DRew: Dynamically Rewired Message Passing with Delay

Benjamin Gutteridge [1]   Xiaowen Dong [1]   Michael Bronstein [2]   Francesco Di Giovanni [3][4]

Abstract

Message passing neural networks (MPNNs) have been shown to suffer from the phenomenon of *over-squashing* that causes poor performance for tasks relying on long-range interactions. This can be largely attributed to message passing only occurring locally, over a node's immediate neighbours. Rewiring approaches attempting to make graphs 'more connected', and supposedly better suited to long-range tasks, often lose the inductive bias provided by distance on the graph since they make distant nodes communicate *instantly* at *every* layer. In this paper we propose a framework, applicable to any MPNN architecture, that performs a *layer-dependent rewiring* to ensure *gradual* densification of the graph. We also propose a *delay* mechanism that permits skip connections between nodes depending on the layer *and* their mutual distance. We validate our approach on several long-range tasks and show that it outperforms graph Transformers and multi-hop MPNNs.

1. Introduction

Graph Neural Networks (GNNs) (Sperduti, 1993; Gori et al., 2005; Scarselli et al., 2008; Bruna et al., 2014), deep learning architectures that operate on graph-structured data, are significantly represented by the message-passing paradigm (Gilmer et al., 2017), in which layers consisting of a local neighbourhood aggregation are stacked to form Message Passing Neural Networks (MPNNs). The most commonly used MPNNs (henceforth referred to as 'classical'), perform only *local* aggregation, with information being shared at each layer only between nodes that are immediate neighbours (i.e., directly connected by an edge). Accordingly,

for nodes that are distant from one another to share information, that information must 'flow' through the graph at a rate of one edge per layer, necessitating appropriately deep networks when such 'long-range interactions' are required for solving the task at hand (Barceló et al., 2019). Unfortunately, this often leads to poor model performance, as deep MPNNs are particularly prone to the phenomena of *over-squashing* (Alon & Yahav, 2021; Topping et al., 2022; Di Giovanni et al., 2023) and *over-smoothing* (Nt & Maehara, 2019; Oono & Suzuki, 2020).

In this paper, we introduce **Dynamically Rewired Message Passing** (DRew), a novel framework for layer-dependent, multi-hop message passing that takes a principled approach to information flow, is robust to over-squashing, and can be applied to any MPNN for deep learning on graphs.

Contributions. First, we formalize DRew, a new framework of aggregating information over distant nodes that goes beyond the limitations of classical MPNNs, but respects the inductive bias provided by the graph: nodes that are *closer* should interact *earlier* in the architecture. Second, we introduce the concept of **delay** for message passing, controlled by a tunable parameter ν, and generalize DRew to account for delay in order to alleviate issues such as over-smoothing arising from deep MPNN-architectures; we call this framework νDRew. [1] Third, we present a theoretical analysis that proves that our proposed frameworks can mitigate over-squashing. Lastly, we experimentally evaluate our framework on both synthetic and real-world datasets.[2] Our experiments demonstrate the robustness of DRew and the effectiveness of delayed propagation when applied to deep MPNN architectures or long-range tasks.

2. Message Passing Neural Networks

In this section we introduce the class of Message Passing Neural Networks and discuss some of its main limitations. We first review some important notions concerning graphs.

2.1. Preliminaries

Let $G = (V, E)$ be a graph consisting of nodes V and edges E. We assume that G is undirected and connected. The

---

[1]Department of Engineering Science, University of Oxford [2]Department of Computer Science, University of Oxford [3]Department of Computer Science and Technology, University of Cambridge [4]Faculty of Informatics, University of Lugano. Correspondence to: Benjamin Gutteridge <beng@robots.ox.ac.uk>.

---

[1]Pronounced 'Andrew'.
[2]https://github.com/BenGutteridge/DRew
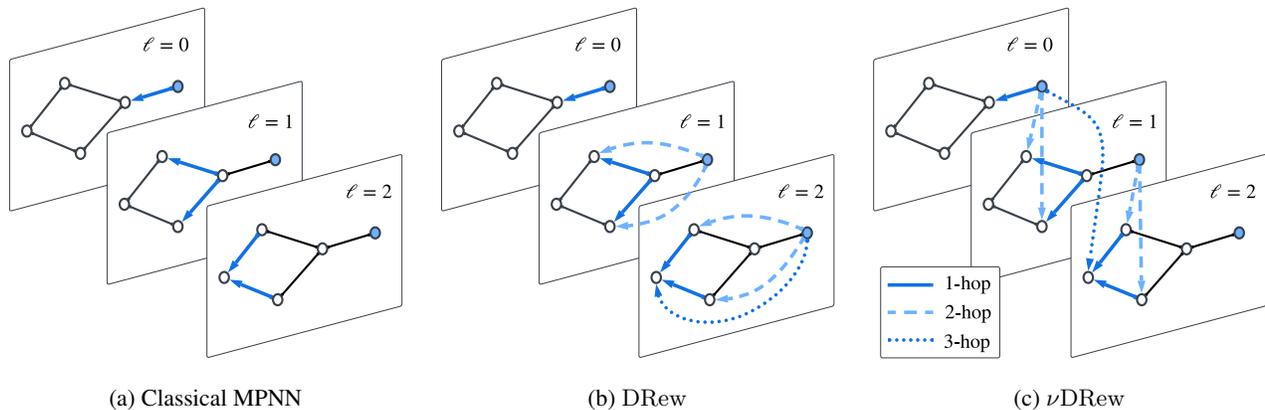
Figure 1. Illustration of the graph across three layers $\ell \in \{0, 1, 2\}$ for (a) a classical MPNN, (b) DRew and (c) νDRew. We choose a source node (coloured blue) on which to focus and demonstrate information flow from this node at each layer. We use arrows to denote direction of information transfer and specify hop-connection distance. In the classicical MPNN setting, at every layer information only travels from a node to its immediate neighbours. In DRew, the graph changes based on the layer, with newly added edges connecting nodes at distance r from layer $r - 1$ onward. Finally, in νDRew, we also introduce a delay mechanism equivalent to skip-connections between *different* nodes based on their mutual distance (see Section 3.3).

structure of the graph is encoded in the adjacency matrix $\mathbf{A} \subset \mathbb{R}^{n \times n}$, with number of nodes $n = |V|$. The simplest quantity measuring the connectivity of a node is the *degree*, which can be computed as $d_i = \sum_j A_{ij}$, for $i \in V$. The notion of 'locality' in G is induced by the *shortest walk* (or *geodesic*) *distance* $d_G : V \times V \to \mathbb{R}_{\geq 0}$, which assigns the length of the minimal walk connecting any given pair of nodes. If we fix a node i, the distance allows us to partition the graph into level sets of $d_G(i, \cdot)$ which we refer to as *k-hop* (or *k-neighbourhood*) and denote by

$$\mathcal{N}_k(i) := \{j \in V : d_G(i, j) = k\}.$$

$\mathcal{N}_1(i)$ is the set of immediate (or 1-hop) neighbours of node i. We stress that in our notations, the k-hop of a node i represents the nodes at distance *exactly* k — a subset of the nodes that can be reached by a walk of length k.

The MPNN class. Consider a graph G with node features $\{h_i \in \mathbb{R}^d, i \in V\}$ and assume we are interested in predicting a quantity (or label) y_{G_i}. Typically a GNN processes both the topological data G and the feature information $\mathbf{H} \in \mathbb{R}^{n \times d}$ via a sequence of layers, before applying a readout map to output a final prediction h_G. The most studied GNN paradigm is MPNNs (Gilmer et al., 2017), where the layer update is given by

$$
\begin{aligned}
a_i^{(\ell)} &= \text{AGG}^{(\ell)} \left(\{h_j^{(\ell)} : j \in \mathcal{N}_1(i)\} \right), \\
h_i^{(\ell+1)} &= \text{UP}^{(\ell)} \left(h_i^{(\ell)}, a_i^{(\ell)} \right),
\end{aligned}
\tag{1}
$$

for learnable update and aggregation maps UP and AGG. While the choice of the maps UP and AGG may change

across specific architectures (Bresson & Laurent, 2017; Hamilton et al., 2017; Kipf & Welling, 2017; Veličković et al., 2018), in all MPNNs messages travel from one node to its 1-hop neighbours *at each layer*. Accordingly, for a node i to exchange information with node $j \in \mathcal{N}_k(i)$, we need to stack at least k layers. In Section 3, we discuss how the interaction between two node representations should, in fact, change based on both their mutual distance *and* their state in time (i.e. the layer of the network). We argue that it is important **not simply *how*** two node states interact with each other, **but also *when*** that happens.

