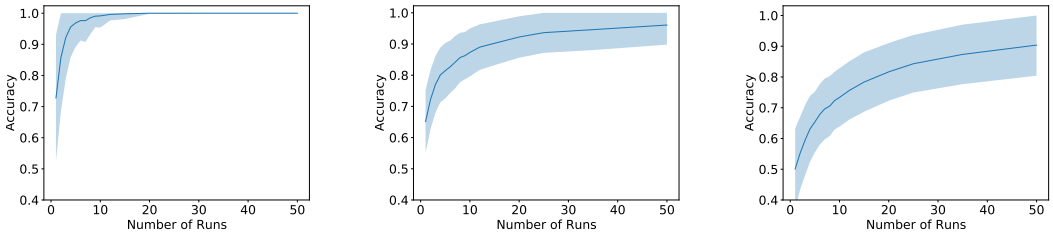
Let us now see the capabilities DropGNN in practice. We test on existing synthetic datasets, which are known to require expressiveness beyond the WL-test. We use the datasets from Sato et al. [21] that are based on $3-$regular graphs. Nodes have to predict whether they are part of a triangle (TRIANGLES) or have to predict their local clustering coefficient (LCC). We test on the two counterexamples LIMITS 1 (Figure 2a) and LIMITS 2 from Garg et al. [8] where we compare two smaller structures versus one larger structure. We employ the dataset by Loukas [14] to classify graphs on containing a cycle of length 4 (4-CYCLES). We increase the regularity in this dataset by connecting every node to one additional node. Finally we experiment on circles with skip edges (SKIP-CIRCLES) Chen et al. [5].

For comparison, we try several other GNN modifications which increase expressiveness. For control, we run a vanilla GNN on these datasets. We then extend this base GNN with (i) ports [20] (randomly assigned), (ii) node IDs [14] from $[0; n-1]$(randomly permuted), and (iii) a random feature from the standard normal distribution [21]. The architecture for all GNNs is a 5-layer GIN with "sum" as aggregation, identity as message function, and $\varepsilon = 0$. For DropGNN $r = 50$ runs are performed.

We train all methods for $1,000$ epochs and then evaluate the accuracy on the training set. We then test on a new graph (with new features). We report training and testing averaged across 10 initializations in Table 1. We can see that DropGNN outperforms the competition.

| | GIN | | +Ports | | +IDs | | +Random feat. | | +Dropout | |
|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | Train | Test | Train | Test | Train | Test | Train | Test | Train | Test |
| LIMITS 1 [8] | 0.5±0.0 | 0.5±0.0 | 0.5±0.0 | 0.5±0.0 | 0.92±0.12 | 0.57±0.15 | 1.0±0.0 | 0.53±0.14 | 1.0±0.0 | **0.92±0.06** |
| LIMITS 2 [8] | 0.5±0.0 | 0.5±0.0 | 0.5±0.0 | 0.5±0.0 | 0.9±0.17 | 0.49±0.26 | 1.0±0.0 | 0.55±0.1 | 1.0±0.0 | **0.89±0.16** |
| 4-CYCLES [14] | 0.5±0.0 | 0.5±0.0 | 0.5±0.0 | 0.5±0.0 | 0.73±0.13 | 0.57±0.15 | 1.0±0.0 | 0.52±0.06 | 0.97±0.06 | **0.91±0.14** |
| LCC [21] | 0.47±0.06 | 0.36±0.06 | 0.47±0.06 | 0.36±0.06 | 0.51±0.06 | 0.36±0.06 | 0.85±0.1 | 0.38±0.13 | 0.74±0.13 | **0.64±0.16** |
| TRIANGLES [21] | 0.64±0.02 | 0.65±0.01 | 0.64±0.02 | 0.65±0.01 | 0.65±0.02 | 0.63±0.05 | 0.92±0.06 | 0.46±0.12 | 0.88±0.1 | **0.87±0.1** |
| SKIP-CIRCLES [5] | 0.1±0.0 | 0.1±0.0 | 0.1±0.0 | 0.1±0.0 | 0.1±0.0 | 0.09±0.03 | 0.91±0.24 | 0.11±0.03 | 0.46±0.27 | **0.48±0.24** |

Table 1: Evaluation of techniques that increase GNN expressiveness on challenging synthetic datasets. We highlight the best test scores in bold. Using node IDs or random features achieves high training accuracy showing their expressiveness, but the model does not generalize to the test set. On the other hand, DropGNN achieves high training accuracy but also generalizes well to the test set.



