ANTI-SYMMETRIC DGN: A STABLE ARCHITECTURE FOR DEEP GRAPH NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep Graph Networks (DGNs) currently dominate the research landscape of learning from graphs, due to their efficiency and ability to implement an adaptive message-passing scheme between the nodes. However, DGNs are typically limited in their ability to propagate and preserve long-term dependencies between nodes, i.e., they suffer from the over-squashing phenomena. This reduces their effectiveness, since predictive problems may require to capture interactions at different, and possibly large, radii in order to be effectively solved. In this work, we present Anti-Symmetric Deep Graph Networks (A-DGNs), a framework for stable and non-dissipative DGN design, conceived through the lens of ordinary differential equations. We give theoretical proof that our method is stable and non-dissipative, leading to two key results: long-range information between nodes is preserved, and no gradient vanishing or explosion occurs in training. We empirically validate the proposed approach on several graph benchmarks, showing that A-DGN leads to improved performance and enables to learn effectively even when dozens of layers are used.

1 INTRODUCTION

Representation learning for graphs has become one of the most prominent fields in machine learning. Such popularity derives from the ubiquitousness of graphs. Indeed, graphs are an extremely powerful tool to represent systems of relations and interactions and are extensively employed in many domains (Battaglia et al., 2016; Gilmer et al., 2017; Zitnik et al., 2018; Monti et al., 2019; Derrow-Pinion et al., 2021). For example, they can model social networks, molecular structures, protein-protein interaction networks, recommender systems, and traffic networks.

The primary challenge in this field is how we capture and encode structural information in the learning model. Common methods used in representation learning for graphs usually employ *Deep Graph Networks* (DGNs) (Bacciu et al., 2020; Wu et al., 2021). DGNs are a family of learning models that learn a mapping function that compresses the complex relational information encoded in a graph into an information-rich feature vector that reflects both the topological and the label information in the original graph. As widely popular with neural networks, also DGNs consists of multiple layers. Each of them updates the node representations by aggregating previous node states and their neighbors, following a message passing paradigm. However, in some problems, the exploitation of local interactions between nodes is not enough to learn representative embeddings. In this scenario, it is often the case that the DGN needs to capture information concerning interactions between nodes that are far away in the graph, i.e., by stacking multiple layers. A specific predictive problem typically needs to consider a specific range of node interactions in order to be effectively solved, hence requiring a specific number (possibly large) of DGN layers.

Despite the progress made in recent years in the field, many of the proposed methods suffer from the *over-squashing* problem (Alon & Yahav, 2021) when the number of layers increases. Specifically, when increasing the number of layers to cater for longer-range interactions, one observes an excessive amplification or an annihilation of the information being routed to the node by the message passing process to update its fixed length encoding. As such, over-squashing prevents DGNs to learn long-range information.

In this work, we present *Anti-Symmetric Deep Graph Network* (A-DGN), a framework for effective long-term propagation of information in DGN architectures designed through the lens of ordinary

differential equations (ODEs). Leveraging the connections between ODEs and deep neural architectures, we provide theoretical conditions for realizing a *stable* and *non-dissipative* ODE system on graphs through the use of anti-symmetric weight matrices. The formulation of the A-DGN layer then results from the forward Euler discretization of the achieved graph ODE. Thanks to the properties enforced on the ODE, our framework preserves the long-term dependencies between nodes as well as prevents from gradient explosion or vanishing. Interestingly, our analysis also paves the way for rethinking the formulation of standard DGNs as discrete versions of non-dissipative and stable ODEs on graphs.

The key contributions of this work can be summarized as follows:

- We introduce A-DGN, a novel design scheme for deep graph networks stemming from an ODE formulation. Stability and non-dissipation are the main properties that characterize our method, allowing the preservation of long-term dependencies in the information flow.
- We theoretically prove that the employed ODE on graphs has stable and non-dissipative behavior. Such result leads to the absence of exploding and vanishing gradient problems during training, typical of unstable and lossy systems.
- We conduct extensive experiments to demonstrate the benefits of our method. A-DGN can outperform classical DGNs over several datasets even when dozens of layers are used.