2.2. Long-range dependencies and network depth

A task exhibits *long-range interactions* if, to be solved, there exists some node i whose representation needs to account for the information contained at a node j with $d_G(i, j) \gg 1$ (Dwivedi et al., 2022). MPNNs rely on 1-hop message propagation, so to capture such non-local interactions, multiple layers must be stacked; however, this leads to undesirable phenomena with increasing network depth. We focus on one such problem, known as over-squashing, below.

Over-squashing. In a graph, the number of nodes in the receptive field of a node i often expands exponentially with hop distance k. Accordingly, for i to exchange information with its k-hop neighbours, an exponential volume of messages must pass through fixed-size node representations, which may ultimately lead to a loss of information (Alon & Yahav, 2021). This problem is known as *over-squashing*, and has been characterized via sensitivity analysis (Topping et al., 2022). Methods to address the over-squashing problem typically resort to some form of *graph rewiring*, in the

sense that the graph used for message passing is (partly) decoupled from the input one. A 'local' form of graph rewiring consists in aggregating over multiple hops at each layer layer (Abu-El-Haija et al., 2019; 2020; Zhang & Li, 2021; Abboud et al., 2022). A 'global' form of graph rewiring is taken to the extreme in graph Transformers (Dwivedi & Bresson, 2020; Ying et al., 2021; Kreuzer et al., 2021; Rampášek et al., 2022), which replace the input graph with a complete graph where every pair of nodes is connected by an attention-weighted edge. Transformers, however, are computationally expensive and tend to throw away information afforded by the graph topology. Since all nodes can interact in a single layer, any notion of locality induced by distance d_G is discarded and must be rediscovered implicitly via positional and structural encoding.

Over-smoothing and other phenomena. The use of deep MPNNs gives rise to other issues beyond over-squashing. A well-known problem is *over-smoothing*, where, in the limit of many layers, features become indistinguishable (Nt & Maehara, 2019; Oono & Suzuki, 2020). While over-smoothing is now fairly understood and has been formally characterized in recent works (Bodnar et al., 2022; Cai & Wang, 2020; Di Giovanni et al., 2022; Rusch et al., 2022), it is unclear whether the often observed degradation in performance with increasing depth is mainly caused by over-smoothing, over-squashing, or more classical vanishing gradient problem (Di Giovanni et al., 2023). It is hence generally desirable to propose frameworks that are not just robust to depth, but can actually adapt to the underlying task, either by 'fast' exploration of the graph in fewer layers or by 'slow' aggregation through multiple layers. *We introduce a new framework for message-passing that can accomplish this thanks to two principles: (i) dynamically rewired message passing and (ii) a delay mechanism.*

3. Dynamically Rewired MPNNs

In this section we introduce our framework to handle the aggregation of messages in MPNNs. We discuss how MPNNs present a 'static' way of collecting messages at each layer which is ultimately responsible for over-squashing. By removing such static inductive bias, we unlock a physics-inspired way for MPNNs to exchange information that is more suited to handle long-range interactions.

Information flow in MPNNs. Consider two nodes $i, j \in V$ at distance r. In a classic MPNN, these two nodes start interacting at layer r, meaning that

$$\min\left\{\ell : \frac{\partial h_i^{(\ell)}}{\partial h_j^{(0)}} \neq 0\right\} \geq d_G(i, j). \qquad (2)$$

In fact, since the aggregation at each layer is computed

using the same graph G, one can bound such interaction with powers of the adjacency $\mathbf{A}$ as used in Topping et al. (2022)

$$\left|\frac{\partial h_i^{(r)}}{\partial h_j^{(0)}}\right| \leq c\left(\mathbf{A}^r\right)_{ij}, \qquad (3)$$

with constant c depending only on the Lipschitz-regularity of the MPNN and independent of the graph topology. We see that communication between i and j must be filtered by intermediate nodes that are traversed along each path connecting i to j. This, in a nutshell, is the reason behind over-squashing; indeed, the bound in Eq. (3) may decay exponentially with the distance r whenever $\mathbf{A}$ is degree-normalized. By the same argument, in an MPNN two nodes at distance r always interact with a latency or *delay of exactly* r, meaning that for any intermediate state ℓ_0 we have

$$\min\left\{\ell : \frac{\partial h_i^{(\ell)}}{\partial h_j^{(\ell_0)}} \neq 0\right\} \geq \ell_0 + d_G(i, j), \qquad (4)$$

and similar Jacobian bounds apply in this case. Accordingly, in a classic MPNN we have two problems:

(i) Distant nodes can only communicate by exchanging information with their neighbours.

(ii) Distant nodes always interact with a fixed delay given by their distance.

Information flow in multi-hop MPNNs. The first issue can, in principle, be easily addressed by *rewiring* the graph via a process where any pair of nodes within a certain threshold are connected via an edge, and which can generally be given its own encoding or weight (Abboud et al., 2022; Brüel-Gabrielsson et al., 2022; Rampášek et al., 2022). In this way, distant nodes can now exchange information *directly*; this avoids iterating messages through powers of the adjacency and hence mitigates over-squashing by reducing the exponent in Eq. (3). However, this process brings about two phenomena which could lead to undesirable effects: (i) the computational graph is much denser — with implications for efficiency — and (ii) most of the inductive bias afforded by the graph distance information is thrown away, given that nodes i, j at distance r are now able to interact *at each layer* of the architecture, without any form of latency. In particular, this **static** rewiring, where the computational graph is densely 'filled' from the first MPNN layer, prevents messages from being sent *first* among nodes that are closer together in the input graph.

3.1. A new framework: (ν)DRew **message passing**

Dynamic rewiring. We start by addressing the limitation of MPNNs that nodes can only communicate through intermediate neighbours. To motivate our framework, take two

nodes $i, j \in V$ at distance $r > 1$. For classical MPNNs, we must wait for r layers (i.e. time units *with respect to the architecture*) before i and j can interact with each other. As argued above, this preserves information about locality and distance induced by the input graph, since nodes that are *closer* communicate *earlier*; however, since the two nodes have waited 'long enough', we argue that they should interact directly without necessarily relaying messages to their neighbours first. Accordingly, given update and aggregation functions as per the MPNN paradigm in Eq. (1), we define the update in a **Dynamically Rewired (DRew-)MPNN** by:

$$a_{i,k}^{(\ell)} = \text{AGG}_k^{(\ell)}\Big(\{h_j^{(\ell)} : j \in \mathcal{N}_k(i)\}\Big), 1 \leq k \leq \ell+1$$
$$h_i^{(\ell+1)} = \text{UP}_k^{(\ell)}\Big(h_i^{(\ell)}, a_{i,1}^{(\ell)}, \ldots, a_{i,\ell+1}^{(\ell)}\Big). \quad (5)$$

Some important comments: first, if $\text{AGG}_k = I$ for each $k > 1$, this reduces to the classical MPNN setting. Second, unlike augmented MPNNs, the sets over which we compute aggregation *differ depending on the layer*, with the hop $\mathcal{N}_k(i)$ only being added from the k-th layer on. So, while this framework shares similarities with other multi-hop MPNN architectures like Abboud et al. (2022), it features a novel mechanism: dynamic rewiring of the graph at each layer ℓ to include aggregation from each k-hop within distance $\ell+1$. For example, at the first layer, $\ell = 0$, our DRew layer is identical to the base MPNN represented by the choice of UP and AGG, but at each subsequent layer the receptive field of node i expands by 1 hop. This allows distant nodes to exchange information without intermediate steps, hence solving one of the problems of the MPNN paradigm, but also preserving the inductive bias afforded by the topology since the graph is filled gradually according to distance rather than treating each layer in the same way.

DRew-*MPNN explores the middle ground between classical MPNNs and methods like graph Transformers that consider all pairwise interactions at once*.