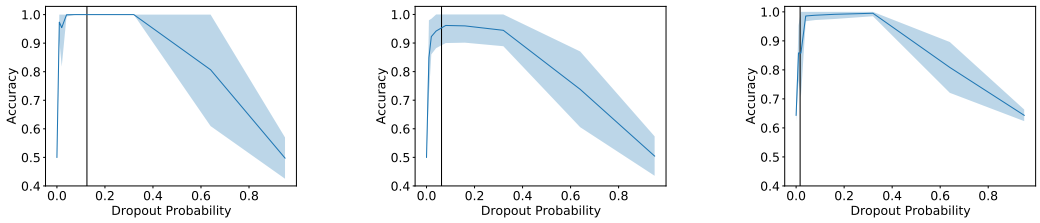

(a) LIMITS 1 (b) 4−CYCLES (c) TRIANGLES

Figure 4: Investigating the impact of the number of runs ($x-$axis) versus the classification accuracy ($y-axis$). In all three plots, having more runs, allows for more stable dropout observations, increasing accuracy. The tradeoff is higher runtime since the model computes more runs.

### 5.2 Sensitivity analysis

We investigate the impact of the number of independent runs on the overall accuracy. Generally, we expect an increasing number of runs to reliably produce informative dropouts. We train with a sufficiently high number of runs ($50$) with the same setting as before. Now, we reevaluate DropGNN but limit the runs to a smaller number. We measure the average accuracy over $10$ seeds each with $10$ tests and plot this average in Figure 4 on three datasets: Limits 1 (Figure 4a), $4$-cycles (Figure 4b), and Triangles (Figure 4c). In all three datasets, more runs directly translate to higher accuracy until the effect saturates around $1$. Thus, the number of runs is a hyperparameter following the compute-accuracy tradeoff.

Next, we investigate the impact of the dropout probability $p$. We repeat the setting from before, but instead of varying the number of runs in the reevaluation, train and test with different probabilities $p$ on an exponential scale from $0.01$ to $0.64$. We also try $0$ (no dropout) and $0.95$ (almost everything is dropped). Figure 5 shows the accuracy for each dropout probability, again averaged over $10$ seeds with $10$ tests each. Generally, DropGNN is robust to different values of $p$ until $p$ becomes very large.




(a) LIMITS 1 (b) 4−CYCLES (c) TRIANGLES

Figure 5: Investigating the impact of the dropout probability ($x-$axis) versus the classification accuracy ($y-$axis). DropGNN is robust to the choice of $p$ for decently small $p$. Choosing $p \approx \gamma^{-1}$ is a decent default that is shown by vertical black lines.

| Model | Complexity | MUTAG | PTC | PROTEINS | IMDB-B | IMDB-M |
|---|---|---|---|---|---|---|
| WL subtree [27; 23] | $O(n)$ | 90.4 ±5.7 | 59.9 ±4.3 | 75.0 ±3.1 | 73.8 ±3.9 | 50.9 ±3.8 |
| DCNN [2] | $O(n)$ | - | - | 61.3 ±1.6 | 49.1 ±1.4 | 33.5 ±1.4 |
| PatchySan [17] | $O(n)$ | 89.0 ±4.4 | 62.3 ±5.7 | 75.0 ±2.5 | 71.0 ±2.3 | 45.2 ±2.8 |
| DGCNN [29] | $O(n)$ | 85.8 ±1.7 | 58.6 ±2.5 | 75.5 ±0.9 | 70.0 ±0.9 | 47.8 ±0.9 |
| GIN [27] | $O(n)$ | 89.4 ±5.6 | 64.6 ±7.0 | 76.2 ±2.8 | ***75.1 ±5.1*** | ***52.3 ±2.8*** |
| DropGIN (ours) | $O(rn), r \approx 20$ | **89.9 ±6.4** | **65.1 ±6.5** | **76.3 ±5.6** | 74.9 ±3.8 | 51.7 ±4.6 |
| 1-2-3 GNN [16] | $O(n^4)$ | 86.1 | 60.9 | 75.5 | **74.2** | 49.5 |
| ProvablyPowerfullGNN [15]* | $O(n^3)$ | **90.6 ±8.7** | **66.2 ±6.5** | **77.2 ±4.7** | 73 ±5.8 | **50.5 ±3.6** |

Table 2: Graph classification accuracy (%). The best performing model in each complexity class is highlighted in bold. *We report the best result achieved by either of the three versions of their model.

### 5.3 Real-world datasets

We evaluate and compare our modified GIN model (DropGIN) with the original GIN model and other GNN models of various expressiveness levels on real-world graph classification datasets. We use three bioinformatics datasets (MUTAG, PTC, PROTEINS) and two social networks (IMDB-BINARY and IMDB-MULTI). Following [27] node degree is used as the sole input features for the IMDB datasets, while for the bioinformatics datasets the original categorical node features are used.