The rest of this paper is organized as follows. We introduce the A-DGN framework in Section 2 by theoretically proving its properties. In Section 3, we give an overview of the related work in the field of representation learning for graphs and continuous dynamic models. Afterwards, we provide the experimental assessment of our method in Section 4. Finally, Section 5 concludes the paper.

2 Anti-Symmetric Deep Graph Network

Recent advancements in the field of representation learning propose to treat neural network architectures as an ensemble of continuous (rather than discrete) layers, thereby drawing connections between deep neural networks and ordinary differential equations (ODEs) (Haber & Ruthotto, 2017; Chen et al., 2018). This connection can be pushed up to neural processing of graphs as introduced in (Poli et al., 2019), by making a suitable ODE define the computation on a graph structure.

We focus on static graphs, i.e., on structures described by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ and $\mathcal{E}$ respectively denoting the fixed sets of nodes and edges. For each node $u \in \mathcal{V}$ we consider a state $\mathbf{x}_u(t) \in \mathbb{R}^d$, which provides a representation of the node u at time t. We can then define a Cauchy problem on graphs in terms of the following node-wise defined ODE:

$$\frac{\partial \mathbf{x}_u(t)}{\partial t} = f_{\mathcal{G}}(\mathbf{x}_u(t)), \tag{1}$$

for time $t \in [0, T]$, and subject to the initial condition $\mathbf{x}_u(0) = \mathbf{x}_u^0 \in \mathbb{R}^d$. The dynamics of node's representations is described by the function $f_{\mathcal{G}} : \mathbb{R}^d \to \mathbb{R}^d$, while the initial condition $\mathbf{x}_u(0)$ can be interpreted as the initial configuration of the node's information, hence as the input for our computational model. As a consequence, the ODE defined in Equation 1 can be seen as a continuous information processing system over the graph, which starting from the input configuration $\mathbf{x}_u(0)$ computes the final node's representation (i.e., embedding) $\mathbf{x}_u(T)$. Notice that this process shares similarities with standard DGNs, in what it computes nodes' states that can be used as an embedded representation of the graph and then used to feed a readout layer in a downstream task on graphs. The top of Figure 1 visually summarizes this concept, showing how nodes evolve following a specific graph ODE in the time span between 0 and a terminal time $T > 0$.

Since for most ODEs it is impractical to compute an analytical solution, a common approach relies on finding an approximate one through a numerical discretization procedure (such as the forward Euler method). In this way, the time variable is discretized and the ODE solution is computed by the successive application of an iterated map that operates on the discrete set of points between 0 and T, with a step size $\epsilon > 0$. Crucially, as already observed for feed-forward and recurrent neural models (Haber & Ruthotto, 2017; Chang et al., 2019), each step of the ODE discretization process can be equated to one layer of a DGN network. The whole neural architecture contains as many layers as the integration steps in the numerical method (i.e., $L = T/\epsilon$), and each layer $\ell = 1, ..., L$ computes