The delay mechanism. Next, we generalize DRew-MPNN to also account for whether nodes should interact with fixed (if any) delay. Currently, we have two opposing scenarios: in MPNNs, nodes interact with a constant delay given by their distance – leading to the same lag of information – while in DRew, nodes interact only from a certain depth of the architecture, but without any delay. For DRew, two nodes i, j at distance r communicate directly after r layers, since information has now been able to travel from j to i. But what if we consider the state of j as it was when the information 'left' to flow towards i? We account for an extra degree of freedom τ representing the **delay** of messages exchanged among nodes at distance r in DRew. If $\tau = 0$, then nodes at distance k interact at the k-th layer without delay, i.e *instantly* as per Eq. (5), otherwise node i 'sees' the state of j at the k-th layer but *delayed* by $\tau := k - \nu$, for some

ν. We formalize this by introducing $\tau_\nu(k) = \max(0, k - \nu)$ and generalize DRew as

$$a_{i,k}^{(\ell)} = \text{AGG}_k^{(\ell)}\Big(\{h_j^{(\ell-\tau_\nu(k))} : j \in \mathcal{N}_k(i)\}\Big), 1 \leq k \leq \ell+1$$
$$h_i^{(\ell+1)} = \text{UP}_k^{(\ell)}\Big(h_i^{(\ell)}, a_{i,1}^{(\ell)}, \ldots, a_{i,\ell+1}^{(\ell)}\Big). \quad (6)$$

If there is no delay, i.e. $\nu = \infty$, then we recover Eq. (5). The opposite case is given by $\nu = 1$, so that at layer ℓ and for any j at distance k, node i receives 'delayed' representation $h_j^{(\ell-\tau_1(k))}$, i.e. the state of j as it was $k-1$ layers ago. From now on, we refer to Eq. (6) as νDRew. We also note that in our experiments we treat ν as a tunable hyperparameter.

*Delay allows for expressive control of information flow. No delay means that messages travel faster, with distant nodes interacting instantly once an edge is added; conversely, the more delay, the slower the information flow, with distant nodes accessing **past** states when an edge is added.*

Our framework *generalizes any MPNN* since it acts on the computational graph (which nodes exchange information and *when*) and does not govern the architecture (see Section 3.3). We describe three instances of νDRew below.

3.2. Instances of our framework

In this section we provide examples for the νDRew-MPNN template in Eq. (6) for three classical MPNNs: GCN (Kipf & Welling, 2017), GIN (Xu et al., 2019) and GatedGCN (Bresson & Laurent, 2017). We will use these variants for our experiments in Section 5. For νDRew-GCN, we write the layer-update as

$$h_i^{(\ell+1)} = h_i^{(\ell)} + \sigma\left(\sum_{k=1}^{\ell+1} \sum_{j \in \mathcal{N}_k(i)} \mathbf{W}_k^{(\ell)} \gamma_{ij}^k h_j^{(\ell-\tau_\nu(k))}\right), \quad (7)$$

where σ is a pointwise nonlinearity, $\mathbf{W}_k^{(\ell)}$ are learnable channel-mixing matrices for the convolution at layer ℓ on the k-neighbourhood, and $\mathbf{\Gamma}^k \subset \mathbb{R}^{n \times n}$ are matrices with elements

$$\gamma_{ij}^k = \begin{cases} \frac{1}{\sqrt{d_i d_j}}, & \text{if } d_G(i,j) = k \\ 0, & \text{otherwise.} \end{cases} \quad (8)$$

We note again that if $j \in \mathcal{N}_k(i)$, then i, j only communicate from layer k onward, while ν determines the communication delay. The choice of degree normalization for $\mathbf{\Gamma}^k$ is to provide a consistent normalization for all terms.

We define νDRew-GIN in a similar fashion, as per Eq. (6); the layer update used below is inspired by Brockschmidt

(2020) and Abboud et al. (2022):

$$h_i^{(\ell+1)} = (1 + \epsilon)\mathrm{MLP}_s^{(\ell)}(h_i^{(\ell)})$$
$$+ \sum_{k=1}^{\ell+1} \sum_{j \in \mathcal{N}_k(i)} \mathrm{MLP}_k^{(\ell)}(h_j^{(\ell-\tau_\nu(k))}), \qquad (9)$$

where an MLP is one or more linear layers separated by ReLU activation and ϵ is a weight parameter. $\mathrm{MLP}_s^{(\ell)}$ is the self-loop (or residual) aggregation while $\mathrm{MLP}_k^{(\ell)}$ operates on the k-neighbourhood at layer ℓ.

Lastly, we define νDRew-GatedGCN as follows:

$$h_i^{(\ell+1)} = \mathbf{W}_1^{(\ell)} h_i^{(\ell)} + \sum_{k=1}^{\ell+1} \sum_{j \in \mathcal{N}_k(i)} \eta_{i,j}^k \odot \mathbf{W}_2^{(\ell)} h_j^{(\ell-\tau_\nu(k))},$$

$$\eta_{i,j}^k = \frac{\hat{\eta}_{i,j}^k}{\sum_{j \in \mathcal{N}_k(i)} (\hat{\eta}_{i,j}^k) + \epsilon},$$

$$\hat{\eta}_{i,j}^k = \sigma\left(\mathbf{W}_3^{(\ell)} h_i^{(\ell)} + \mathbf{W}_4^{(\ell)} h_j^{(\ell-\tau_\nu(k))}\right),$$
$$(10)$$

where σ is the sigmoid function, $\odot$ is the element-wise product, ϵ is a small fixed constant for numerical stability and $\mathbf{W}_1^\ell, \mathbf{W}_2^\ell, \mathbf{W}_3^\ell, \mathbf{W}_4^\ell$ are learned channel-mixing matrices. We note that in Eq. (10), unlike Eq. (7) and Eq. (9), weight matrices are shared between k-hop neighbourhoods. We do this because k-neighbourhood weight sharing achieves comparably strong results with non-weight-shared DRew-GatedGCN (see Section 5) while maintaining a lower parameter count, whereas we see performance drops when using weight sharing for DRew-GCN and -GIN. This can be explained by the edge gates $\eta_{i,j}^k$ serving as a soft-attention mechanism (Dwivedi et al., 2020) not present in GCN and GIN, affording shared weights more flexibility to model relationships between nodes at varying hop distances.

3.3. The graph-rewiring perspective: νDRew as distance-aware skip-connections

We conclude this section by providing an alternative explanation of νDRew from a graph-rewiring perspective. Given an underlying MPNN, we study how information travels in the graph at each layer. Referring to Figure 1 to illustrate our explanation, we say that messages travel **horizontally** when they move inside a layer (slice), and that they move **vertically** when they travel across different layers (slices). In a classical MPNN, the graph adopted at each layer coincides with the input graph G; information can only travel horizontally from a node to its 1-hop neighbours. In the DRew setting — which we recall to be the version of νDRew without delay (i.e. $\nu = \infty$) — the graph changes *depending on the layer*: this amounts to a dynamic rewiring where G is replaced with a sequence $\{\mathcal{R}_k(G)\}$, where at each layer ℓ we

add edges between any node i and $\mathcal{N}_{\ell+1}(i)$. Messages only travel horizontally as before, but the graph is *progressively filled* with each layer. Finally, in the delayed version of Eq. (6), messages can travel both horizontally *and* vertically, meaning that we are also 'rewiring' the graph along the time (layer) axis. Residual connections can also be thought of as allowing information to move 'vertically', though such connections are only made between the *same* node i at *different* layers; typically $\ell, \ell + 1$. From this perspective, the νDRew framework **is equivalent to adding geometric skip connections** among *different* nodes based on their distance. This is a powerful mechanism that combines skip-connections, a key tool in architecture design, with metric information provided by the geometry of the data; in this case the distances between vertices in a graph G.

4. Why νDRew Improves Information Processing

Mitigating over-squashing. In this section we discuss why the νDRew framework mitigates over-squashing and is hence more suited to handle long-range interactions in a graph. We focus on the case of maximal delay $\nu = 1$. We also restrict our discussion to Eq. (7): νDRew-GCN, though the conclusion extends easily to any νDRew-MPNN. Consider nodes $i, j \in V$ at distance r. For a traditional MPNN, i, j first exchange information at layer r, meaning that Eq. (2) is satisfied; however, a crucial difference from the MPNN paradigm is given by the addition of distance-aware skip connections between i, j as in Eq. (6). One can extend the approach from Topping et al. (2022) and derive

$$\left| \frac{\partial h_i^{(r)}}{\partial h_j^{(0)}} \right| \leq C\left(\sum_{k_1 + \cdots + k_\ell = r} \left(\prod_{k_1,\ldots,k_\ell} (\gamma^k)_{ij} \right) \right),$$

recalling that matrices $\mathbf{\Gamma}^k$ are defined in Eq. (8). We see how, differently from the standard MPNN formalism, nodes at distance r can now interact via products of message-passing matrices containing *fewer* than r factors. In fact, the right-hand side also accounts for a direct interaction between i, j via the matrix $\mathbf{\Gamma}^r$. Topping et al. (2022) showed that over-squashing arises precisely due to the entries ij of $\mathbf{A}^r$ decaying to zero exponentially with the distance, r, for (normalized) message-passing matrices $\mathbf{A}$; on the other hand, using matrices like $\mathbf{\Gamma}^r$, which are not powers of the same adjacency matrix, mitigates over-squashing.