We follow exactly the evaluation and model selection protocol described in [27]. Following [28] 10-fold cross-validation accuracies are reported. We extend the original 5-layer GIN model described in [27] by including the sum aggregation of the different runs before the graph readout step. An auxiliary readout head is added to produce predictions based on each individual run. These predictions are used for an auxiliary loss term with a weight of $\frac{1}{3}$. We do not make any other changes to the architecture and use the same hyper-parameter selection as [27], see Appendix E for a detailed description. For our dropout technique, we set the number of runs to $m$ and choose the dropout probability to be $p = \frac{2}{m}$, where $m$ is the mean number of nodes of the graphs in the dataset. This is based on the assumption, that the 5-layer GIN will usually have the receptive field which covers the whole molecule and from Figure 5 we can see that it is usually safer to use a slightly larger $p$ than a slightly smaller one.

Our method successfully improves the results achieved by the original GIN model on the bioinformatics datasets (Table 2) and is, in general, competitive with the more complex and computationally expensive expressive GNNs. Namely, the 1-2-3 GNN [16] which has expressive power close to that of 3-WL with $O(n^4)$ computational and $O(n^4)$ space complexities, and the Provably Powerfull GNN [15] which has 3-WL expressive power with $O(n^3)$ computational and $O(n^2)$ space complexities. Compared to that, our method has $O(rn)$ computational and $O(n)$ space complexities. However, we do observe, that our approach slightly underperforms the original GIN model on the IMDB datasets. Since the other expressive GNNs also do not outperform the original GIN model, it is possible, that graphs in these datasets do not require higher expressiveness. In such cases, our model can lose accuracy compared to the base GNN as many runs are required to achieve a fully stable dropout distribution.

## 6 Conclusion

We have introduced a theoretically motivated DropGNN framework, which allows us to easily increase the expressive power of existing message passing GNNs, both in theory and practice. DropGNNs are also competitive with more complex GNN architectures which are specially designed to have high expressive power but have high computational complexity. In contrast, our framework allows for an arbitrary trade-off between expressiveness and computational complexity by choosing the number of rounds $r$ accordingly.

**Societal Impact.** In summary, we proposed a model-agnostic architecture improvement for GNNs. We do not strive to solve a particular problem but to enhance the GNN toolbox. Therefore, we do not see an immediate impact on society. We found in our experiments that DropGNN works best on graphs with smaller degrees, such as molecular graphs. Therefore, we imagine that using DropGNN in these scenarios is interesting to explore further; for example in the context of molecular graphs.

# References


[1] R. Abboud, İ. İ. Ceylan, M. Grohe, and T. Lukasiewicz. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.

[2] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. In *Advances in neural information processing systems*, pages 1993–2001, 2016.

[3] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[4] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[5] Z. Chen, L. Chen, S. Villar, and J. Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. *Advances in neural information processing systems*, 2019.

[6] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, 2016.

[7] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang. Graph random neural networks for semi-supervised learning on graphs. *Advances in Neural Information Processing Systems*, 33, 2020.

[8] V. Garg, S. Jegelka, and T. Jaakkola. Generalization and representational limits of graph neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3419–3430. PMLR, 13–18 Jul 2020.

[9] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML), Sydney, Australia*, Aug. 2017.

[10] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, 2017.

[11] F. Harary. A survey of the reconstruction conjecture. In R. A. Bari and F. Harary, editors, *Graphs and Combinatorics*, pages 18–28, Berlin, Heidelberg, 1974. Springer Berlin Heidelberg. ISBN 978-3-540-37809-9.

[12] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations ICLR, Toulon, France*, Apr. 2017.

[13] J. Klicpera, J. Groß, and S. Günnemann. Directional message passing for molecular graphs. In *International Conference on Learning Representations*, 2020.

[14] A. Loukas. What graph neural networks cannot learn: depth vs width. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[15] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. *arXiv preprint arXiv:1905.11136*, 2019.

[16] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[17] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023. PMLR, 2016.

[18] Y. Rong, W. Huang, T. Xu, and J. Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[19] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[20] R. Sato, M. Yamada, and H. Kashima. Approximation ratios of graph neural networks for combinatorial problems. In *Neural Information Processing Systems (NeurIPS)*, 2019.

[21] R. Sato, M. Yamada, and H. Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341. SIAM, 2021.

[22] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 2008.

[23] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[24] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR), Vancouver, BC, Canada*, May 2018.

[25] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[26] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning (ICML), Stockholm, Sweden*, July 2018.

[27] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[28] P. Yanardag and S. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015.

[29] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

# Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] The theoretical limits of the approach are explicitly discussed in Section 4.2 and Appendices C–D.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] The impacts are discussed in the Conclusion (Section 6)
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 4 for the assumptions. For some of the theorems, Appendices B–D also contain a more detailed discussion of these assumptions.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendices B–D.

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Refer to the README in the added repository
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Standard deviations in tables, error regions in plots.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cite in particular the libraries PyTorch, DGL, and PytorchGeometric
   (b) Did you mention the license of the assets? [No] We did not use any proprietary assets but only libraries well known and spread in the community (PyTorch, DGL, and PytorchGeometric)
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Code
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] Datasets are synthetic or publicly available for benchmarking.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] No offensive content.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] There is no such aspect to our work.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]