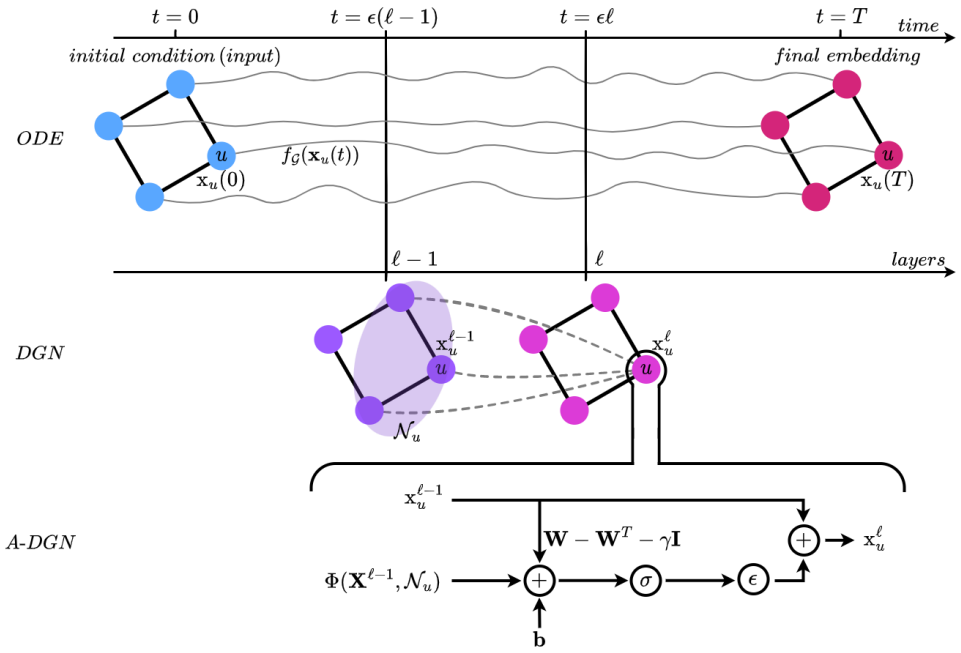

Figure 1: A high level overview of our proposed framework, summarizing the involved concepts of ODE over a graph, its discretization as layers of a DGN, and the resulting node update of Anti-symmetric DGN. At the top, it is illustrated the continuous processing of nodes' states in the time span between 0 and $T > 0$, as a Cauchy problem on graphs. The node-wise ODE $f_{\mathcal{G}}$ determines the evolution of the states $\mathbf{x}_u(t)$, while the initial conditions $\mathbf{x}_u(0)$ play the role of input information. In the middle, the discretized solution of the graph ODE is interpreted as a succession of DGN layers in a neural network architecture. The node state $\mathbf{x}_u^\ell$ computed at layer ℓ is updated iteratively by leveraging its neighborhood and self representations at the previous layer $\ell - 1$. The bottom part sketches the computation performed by a layer in an Anti-Symmetric DGN (in Equation 5), resulting from the forward Euler discretization of a stable and non-dissipative ODE on graphs. The more the layers, the more long-range dependencies are included in the final nodes' representations.

nodes' states $\mathbf{x}_u^\ell$ which approximates $\mathbf{x}_u(\epsilon\,\ell)$. This process is summarized visually in the middle of Figure 1.

Leveraging the concept of graph neural ODEs (Poli et al., 2019), in this paper we perform a further step by reformulating a DGN as a solution to a *stable and non-dissipative* Cauchy problem over a graph. The main goal of our work is therefore achieving preservation of long-range information between nodes, while laying down the conditions that prevent gradient vanishing or explosion. Inspired by the works on stable deep architectures that discretize ODE solutions (Haber & Ruthotto, 2017; Chang et al., 2019), we do so by first deriving conditions under which the graph ODE is constrained to the desired stable and non-dissipative behavior.

Since we are dealing with static graphs, we instantiate Equation 1 for a node u as follows:

$$\frac{\partial \mathbf{x}_u(t)}{\partial t} = \sigma\left(\mathbf{W}_t \mathbf{x}_u(t) + \Phi(\mathbf{X}(t), \mathcal{N}_u) + \mathbf{b}_t\right), \tag{2}$$

where σ is a monotonically non-decreasing activation function, $\mathbf{W}_t \in \mathbb{R}^{d\times d}$ and $\mathbf{b}_t \in \mathbb{R}^d$ are, respectively, a weight matrix and a bias vector that contain the trainable parameters of the system. We denote by $\Phi(\mathbf{X}(t), \mathcal{N}_u)$ the aggregation function for the states of the nodes in the neighborhood of u. We refer to $\mathbf{X}(t) \in \mathbb{R}^{|\mathcal{V}|\times d}$ as the node feature matrix of the whole graph, with d the number of available features. For simplicity, in the following we keep $\mathbf{W}_t$ and $\mathbf{b}_t$ constant over time, hence dropping the t subscript in the notation.

Well-posedness and stability are essential concepts when designing DGNs as solutions to Cauchy problems, both relying on the continuous dependence of the solution from initial conditions. An

ill-posed unstable system, even if potentially yielding a low training error, is likely to lead to a poor generalization error on perturbed data. On the other hand, the solution of a Cauchy problem is stable if the long-term behavior of the system does not depend significantly on the initial conditions (Ascher et al., 1995). In our case, where the ODE defines a message passing diffusion over a graph, our intuition is that a stable encoding system will be robust to perturbations in the input nodes information. Hence, the state representations will change smoothly with the input, resulting in a non-exploding forward propagation and better generalization. This intuition is formalized by the following definition.