Interpreting delay as local smoothing. We comment here on a slightly different perspective from which to understand the role of delay in νDRew. As usual, we consider nodes i, j at distance r. In our framework, node i starts collecting messages from j starting from the r-th layer. A larger delay (i.e. smaller value of ν), means that i aggregates the features from j before they are (significantly) 'smoothed'

Table 1. Classical MPNN benchmarks vs their DRew variants (without positional encoding) across four LRGB tasks: (from left to right) graph classification, graph regression, link prediction and node classification. All results are for the given metric on test data.

Model	Peptides-func AP ↑	Peptides-struct MAE ↓	PCQM-Contact MRR ↑	PascalVOC-SP F1 ↑
GCN	0.5930±0.0023	0.3496±0.0013	0.3234±0.0006	0.1268±0.0060
+DRew	**0.6996±0.0076**	**0.2781±0.0028**	**0.3444±0.0017**	**0.1848±0.0107**
GINE	0.5498±0.0079	0.3547±0.0045	0.3180±0.0027	0.1265±0.0076
+DRew	**0.6940±0.0074**	**0.2882±0.0025**	**0.3300±0.0007**	**0.2719±0.0043**
GatedGCN	0.5864±0.0077	0.3420±0.0013	0.3218±0.0011	0.2873±0.0219
+DRew	**0.6733±0.0094**	**0.2699±0.0018**	**0.3293±0.0005**	**0.3214±0.0021**

by repeated message passing. Conversely, a smaller delay (i.e. larger value of ν), implies that when i communicates with j, it also leverages the structure around j which has been encoded in the representation of j via the earlier layers. Therefore, we see how beyond mitigating over-squashing, the delay offers an extra degree of freedom to our framework, which can be used to *adapt to the underlying task and how quickly the graph topological information needs to be mixed across different regions*.

Expressivity. As multi-hop aggregations used in νDRew are based on shortest path distances, they are able to distinguish any pair of graphs distinguished by the shortest path kernel (Abboud et al., 2022; Borgwardt & Kriegel, 2005). Shortest path can distinguish disconnected graphs, a task at which 1-WL (Weisfeiler & Leman, 1968), which bounds the expressiveness of classical MPNNs (Xu et al., 2019), fails. We can therefore state that, at minimum, (ν)DRew is more expressive than 1-WL and, therefore, classical MPNNs. We leave more detailed expressivity analysis for future work.

5. Empirical Analysis

In this section we focus on two strengths of our model. First, we validate **performance** in comparison with benchmark models, including vanilla and multi-hop MPNNs and graph Transformers, over five real-world tasks spanning graph-, node- and edge-level tasks. Second, we validate the **robustness** of νDRew for long-range-dependent tasks and increased-depth architectures, using a synthetic task and a real-world molecular dataset. All code to reproduce experimental results is available at https://github.com/BenGutteridge/DRew.

Parameter scaling. We note here that many of the DRew-MPNNs used in our experiments exhibit parameter scaling of approximately $L^2/2$ for network depth L, whereas MPNNs scale with L. For fair comparison, a *fixed parameter budget* is maintained for all performance experiments across all network depths via suitable adjustment of hidden

dimension d, for both MPNNs and Transformers – we reserve the exploration of optimal sharing of weights for future work. We discuss space-time complexity in Appendix A.

5.1. Long-range graph benchmark

The Long Range Graph Benchmark (LRGB; Dwivedi et al. (2022)) is a set of GNN benchmarks involving long-range interactions. We provide experiments for three datasets from this benchmark (two molecular property prediction, one image segmentation) spanning the full range of tasks associated with GNNs: graph regression (Peptides-func), graph classification (Peptides-struct), link prediction (PCQM-Contact) and node classification (PascalVOC-SP). The tasks presented by LRGB are characterised as possessing long-range dependencies according to the criteria of (a) graph size (i.e. having a large number of nodes), (b) requiring a long range of interaction, and (c) output sensitivity to global graph structure. We compare our DRew-MPNN variants from Section 3.2 against classical MPNN benchmarks in Table 1, and against a range of models in Table 2, including classical and DRew MPNN variants, graph Transformers (Dwivedi et al., 2022; Rampášek et al., 2022), a multi-hop baseline (MixHop-GCN; Abu-El-Haija et al. (2019)) and a static graph rewiring benchmark (DIGL; Klicpera et al. (2019)).

Experimental details. All experiments are averaged over three runs and were allowed to train for 300 epochs or until convergence. Classical MPNN and graph Transformer results are reproduced from Dwivedi et al. (2022), except GraphGPS which is reproduced from Rampášek et al. (2022). DRew-MPNN, DIGL and MixHopGCN models were trained using similar hyperparameterisations to their classical MPNN counterparts (see Appendix B. Some models include positional encoding (PE), either Laplacian (LapPE; Dwivedi et al. (2020)) or Random Walk (RWSE; Dwivedi et al. (2021)), as this improves performance and is necessary to induce a notion of locality in Transformers. We provide the performance of the best-case νDRew model

Table 2. Performance of various classical, multi-hop and static rewiring MPNN and graph Transformer benchmarks against DRew-MPNNs across four LRGB tasks. The **first-**, **second-** and **third-**best results for each task are colour-coded; models whose performance are within a standard deviation of one another are considered equal.

Model	Peptides-func AP ↑	Peptides-struct MAE ↓	PCQM-Contact MRR ↑	PascalVOC-SP F1 ↑
GCN	0.5930±0.0023	0.3496±0.0013	0.3234±0.0006	0.1268±0.0060
GINE	0.5498±0.0079	0.3547±0.0045	0.3180±0.0027	0.1265±0.0076
GatedGCN	0.5864±0.0077	0.3420±0.0013	0.3218±0.0011	0.2873±0.0219
GatedGCN+PE	0.6069±0.0035	0.3357±0.0006	0.3242±0.0008	0.2860±0.0085
DIGL+MPNN	0.6469±0.0019	0.3173±0.0007	0.1656±0.0029	0.2824±0.0039
DIGL+MPNN+LapPE	0.6830±0.0026	**0.2616±0.0018**	0.1707±0.0021	0.2921±0.0038
MixHop-GCN	0.6592±0.0036	0.2921±0.0023	0.3183±0.0009	0.2506±0.0133
MixHop-GCN+LapPE	0.6843±0.0049	**0.2614±0.0023**	0.3250±0.0010	0.2218±0.0174
Transformer+LapPE	0.6326±0.0126	**0.2529±0.0016**	0.3174±0.0020	0.2694±0.0098
SAN+LapPE	0.6384±0.0121	0.2683±0.0043	**0.3350±0.0003**	**0.3230±0.0039**
GraphGPS+LapPE	0.6535±0.0041	**0.2500±0.0005**	0.3337±0.0006	**0.3748±0.0109**
DRew-GCN	**0.6996±0.0076**	0.2781±0.0028	**0.3444±0.0017**	0.1848±0.0107
DRew-GCN+LapPE	**0.7150±0.0044**	**0.2536±0.0015**	**0.3442±0.0006**	0.1851±0.0092
DRew-GIN	**0.6940±0.0074**	0.2799±0.0016	0.3300±0.0007	0.2719±0.0043
DRew-GIN+LapPE	**0.7126±0.0045**	**0.2606±0.0014**	**0.3403±0.0035**	0.2692±0.0059
DRew-GatedGCN	0.6733±0.0094	0.2699±0.0018	0.3293±0.0005	**0.3214±0.0021**
DRew-GatedGCN+LapPE	**0.6977±0.0026**	**0.2539±0.0007**	0.3324±0.0014	**0.3314±0.0024**

with respect to $\nu \in \{1, \infty\}$ and network depth L for both the PE and non-PE cases. Hyperparameters and other experimental details are available in Appendix B. As in Dwivedi et al. (2022), **we use a fixed ~500k parameter budget.**

Discussion. As shown in Table 1, νDRew-MPNNs substantially outperform their classical counterparts across all four tasks. We particularly emphasise this result for GINE and GatedGCN, as both models utilise edge features, unlike their DRew counterparts. Furthermore, DRew outperforms the static rewiring and multi-hop benchmarks in all tasks, and at least one DRew-MPNN model matches or beats the best 'classical' graph Transformer baseline from Dwivedi et al. (2022) *in all four tasks*. GraphGPS (Rampášek et al., 2022) outperforms the best DRew model in the PascalVOC-SP and and Peptides-struct tasks, but we stress that GraphGPS is a much more sophisticated architecture that *combines dense Transformers with message passing*, and therefore supports our claim that pure global attention throws away important inductive bias afforded by MPNN approaches. Even so, DRew still surpasses GraphGPS in the Peptides-func and PCQM-Contact tasks.

5.2. QM9

QM9 (Ramakrishnan et al., 2014) is a molecular multi-task graph regression benchmark dataset of ~130,000 graphs with ~18 nodes each and a maximum graph diameter of 10. We compare νDRew-GIN against a number of benchmark MPNNs and a GIN-based multi-hop MPNN: shortest path network (SPN; Abboud et al. (2022)), which is similar to our work in that it uses a multi-hop aggregation based on shortest path distances, but differs crucially in the lack of dynamic, per-layer rewiring or delay. Experimental results for all regression targets are given in Table 3.

Experimental details. Our experimental setup is based on Brockschmidt (2020) and uses the same fixed data splits. We use the overall-best-performing SPN parameterisation with a 'max distance' of $k_{\max} = 10$, referring to the k-hop neighbourhood aggregated over at each layer. This allows every node to interact with every other node at each layer when applied to a small-graph dataset like QM9, amounting to a dense static rewiring. DRew-GIN and SPN models use a **parameter budget** of ~800,000, use 8 layers and train for 300 epochs; results are averaged over three runs. Neither SPN or DRew-GIN use relational edge features (denoted 'R-'; Schlichtkrull et al. (2018); Brockschmidt (2020)) as its impact is minimal (see Appendix B). Other results are reproduced from their respective works (Brockschmidt, 2020; Alon & Yahav, 2021); several of these include a final fully adjacent layer (+FA) which we include rather than the base models as they afford improved performance overall.

Discussion. DRew demonstrates improvement over the

Table 3. Performance of νDRew compared with MPNN benchmarks on `QM9`. Scores reported are test MAE, i.e. lower is better.

Property	R-GIN+FA	R-GAT+FA	R-GatedGNN+FA	GNN-FiLM	SPN	DRew-GIN	ν_1DRew-GIN
mu	2.54±0.09	2.73±0.07	3.53±0.13	2.38±0.13	2.32±0.28	**1.93±0.06**	2.00±0.05
alpha	2.28±0.04	2.32±0.16	2.72±0.12	3.75±0.11	1.77±0.09	**1.63±0.03**	**1.63±0.05**
HOMO	1.26±0.02	1.43±0.02	1.45±0.04	1.22±0.07	1.26±0.09	**1.16±0.01**	1.17±0.02
LUMO	1.34±0.04	1.41±0.03	1.63±0.06	1.30±0.05	1.19±0.05	**1.13±0.02**	1.15±0.02
gap	1.96±0.04	2.08±0.05	2.30±0.05	1.96±0.06	1.89±0.11	**1.74±0.02**	**1.74±0.03**
R2	12.61±0.37	15.76±1.17	14.33±0.47	15.59±1.38	10.66±0.40	**9.39±0.13**	9.94±0.07
ZPVE	5.03±0.36	5.98±0.43	5.24±0.30	11.00±0.74	**2.77±0.17**	2.73±0.19	2.90±0.30
U0	2.21±0.12	2.19±0.25	3.35±1.68	5.43±0.96	1.12±0.13	**1.01±0.09**	1.00±0.07
U	2.32±0.18	2.11±0.10	2.49±0.34	5.95±0.46	**1.03±0.09**	0.99±0.08	0.97±0.04
H	2.26±0.19	2.27±0.29	2.31±0.15	5.59±0.57	**1.05±0.04**	1.06±0.09	1.02±0.09
G	2.04±0.24	2.07±0.07	2.17±0.29	5.17±1.13	**0.97±0.06**	1.06±0.14	1.01±0.05
Cv	1.86±0.03	2.03±0.14	2.25±0.20	3.46±0.21	1.36±0.06	**1.24±0.02**	1.25±0.03
Omega	0.80±0.04	0.73±0.04	0.87±0.09	0.98±0.06	0.57±0.04	**0.55±0.01**	0.60±0.03

classical and multi-hop MPNN benchmarks, beating or matching the next-best model, SPN, for 12 out of 13 regression targets. We note that, overall, the best average performance across targets is achieved by DRew without delay ($\nu = \infty$). This is as we might expect, as L-layer models with 'slow' information flow such as classical GCNs and ν_1DRew cannot guarantee direct interaction between all node pairs on graphs with maximum diameter $> L$.

5.3. Validating robustness

In this section we demonstrate the robustness properties, rather than raw performance, of νDRew with increasing network depth for long-range tasks.

5.3.1. RINGTRANSFER

`RingTransfer` is a synthetic task for empirically validating the ability of a GNN to capture **long-range** node dependencies (Bodnar et al., 2021). The dataset consists of N ring graphs (chordless cycles) of length k. Each graph has a single **source** node and a single **target** node that are always $\lfloor \frac{k}{2} \rfloor$ hops apart. Source node features are one-hot class label vectors of length C; all other nodes features are uniform. The task is for the **target** node to output the correct class label at the source. We compare (ν)DRew-GCN agaist a GCN and SP-GCN, an instance of the SPN framework (Abboud et al., 2022). Results are given in Figure 2 where the number of layers $L = \lfloor \frac{k}{2} \rfloor$ is the minimum required depth for source-target interaction.

Discussion. `RingTransfer` demonstrates the power of νDRew in mitigating MPNN performance issues brought on by increased depth. While the classical GCN fails after fewer than 10 layers, ν_1DRew achieves strong performance for 30 or more. These results also allow us to directly assess the impact of delay. The 'full-delay' ν_1DRew-GCN
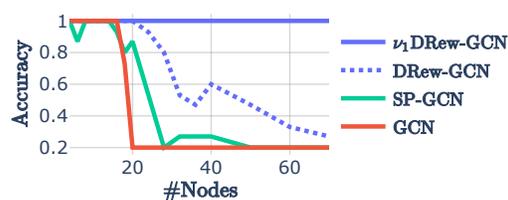
Figure 2. Performance on `RingTransfer` task for models with varying k, L. Accuracy of 0.2 corresponds to a random guess.

consistently achieves perfect accuracy for 30+ layers with no drop in performance. We can attribute this to the direct interaction between the target and delayed source node. SP-GCN, however, with its static rewiring and dense computational graph, improves on the classical GCN, likely due to increased expressivity, but still fails at much shallower L than νDRew, *with or without delay*.

5.3.2. LAYERWISE PERFORMANCE ON PEPTIDES-FUNC

In this section we strip back our experiments on `Peptides-func` to demonstrate the robustness of νDRew for increasing network depth L, as well as the impact of ν, the parameter denoting rate of information flow during message passing. For these experiments we fix the hidden dimension to 64. In Figure 3 we plot model performance against L for three different parameterisations of νDRew-GCN: the non-delay version $\nu = \infty$ (which reduces to DRew), the full-delay version $\nu = 1$, and a midpoint, where we set $\nu = L/2$.

Discussion. Figure 3 demonstrates a crucial contribution of our framework: the ability to tune ν to suit the task. It is evident that using νDRew with delay ensures more robust training when using a deeper architecture; in fact, **the more delay used** (i.e. the lower the value of ν), **the**
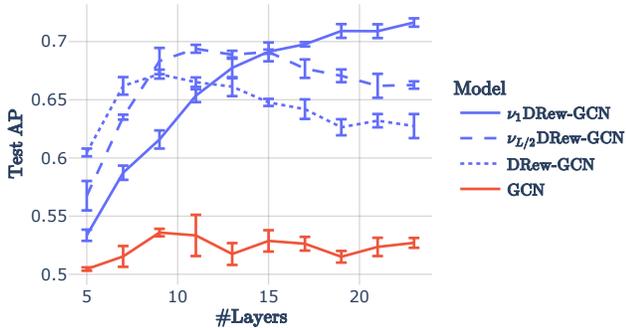
Figure 3. Comparing three parameterizations of νDRew plus a classical residual GCN on `Peptides-func` over varying L.

better the performance for large L, whereas using less delay (high ν) ensures faster filling of the computational graph and greater density of connections after fewer layers. This means that, when using lower L, non-delay DRew often performs better, especially when combined with PE. Conversely, more delay 'slows down' the densification of node connections, yielding stronger long-term performance with L. Figure 3 demonstrates this with a long-range task: ν_1DRew consistently improves with more layers.