Definition 1. *A solution $\mathbf{x}_u(t)$ of the ODE in Equation 2, with initial condition $\mathbf{x}_u(0)$, is stable if for any $\omega > 0$, there exists a $\delta > 0$ such that any other solution $\tilde{\mathbf{x}}_u(t)$ of the ODE with initial condition $\tilde{\mathbf{x}}_u(0)$ satisfying $|\mathbf{x}_u(0) - \tilde{\mathbf{x}}_u(0)| \leq \delta$ also satisfies $|\mathbf{x}_u(t) - \tilde{\mathbf{x}}_u(t)| \leq \omega$, for all $t \geq 0$.*

This means that a small perturbation of size δ of the initial state (i.e., the node input features) results in a perturbation on the subsequent states that is at most ω. As known from the stability theory of autonomous systems (Ascher et al., 1995), this condition is met when the maximum real part of the Jacobian's eigenvalues of $f_{\mathcal{G}}$ is smaller or equal than 0, i.e., $\max_{i=1,\dots,d} Re(\lambda_i(\mathbf{J}(t))) \leq 0, \forall t \geq 0$.

Although stability is a necessary condition for successful learning, it alone is not sufficient to capture long-term dependencies. As it is discussed in Haber & Ruthotto (2017), if $\max_{i=1,\dots,d} Re(\lambda_i(\mathbf{J}(t))) \ll 0$ the result is a lossy system subject to catastrophic forgetting during propagation. Thus, in the graph domain, this means that only local neighborhood information is preserved by the system, while long-range dependencies among nodes are forgotten. If no long-range information is preserved, then it is likely that the DGN will underperform, since it will not be able to reach the minimum radius of inter-nodes interactions needed to effectively solve the task.

Therefore, we can design an ODE for graphs which is stable and non-dissipative that leads to well-posed learning, when the criterion that guarantees stability is met and the Jacobian's eigenvalues of $f_{\mathcal{G}}$ are nearly zero. Under this condition, the forward propagation produces at most moderate amplification or shrinking of the input, which enables to preserve long-term dependencies in the node states. During training, the backward propagation needed to compute the gradient of the loss $\partial\mathcal{L}/\partial\mathbf{x}_u(t)$ will have the same properties of the forward propagation. As such, no gradient vanish nor explosion is expected to occur. More formally:

Proposition 1. *The forward and backward propagations of the ODE in Equation 2 are stable and non-dissipative if*

$$Re(\lambda_i(\mathbf{J}(t))) = 0, \quad \forall i = 1, ..., d. \tag{3}$$

see the proof in Appendix A.

A simple way to impose the condition in Equation 3 is to use an anti-symmetric[1] weight matrix in Equation 2. Under this assumption, we can rewrite Equation 2 as follows:

$$\frac{\partial \mathbf{x}_u(t)}{\partial t} = \sigma\left((\mathbf{W} - \mathbf{W}^T)\mathbf{x}_u(t) + \Phi(\mathbf{X}(t), \mathcal{N}_u) + \mathbf{b}\right) \tag{4}$$

where $(\mathbf{W} - \mathbf{W}^T) \in \mathbb{R}^{d\times d}$ is the anti-symmetric weight matrix. As granted by the following Proposition 2, the Jacobian of the resulting ODE has imaginary eigenvalues, hence it is stable and non-dissipative according to Proposition 1.

Proposition 2. *The Jacobian matrix of the ODE in Equation 4 has purely imaginary eigenvalues, i.e.*

$$Re(\lambda_i(\mathbf{J}(t))) = 0, \forall i = 1, ..., d.$$

Therefore the ODE in Equation 4 is stable and non-dissipative.

See proof in Appendix B.

We now proceed to discretize the ODE in Equation 4 by means of the *forward Euler's method*. To preserve stability of the discretized system[2], we add a diffusion term to Equation 4, yielding the following node state update equation:

$$\mathbf{x}_u^{\ell} = \mathbf{x}_u^{\ell-1} + \epsilon\sigma\left((\mathbf{W} - \mathbf{W}^T - \gamma\mathbf{I})\mathbf{x}_u^{\ell-1} + \Phi(\mathbf{X}^{\ell-1}, \mathcal{N}_u) + \mathbf{b}\right) \tag{5}$$

[1] A matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$ is anti-symmetric (i.e., skew-symmetric) if $\mathbf{A}^T = -\mathbf{A}$.