6. Related Work

Various methods have been developed to improve learning on graphs and avoid issues such as over-squashing. Many of these are broadly classified as 'graph rewiring' methods; one such family involves sampling of nodes and/or edges of the graph based on some sort of performance or topological metric. Examples include sparsification (Hamilton et al., 2017), node (edge) dropout (Rong et al., 2019; Papp et al., 2021), rewiring based on graph diffusion (Klicpera et al., 2019), Cayley graphs (Deac et al., 2022), commute times, spectral gap or Ricci curvature to combat over-squashing (Arnaiz-Rodríguez et al., 2022; Topping et al., 2022; Black et al., 2023). Most of these methods remove supposedly irrelevant elements from the graph to make it more amenable to analysis, though many methods also add elements to increase connectivity. This might be a global node (Battaglia et al., 2016; Gilmer et al., 2017), or global layer, such as positional/structural encoding (Dwivedi et al., 2020; 2021; Rampášek et al., 2022; Wang et al., 2022), or adding a fully adjacent layer after message passing (Alon & Yahav, 2021). It may also take the form of multiple-hop rewiring, in which aggregations occur over nodes at >1-hop distance at each layer; we distinguish these into 'local' graph rewiring, also known as multi-hop MPNNs (Abboud et al., 2022; Abu-El-Haija et al., 2019; 2020; Nikolentzos et al., 2020; Zhang & Li, 2021), and 'global' methods such as graph Transformers (Dwivedi et al., 2022; Kreuzer et al., 2021; Rampášek et al., 2022; Ying et al., 2021; Yun et al., 2019), which fully connect the input graph.

Unlike all of these methods, the rewiring used in DRew is layer-dependent, i.e. it is adaptive rather than static. Our method is also unique in its ability to control the rate of information flow by tuning the delay parameter ν. Our use of delay is loosely inspired by delay differential equations (DDEs), which have also inspired architectures which leverage delay in the wider deep learning space (Anumasa & PK, 2021; Zhu et al., 2021; 2022) based on neural ordinary differential equations (Chen et al., 2018), but we know of no DDE-inspired works in the graph machine learning space. The idea of accessing previous node states resonates with Xu et al. (2018) and Strathmann et al. (2021). Faber & Wattenhofer (2022) use a form of delay to create node identifiers as a means to allow nodes to ignore irrelevant messages from their neighbours, but all of these works bear little relation to DRew, which treats dynamic rewiring and delay from the perspective of distance on graphs.

7. Conclusion and Future Work

We have introduced an adaptive MPNN framework based on a layer-dependent dynamic rewiring that can be adapted to any MPNN. We have also proposed a delay mechanism permitting local skip-connections among different nodes based on their mutual distance. Investigating the expressive power of this framework represents a promising future avenue to compare static and dynamic rewiring approaches, as well as the impact of distance-aware skip-connections.

Limitations. Our framework is expected to be useful for tasks with long-range interactions or, more generally, when one requires very deep GNN models, as confirmed by our experiments. Accordingly, we do not expect our framework to be advantageous when applied to (for example) homophilic node classification tasks where shallow GNNs acting as low-pass filters are sufficient to perform strongly. This may partly explain why DRew is outperformed by GraphGPS for `PascalVOC-SP` (see Table 2), as this dataset presents an image segmentation task which likely displays a reasonable degree of homophily. As a result, the extent to which long-range interactions are truly present is uncertain.

References

Abboud, R., Dimitrov, R., and Ceylan, İ. İ. Shortest path networks for graph property prediction. *arXiv preprint arXiv:2206.01003*, 2022.

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Ver Steeg, G., and Galstyan, A. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pp. 21–29. PMLR, 2019.

Abu-El-Haija, S., Kapoor, A., Perozzi, B., and Lee, J. N-GCN: Multi-scale graph convolution for semi-supervised node classification. In *uncertainty in artificial intelligence*, pp. 841–851. PMLR, 2020.

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=i80OPhOCVH2.

Anumasa, S. and PK, S. Delay differential neural networks. In *2021 6th International Conference on Machine Learning Technologies*, pp. 117–121, 2021.

Arnaiz-Rodríguez, A., Begga, A., Escolano, F., and Oliver, N. Diffwire: Inductive graph rewiring via the Lovász bound. *arXiv preprint arXiv:2206.07369*, 2022.

Barceló, P., Kostylev, E. V., Monet, M., Pérez, J., Reutter, J., and Silva, J. P. The logical expressiveness of graph neural networks. In *ICLR*, 2019.

Battaglia, P., Pascanu, R., Lai, M., Jimenez Rezende, D., et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

Black, M., Nayyeri, A., Wan, Z., and Wang, Y. Understanding oversquashing in gnns through the lens of effective resistance. *arXiv preprint arXiv:2302.06835*, 2023.

Bodnar, C., Frasca, F., Otter, N., Wang, Y., Lio, P., Montufar, G. F., and Bronstein, M. Weisfeiler and Lehman go cellular: CW networks. *Advances in Neural Information Processing Systems*, 34:2625–2640, 2021.

Bodnar, C., Di Giovanni, F., Chamberlain, B. P., Liò, P., and Bronstein, M. M. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in GNNs. *arXiv preprint arXiv:2202.04579*, 2022.

Borgwardt, K. M. and Kriegel, H.-P. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*, pp. 8–pp. IEEE, 2005.

Bresson, X. and Laurent, T. Residual gated graph ConvNets. *arXiv preprint arXiv:1711.07553*, 2017.

Brockschmidt, M. GNN-FiLM: Graph neural networks with feature-wise linear modulation. In *International Conference on Machine Learning*, pp. 1144–1152. PMLR, 2020.

Brüel-Gabrielsson, R., Yurochkin, M., and Solomon, J. Rewiring with positional encodings for graph neural networks. *arXiv preprint arXiv:2201.12674*, 2022.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

Cai, C. and Wang, Y. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Deac, A., Lackenby, M., and Veličković, P. Expander graph propagation. *arXiv preprint arXiv:2210.02997*, 2022.

Di Giovanni, F., Rowbottom, J., Chamberlain, B. P., Markovich, T., and Bronstein, M. M. Graph neural networks as gradient flows. *arXiv preprint arXiv:2206.10991*, 2022.

Di Giovanni, F., Giusti, L., Barbero, F., Luise, G., Lio, P., and Bronstein, M. On over-squashing in message passing neural networks: The impact of width, depth, and topology. *arXiv preprint arXiv:2302.02941*, 2023.

Dwivedi, V. P. and Bresson, X. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021.

Dwivedi, V. P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. Long range graph benchmark. *arXiv preprint arXiv:2206.08164*, 2022.

Faber, L. and Wattenhofer, R. Asynchronous neural networks for learning in graphs. *arXiv preprint arXiv:2205.12245*, 2022.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272. PMLR, 2017.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proc. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734. IEEE, 2005.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 1025–1035, 2017.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., and Leskovec, J. OGB-LSC: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *Proc. of Int. Conference on Learning Representations (ICLR), San Diego, CA, USA, May 2015.*

Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Klicpera, J., Weissenberger, S., and Günnemann, S. Diffusion improves graph learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. Rethinking graph Transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Nikolentzos, G., Dasoulas, G., and Vazirgiannis, M. k-hop graph neural networks. *Neural Networks*, 130:195–205, 2020.

Nt, H. and Maehara, T. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

Papp, P. A., Martinkus, K., Faber, L., and Wattenhofer, R. DropGNN: Random dropouts increase the expressiveness of graph neural networks. *Advances in Neural Information Processing Systems*, 34:21997–22009, 2021.

Ramakrishnan, R., Dral, P., Rupp, M., and Von Lilienfeld, O. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a general, powerful, scalable graph transformer. *arXiv preprint arXiv:2205.12454*, 2022.

Richards, A. University of Oxford Advanced Research Computing. http://dx.doi.org/10.5281/zenodo.22558, 2015.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

Rusch, T. K., Chamberlain, B. P., Mahoney, M. W., Bronstein, M. M., and Mishra, S. Gradient gating for deep multi-rate learning on graphs. *arXiv preprint arXiv:2210.00513*, 2022.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Berg, R. v. d., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

Sperduti, A. Encoding labeled graphs by labeling RAAM. *Advances in Neural Information Processing Systems*, 6, 1993.

Strathmann, H., Barekatain, M., Blundell, C., and Veličković, P. Persistent message passing. *arXiv preprint arXiv:2103.01043*, 2021.

Topping, J., Di Giovanni, F., Chamberlain, B. P., Dong, X., and Bronstein, M. M. Understanding over-squashing and bottlenecks on graphs via curvature. *International Conference on Learning Representations*, 2022.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Wang, H., Yin, H., Zhang, M., and Li, P. Equivariant and stable positional encoding for more powerful graph neural networks. *arXiv preprint arXiv:2203.00199*, 2022.

Weisfeiler, B. and Leman, A. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16, 1968.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462. PMLR, 2018.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do Transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

Yun, S., Jeong, M., Kim, R., Kang, J., and Kim, H. J. Graph transformer networks, 2019. Conference on Neural Information Processing Systems (NeurIPS), 2019.

Zhang, M. and Li, P. Nested graph neural networks. *Advances in Neural Information Processing Systems*, 34: 15734–15747, 2021.

Zhu, Q., Guo, Y., and Lin, W. Neural delay differential equations. *arXiv preprint arXiv:2102.10801*, 2021.

Zhu, Q., Shen, Y., Li, D., and Lin, W. Neural piecewise-constant delay differential equations. *arXiv preprint arXiv:2201.00960*, 2022.

A. Time and Space Complexity

Time complexity. DRew relies on the shortest path and therefore requires up-to k-hop adjacency information for layer k. This can be pre-computed in worst-case time $\mathcal{O}(|V||E|)$ using the same breadth-first search method of Abboud et al. (2022), but once computed it can be re-used for all runs.

In the worst case — when ℓ is greater than the max diameter of the graph — DRew performs aggregation over $\mathcal{O}(V^2)$ elements, i.e. all node pairs. However, each k-neighbourhood aggregation can be computed in parallel, so this is not a serious concern in practice, and as DRew gradually builds the computational graph at each layer, it is faster than comparable multi-hop MPNNs or Transformers that aggregate over the entire graph, or a fixed hop distance, at every layer.

Space complexity. As we use delayed representations of nodes, we must store ℓ representations of the node features at layer ℓ for a given mini-batch; i.e. linear scaling in network depth L. This has not proved to be a bottleneck in practice, and could be addressed if need be by reducing batch size.

B. Further Experimental Details

In this section we provide further details about our experiments, as well as further results on `Peptides-func`.

B.1. Hardware

All experiments were run on server nodes using a single GPU. A mixture of P100, V100, A100, Titan V and RTX GPUs were used, as well as a mixture of Broadwell, Haswell and Cacade Lake CPUs.

B.2. Long range graph benchmark

All results in Tables 1 and 2 use similar experimental setups and identical data train–val–test splits to their classical MPNN counterparts in Dwivedi et al. (2022). We use a fixed parameter budget of ~500,000, which is controlled for by appropriate tuning of hidden dimension when using different network depths L. Significant hyperparameter differences for experiments are given in Table 4; other experimental details are itemised below:

- For all experiments we train for 300 epochs or until convergence and average results over three seeds.

- All results use the AdamW optimizer (Kingma & Ba; Loshchilov & Hutter, 2017) with base learning rate `lr=0.001` (except `PascalVOC-SP`, which uses `0.0005`), `lr_decay=0.1`, `min_lr=1e-5`, `momentum=0.9`, and a reduce-on-plateau scheduler with `reduce_factor=0.5`, `schedule_patience=10` (`20` for `Peptides`).

- All `Peptides`, `PCQM-Contact` and `PascalVOC-SP` experiments use batch sizes of 128, 256 and 32 respectively.

- All experiments use batch normalisation with `eps=1e-5`, `momentum=0.1` and post-layer L_2 normalisation.

- `Peptides` and `PCQM-Contact` use the 'Atom' node encoder (Hu et al., 2020; 2021), whereas `PascalVOC-SP` uses a node encoder which is a single linear layer with a fixed output dimension of 14.

- None of the experiments in Table 2 use edge encoding or edge features.

- Superpixel nodes in `PascalVOC-SP` are extracted using a SLIC compactness of 30 for the SLIC algorithm (Achanta et al., 2012).

- We do not use dropout.

- All Laplacian PE uses hidden dimension of 16 and 2 layers.

- For `PCQM-Contact`, all experiments (except for DIGL) use convex combination with equal weights for aggregation; i.e. each of M channel-mixing matrices per k-neighbourhood aggregation is multiplied by $1/M$. Other tasks do not use any matrix weights.

- Experiments for MixHop-GCN, a a multi-hop MPNN, are parameterised by $\max P$, where integer powers of the adjacency matrix are aggregated up to $\max P$, with equal-size weight matrices per adjacency power. The hyperparameters in Table 4 were determiend by best performance after hyperparameter search over $\max P$ and network depth L.

- We perform DIGL rewiring using the default settings from the Graph Diffusion Convolution transform from `torch_geometric.transforms` using PPR diffusion with $\alpha = 0.2$ and threshold sparsification with average degree d_{avg} given in Table 4.

- All DIGL+MPNN runs use GCN as the base MPNN except for `PascalVOC-SP` which uses GatedGCN instead for fair comparison, as other classical MPNNs perform poorly on this task

- `Peptides-func` results in Figures 3, 4 and 5 use the same experimental setup as described above.

- The reported results for GatedGCN+PE in Table 2 use LapPE for `PascalVOC-SP`and RWSE for all other tasks.

Table 4. Parameter counts (#Ps) and significant hyperparameters (HPs) for for all DIGL, MixHop-GCN and (ν)DRew-MPNN experiments in Table 2. Hyperparameterisation details for reproduced results are available in their respective works.

Model	Peptides-func #Ps	HPs	Peptides-struct #Ps	HPs	PCQM-Contact #Ps	HPs	PascalVOC-SP #Ps	HPs
DIGL+MPNN	499k	$d_{\text{avg}} = 6$ $L = 13$	496k	$d_{\text{avg}} = 6$ $L = 5$	497k	$d_{\text{avg}} = 2$ $L = 5$	502k	$d_{\text{avg}} = 14$ $L = 8$
DIGL+MPNLapPE	493k	$d_{\text{avg}} = 6$ $L = 5$	496k	$d_{\text{avg}} = 6$ $L = 7$	495k	$d_{\text{avg}} = 2$ $L = 5$	502k	$d_{\text{avg}} = 14$ $L = 8$
MixHop-GCN	513k	$\max P = 5$ $L = 17$	510k	$\max P = 7$ $L = 17$	523k	$\max P = 3$ $L = 5$	511k	$\max P = 5$ $L = 8$
MixHop-GCN+LapPE	518k	$\max P = 7$ $L = 14$	490k	$\max P = 7$ $L = 11$	521k	$\max P = 3$ $L = 5$	512k	$\max P = 5$ $L = 8$
DRew-GCN	518k	$\nu = 1$ $L = 23$	498k	$\nu = \infty$ $L = 13$	515k	$\nu = \infty$ $L = 20$	498k	$\nu = 1$ $L = 8$
DRew-GCN+LapPE	502k	$\nu = \infty$ $L = 7$	495k	$\nu = \infty$ $L = 5$	498k	$\nu = \infty$ $L = 10$	498k	$\nu = 1$ $L = 8$
DRew-GIN	514k	$\nu = 1$ $L = 17$	505k	$\nu = \infty$ $L = 15$	507k	$\nu = \infty$ $L = 20$	506k	$\nu = 1$ $L = 8$
DRew-GIN+LapPE	502k	$\nu = 1$ $L = 15$	497k	$\nu = \infty$ $L = 5$	506k	$\nu = \infty$ $L = 10$	506k	$\nu = 1$ $L = 8$
DRew-GatedGCN	495k	$\nu = 1$ $L = 17$	497k	$\nu = \infty$ $L = 13$	506k	$\nu = \infty$ $L = 20$	502k	$\nu = 1$ $L = 8$
DRew-GatedGCN+LapPE	495k	$\nu = \infty$ $L = 7$	494k	$\nu = \infty$ $L = 5$	494k	$\nu = \infty$ $L = 10$	502k	$\nu = 1$ $L = 8$

B.3. QM9

Performance experiments use a fixed parameter budget of 800k, controlled for by appropriate tuning of the hidden dimension when using different network depth L. We mostly follow the experimental setup of (Abboud et al., 2022), using a fixed learning rate of 0.001, mean pooling and no dropout. We use batch normalization, train for 300 epochs averaging over 3 runs, and use data splits from (Brockschmidt, 2020).

Many of the benchmarks we compare against in Table 3 are Relational MPNNs (R-MPNN; Schlichtkrull et al. (2018); Brockschmidt (2020)) which incorporate edge labels by assigning separate weights for each edge type in the 1-hop neighbourhood, and aggregating over each type separately. For our SPN and DRew-GIN experiments, however, we do not incorporate edge features, as (a) DRew demonstrates strong performance even without this information, and (b) we expect the improvement obtained through using R-GIN to be slight given that over 92% of all graph edges in QM9 are of only one type.

B.4. `RingTransfer`

For synthetic `RingTransfer` (Bodnar et al., 2021) experiments we use a dataset of size $N = 2000$ with $C = 5$ classes and a corresponding node feature dimension. GCN and SP-GCN runs use a hidden dimension of 256, and for fair comparison DRew-GCN runs use a varying hidden dimension to ensure the same parameter count as GCN/SP-GCN for each ring length k (and therefore model depth L). All runs use batch normalization, no dropout, and Adam optimization with learning rate 0.01. We train for 50 epochs and average over three experiments, using the accuracy of predicting the source node label from the readout of the source node representation as a metric. We use an 80:10:10 split for train, test and validation.