[2] The interested reader is referred to (Ascher & Petzold, 1998) for an in-depth analysis on the stability of the forward Euler method.

where $\mathbf{I}$ is the identity matrix, γ is a hyper-parameter that regulates the strength of the diffusion, and ϵ is the discretization step. By building on the relationship between the discretization and the DGN layers, we have introduced $\mathbf{x}_u^\ell$ as the state of node u at layer ℓ, i.e. the discretization of state at time $t = \epsilon\ell$.

Now, both ODE and its Euler discretization are stable and non-dissipative. We refer to the framework defined by Equation 5 as *Anti-symmetric Deep Graph Network* (A-DGN), whose state update process is schematically illustrated in the bottom of Figure 1. Notice that having assumed the parameters of the ODE constant in time, A-DGN can also be interpreted as a recursive DGN with weight sharing between layers.

We recall that $\Phi(\mathbf{X}^{\ell-1}, \mathcal{N}_u)$ can be any function that aggregates nodes (and edges) information. Therefore, the general formulation of $\Phi(\mathbf{X}^{\ell-1}, \mathcal{N}_u)$ in A-DGN allows casting all standard DGNs through in their non-dissipative, stable and well-posed version. As a result, A-DGN can be implemented leveraging the aggregation function that is more adequate for the specific task, while allowing to preserve long-range relationships in the graph. As a demonstration of this, in Section 4 we explore two neighborhood aggregation functions, that are

$$\Phi(\mathbf{X}^{\ell-1}, \mathcal{N}_u) = \sum_{j \in \mathcal{N}_u} \mathbf{V}\mathbf{x}_j^{\ell-1}, \tag{6}$$

(which is also employed in Morris et al. (2019)) and the classical GCN aggregation

$$\Phi(\mathbf{X}^{\ell-1}, \mathcal{N}_u) = \mathbf{V} \sum_{j \in \mathcal{N}_u \cup \{u\}} \frac{1}{\sqrt{\hat{d}_j \hat{d}_u}} \mathbf{x}_j^{\ell-1}, \tag{7}$$

where $\mathbf{V}$ is the weight matrix, $\hat{d}_j$ and $\hat{d}_u$ are, respectively, the degrees of nodes j and u.

Finally, although we designed A-DGN with weight sharing in mind (for ease of presentation), a more general version of the framework, with layer-dependent weights $\mathbf{W}^\ell - (\mathbf{W}^\ell)^T$, is possible[3].

3 Related work

Deep Graph Network Nowadays, most of the DGNs typically relies on the concepts introduced by the Message Passing Neural Network (MPNN) (Gilmer et al., 2017), which is a general framework based on the message passing paradigm. The MPNN updates the representation for a node u at layer ℓ as

$$\mathbf{x}_u^\ell = \phi_U(\mathbf{x}_u^{\ell-1}, \sum_{j \in \mathcal{N}_u} \phi_M(\mathbf{x}_u^{\ell-1}, \mathbf{x}_v^{\ell-1}, \mathbf{e}_{uv})) \tag{8}$$

where ϕ_U and ϕ_M are respectively the *update* and *message* functions. Hence, the role of the message function is to compute the message for each node, and then dispatch it among the neighbors. On the other hand, the update function has the role of collecting the incoming messages and update the node state. A typical implementation of the MPNN model is $\mathbf{x}_u^\ell = \mathbf{W}\mathbf{x}_u^{\ell-1} + \sum_{j \in \mathcal{N}_u} \mathrm{MLP}(\mathbf{e}_{uv})\mathbf{x}_v^{\ell-1} + \mathbf{b}$, where $\mathbf{e}_{uv} \in \mathbb{R}^{d_e}$ is the edge feature vector between node u and v. Thus, by relaxing the concepts of stability and non-dissipation from our framework, MPNN becomes a specific discretization instance of A-DGN.