SP-GCN   We define SP-GCN as an instance of the SP-MPNN framework (Abboud et al., 2022):

$$h_i^{(\ell+1)} = h_i^{(\ell)} + \sigma \left(\sum_{k=1}^{k_{\max}} \sum_{j \in \mathcal{N}_k} \alpha_k^{(\ell)} \mathbf{W}^{(\ell)} \gamma_{ij}^k h_j^{(\ell)} \right), \tag{11}$$

where $k_{\max}$ is the max distance parameter that determines the number of hops to aggregate over at each layer and $\boldsymbol{\alpha}^{(\ell)} \subset \mathbb{R}^{k_{\max}}$ are learned weight vectors, $\sum_k^{k_{\max}} \alpha_k^{(\ell)} = 1$. γ_{ij}^k is as in Eq. (8).

B.5. Ablation on `Peptides-func`

In this section we provide additional experimental results on `Peptides-func` using our 500k parameter budget setup from Section 5.1. We train a number of varying-depth GCN, residual GCN and νDRew-GCN models with three parameterizations of ν: 1, $L/2$ and ∞. We provide separate results with and without Laplacian PE (Dwivedi et al., 2020) in Figures 5 and 4 respectively. For reference, on both figures we also mark the best-performing Transformer, SAN with random walk encoding (Kreuzer et al., 2021; Dwivedi et al., 2021), reproduced from Dwivedi et al. (2022) and denoted with a dashed black line.

Discussion   From Figures 4 and 5 we can see that more delay leads to stronger performance at greater network depths, even as the hidden dimension decreases severely. We see that low L, low/no delay νDRew and high L, high delay νDRew outperform GCNs and the best-performing Transformer, *with or without positional encoding*.

We note that the poor performance of low-delay νDRew at high L and vice-versa is as expected. As `Peptides-func` is a long-range task, strong performance requires interaction between distant nodes. Though it uses more powerful multi-hop aggregations, ν_1DRew maintains the same 'rate' of information flow as classical MPNNs, i.e. r-distant nodes are able to interact from the rth layer; therefore small L does not give ν_1DRew sufficient layers to enable long-range interactions, and performance is comparable to classical MPNNs. As our framework is more robust, however, ν_1DRew continues to increase in performance as L increases — as we would hope — while the classical MPNNs GCN and ResGCN succumb to the usual issues that affect MPNNs, and degrade.

Conversely, the low delay parameterizations, $\{\nu_{L/2}, \nu_\infty\}$DRew-GCN, perform strongly for low L and worsen as the network becomes very deep. Again, this is expected: low delay affords fast (or instantaneous) communication between distant nodes after sufficient layers, and therefore has a rate of information flow that is faster than ν_1DRew or classical MPNNs (though still slower than multi-hop MPNNs or Transformers). This means that, for a long-range task such as `Peptides-func`, performance is stronger for fewer layers, once long-range interactions have been facilitated but *before* the computational graph becomes too dense, causing performance drop. These experiments further demonstrate the impact and usefulness of the delay parameter as a means of tuning the model to suit the task.

Referring to Figure 5, we note that the addition of PE exacerbates the behaviours described above, and indeed accelerates the rate of information flow by preceding the message-passing with a global layer.

Positional encoding.   As a final point, we consider the overall impact of PE, particularly Laplacian PE, on this task. We posit that, for `Peptides`, the characterization of Transformers as the strongest long-range models (Dwivedi et al., 2022) is due primarily to PE, rather than the Transformer architecture. As evidence we point to the performance of vanilla GCN, the simplest MPNN, on `func` when LapPE is used; it outperforms SAN with only five layers. We reiterate that νDRew outperforms MPNN and Transformer+PE benchmarks with or without using PE itself.
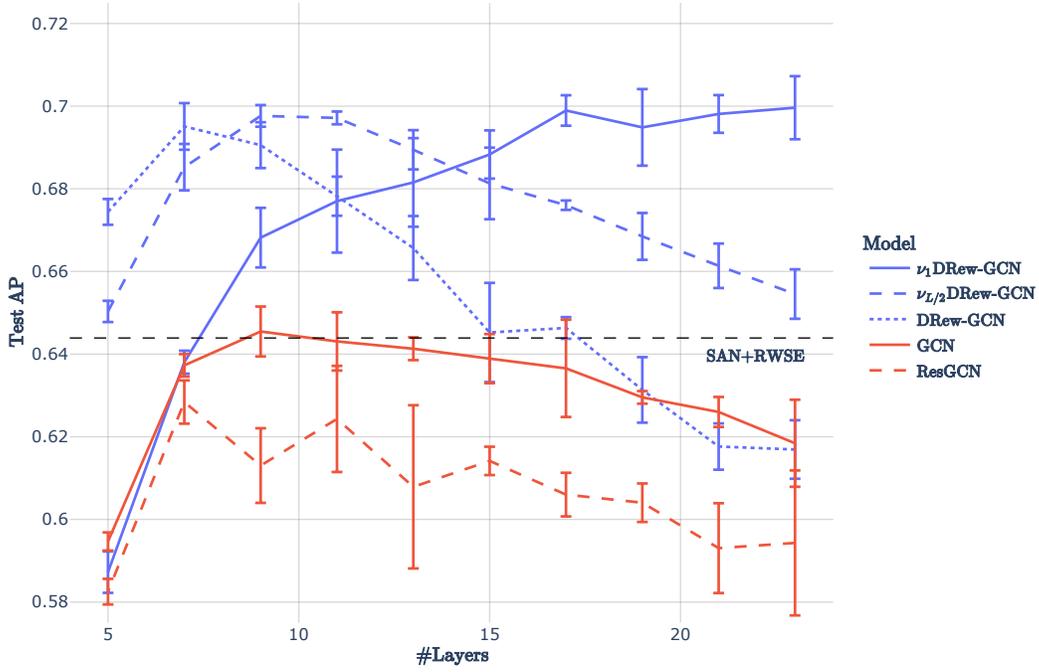
Figure 4. `Peptides-func` experiments over varying L with fixed 500k parameter count using **no** positional encoding.
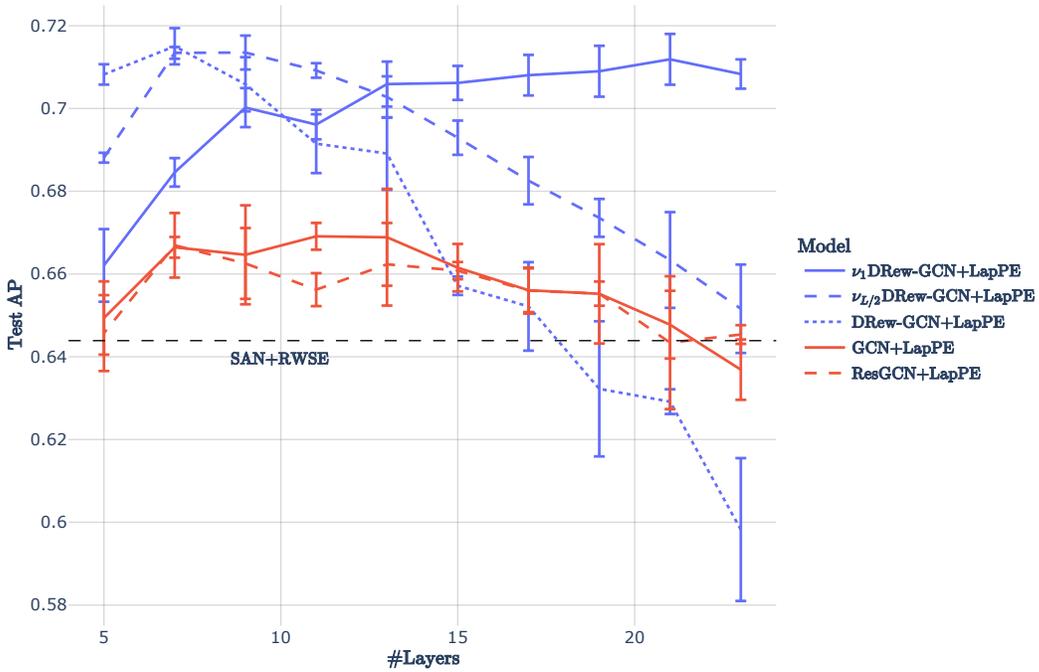
Figure 5. `Peptides-func` experiments over varying L with fixed 500k parameter count using **Laplacian** positional encoding.