Depending on the definition of the update and message functions, it is possible to derive a variety of DGNs that mainly differ on the neighbor aggregation scheme (Kipf & Welling, 2017; Veličković et al., 2018; Hamilton et al., 2017; Xu et al., 2019; Defferrard et al., 2016; Hu et al., 2020). However, all these methods focus on presenting new and effective functions without questioning the stability and non-dissipative behavior of the final network. As a result, most of these DGNs are usually not able to capture long-term interactions. Indeed, only few layers can be employed without falling into the over-squashing phenomenon, as it is discussed by Alon & Yahav (2021).

Since the previous methods are all specific cases of MPNN, they are all instances of the discretized and unconstrained version of A-DGN. Moreover, a proper design of $\Phi(\mathbf{X}^{\ell-1}, \mathcal{N}_u)$ in A-DGN allows

[3]The dynamical properties discussed in this section are in fact still true even in the case of time varying $\mathbf{W}_t$ in Equation 2, provided that $\max_{i=1,\dots,d} Re(\lambda_i(\mathbf{J}(t))) \leq 0$ and $\mathbf{J}(t)$ change sufficiently slow over time (see (Ascher et al., 1995; Haber & Ruthotto, 2017)).

rethinking the discussed DGNs through the lens of non-dissipative and stable ODEs.Ming Chen et al. (2020), Zhou et al. (2021), and Eliasof et al. (2022) proposed three methods to alleviate over-smoothing, which is a phenomenon where all node features become almost indistinguishable after few embedding updates. Similarly to the forward Euler discretization, the first method employs identity mapping. It also exploits initial residual connections to ensure that the final representation of each node retains at least a fraction of input. The second method proposes a DGN that constrains the Dirichlet energy at each layer and leverages initial residual connections, while the latter overcomes the over-smoothing, based on aggregation from random paths defined over the graph nodes.

Continuous Dynamic Models Chen et al. (2018) introduce NeuralODE, a new family of neural network models that parametrizes the continuous dynamics of recurrent neural networks using ordinary differential equations. Similarly, Antisymmetric RNN (Chang et al., 2019) and Euler State Networks (Gallicchio, 2022) draw a connection between ODEs and, respectively, RNNs and Reservoir Computing architectures. Both methods focus on the stability of the solution and the employed numerical method.

Inspired by the NeuralODE approach, Poli et al. (2019) develops a DGN defined as a continuum of layers. In such a work, the authors focus on building the connection between ODEs and DGNs. We extend their work to include stability and non-dissipation, which are fundamental properties to preserve long-term dependencies between nodes and prevent gradient explosion or vanishing during training. Thus, by relaxing these two properties from our framework, the work by Poli et al. (2019) becomes a specific instance of A-DGN. Chamberlain et al. (2021) propose GRAND an architecture to learn graph diffusion as a partial derivative equation (PDE). Differently from GRAND, our framework designs an architecture that is theoretically non-dissipative and free from gradient vanishing or explosion. DGC (Wang et al., 2021) and SGC (Wu et al., 2019) propose linear models that propagate node information as the discretization of the graph heat equation, $\partial \mathbf{X}(t)/\partial t = -\mathbf{L}\mathbf{X}(t)$, without learning. Specifically, DGC mainly focus on exploring the influence of the step size ϵ in the Euler discretization method. Eliasof et al. (2021) and Rusch et al. (2022) present two methods to preserve the energy of the system, i.e., they mitigate the over-smoothing phenomenon. To do so, the first leverages the conservative mapping defined by hyperbolic PDEs, while the latter is defined as second-order ODEs that preserve the Dirichlet energy.

4 Experiments

In this section, we discuss the empirical assessment of our method. Specifically, in Section 4.1, we evaluate our framework on a graph property prediction task where we predict single source shortest path, node eccentricity, and graph diameter. In Section 4.2, we evaluate A-DGN on five classical graph benchmarks. The performance of A-DGN is assessed against DGN variants from the literature.

We carried the experiments on a Dell server with 4 Nvidia GPUs A100. We release openly the code implementing our methodology and reproducing our empirical analysis after acceptance.

4.1 Graph property prediction

Setup For the graph property prediction task, we considered three datasets extracted from the work of Corso et al. (2020). The analysis consists of classical graph theory tasks on undirected unweighted randomly generated graphs sampled from a wide variety of distributions. Specifically, we considered two node level tasks and one graph level task, which are single source shortest path (SSSP), node eccentricity, and graph diameter. Such tasks require capturing long-term dependencies in order to be solved, thus mitigating the over-squashing phenomenon. Indeed, in the SSSP task, we are computing the shortest paths between a given node u and all other nodes in the graph. Thus, it is fundamental to propagate not only the information of the direct neighborhood of u, but also the information of nodes which are extremely far from it. Similarly, for diameter and eccentricity.

We employed the same seed and generator as Corso et al. (2020) to generate the datasets, but we considered graphs with 25 to 35 nodes, instead of 15-25 nodes as in the original work, to increase the task complexity and lengthen long-range dependencies required to solve the task. As in the original work, we used 5120 graphs as training set, 640 as validation set, and 1280 as test set.

We explored the performance of two versions of A-DGN, i.e., weight sharing and layer dependent weights. Moreover, we employed two instances of our method leveraging the two aggregation functions in Equation 6 and 7. We will refer to the former as simple aggregation and to the latter as GCN-based aggregation. We compared our method with GCNII, two neuralODE-based models, i.e., GRAND and DGC, and the four most popular DGNs, i.e., GCN, GAT, GraphSAGE, and GIN.

We designed each model as a combination of three main components. The first is the encoder which maps the node input features into a latent hidden space; the second is the graph convolution (i.e., A-DGN or the DGN baseline); and the third is a readout that maps the output of the convolution into the output space. The encoder and the readout are Multi-Layer Perceptrons that share the same architecture among all models in the experiments.

We performed hyper-parameter tuning via grid search, optimizing the Mean Square Error (MSE). We trained the models using Adam optimizer for a maximum of 1500 epochs and early stopping with patience of 100 epochs on the validation error. For each model configuration, we performed 4 training runs with different weight initialization and report the average of the results.

We report in Appendix D the grid of hyper-parameters exploited for this experiment. We observe that even if we do not directly explore in the hyper-parameter space the terminal time T in which the node evolution produces the best embeddings, that is done indirectly by fixing the values of the step size ϵ and the maximum number of layers L, since $T = L\epsilon$.

Results We present the results on the graph property prediction in Table 1. Specifically, we report $log_{10}(\text{MSE})$ as the evaluation metric. We observe that our method, A-DGN, outperforms all the DGNs employed in this experiment. Indeed, by employing GCN-based aggregation, we achieve an error score that is on average 0.80 points better than the selected baselines. Notably, if we leverage the simple aggregation scheme, A-DGN increases the performance gap from the baselines. A-DGN with simple aggregation is 200% to 300% time better than the best baseline score in each dataset.

We observe that the main challenge when predicting diameter, eccentricity, or SSSP is to leverage not only local information but also global graph information. Such knowledge can only be learned by exploring long-range dependencies. Indeed, the three tasks are extremely correlated. All of them require to compute shortest paths in the graph. Thus, as for standard algorithmic solutions (e.g., Bellman–Ford (Bellman, 1958), Dijkstra's algorithm (Dijkstra, 1959)), more messages between nodes need to be exchanged in order to achieve accurate solutions. This suggests that A-DGN can better capture and exploit such information. Moreover, this indicates also that the simple aggregator is more effective than the GCN-based because the tasks are mainly based on counting distances. Thus, exploiting the information derived from the Laplacian operator is not helpful for solving this kind of algorithmic tasks.

Table 1: Mean test set $log_{10}(\text{MSE})$ and std averaged over 4 random weight initializations for each configuration. The lower the better. For each dataset we present the best performing variant of A-DGN, please refer to Apprendix E for the complete results.

	Diameter	**SSSP**	**Eccentricity**
GCN	0.7424±0.0466	$0.9499 \pm 9.18 \cdot 10^{-5}$	0.8468±0.0028
GAT	0.8221±0.0752	0.6951±0.1499	0.7909±0.0222
GraphSAGE	0.8645±0.0401	0.2863±0.1843	0.7863±0.0207
GIN	0.6131±0.0990	-0.5408±0.4193	0.9504±0.0007
GCNII	0.5287±0.0570	-1.1329±0.0135	0.7640±0.0355
DGC	0.6028±0.0050	-0.1483±0.0231	0.8261±0.0032
GRAND	0.6715±0.0490	-0.0942±0.3897	0.6602±0.1393
Our	**-0.5455**$_{\pm 0.0328}$	**-3.4020**$_{\pm 0.1372}$	**0.3046**$_{\pm 0.1181}$
Our(GCN)	0.2271$_{\pm 0.0804}$	-1.8288$_{\pm 0.0607}$	0.7177$_{\pm 0.0345}$

4.2 GRAPH BENCHMARKS

Setup In the graph benchmark setting we consider five well-known graph datasets for node classification, i.e., PubMed (Namata et al., 2012); coauthor graphs CS and Physics; and the Amazon co-purchasing graphs Computer and Photo from Shchur et al. (2018). Also for this class of experiments, we considered the same baselines and architectural choices as for the graph property prediction task. However, in this experiment we study only the version of A-DGN with weight sharing, since it achieve good performances with low training costs.

Within the aim to accurately assess the generalization performance of the models, we randomly split the datasets into multiple train/validation/test sets. Similarly to Shchur et al. (2018), we use 20 labeled nodes per class as the training set, 30 nodes per class as the validation set, and the rest as the test set. We generate 5 random splits per dataset and 5 random weight initialization for each configuration in each split.

We perform hyper-parameter tuning via grid search, optimizing the accuracy score. We train for a maximum of 10000 epochs to minimize the Cross-Entropy loss. We use an early stopping criterion that stops the training if the validation score does not improve for 50 epochs. We report in Appendix D the grid of hyper-parameters explored for this experiment.

Results We present the results on the graph benchmark in Table 2. Specifically, we report the accuracy as the evaluation metric. Even in this scenario, A-DGN outperforms the selected baseline,except in PubMed and Amazon Computers where GCNII is slightly better than our method. In this benchmark, results that the GCN-based aggregation produces higher scores with respect to the simple aggregation. Thus, additional local neighborhood features extracted from the graph Laplacian seem to strengthen the final predictions. It appears also that there is less benefit from including global information with respect to the graph property prediction scenario. As a result, exploiting extremely long-range dependencies do not strongly improve the performance as the number of layers increase.

To demonstrate that our approach performs well with many layers, we show in Figure 2 how the number of layers affects the accuracy score. Our model maintains or improve the performance as the number of layers increase. On the other hand, all the baselines obtain good scores only with one to two layers and most of them exhibit a strong performance degradation as the number of layers increases. Indeed, in the Coauthor CS dataset we obtain that GraphSAGE, GAT, GCN and GIN lose 24.5% to 78.2% of accuracy. We observe that DGC does not degrade its performance since the convolution do not contain parameters.

Although extreme long-range dependencies do not produce the same boost as in the graph property prediction scenario, including more than 5-hop neighborhoods is fundamental to improve state-of-the-art performances. As clear from Figure 2, this is not practical when standard DGNs are employed. On the other hand, A-DGN demonstrates that can capture and exploit such information without any performance drop.

Table 2: Mean test set accuracy and std in percent averaged over 5 random train/validation/test splits and 5 random weight initializations for each configuration in each split. The higher the better.

	PubMed	**Coauthor CS**	**Coauthor Physics**	**Amazon Computers**	**Amazon Photo**
GCN	76.75±1.29	90.34±0.31	92.80±0.44	81.63±0.93	89.14±0.59
GAT	75.64±1.27	81.57±1.02	89.25±0.82	76.36±0.89	85.58±0.91
GraphSAGE	74.96±1.69	89.93±0.79	92.47±0.94	79.37±1.38	88.04±0.85
GIN	76.24±1.86	89.26±0.31	91.40±0.70	79.64±0.72	87.69±1.16
GCNII	**77.39±1.36**	91.16±0.28	92.97±0.60	**82.72±0.98**	89.98±0.86
DGC	66.71±2.55	85.84±0.01	82.95±1.20	66.44±0.63	76.13±0.01
GRAND	76.18±1.56	89.20±0.62	90.72±0.87	81.09±0.70	89.05±0.73
Our	76.57±1.00	91.35±0.88	92.45±0.53	81.83±0.75	88.83±1.12
Our(GCN)	**76.82±0.86**	**91.71±0.43**	**93.27±0.62**	**82.35±0.89**	**90.52±0.40**

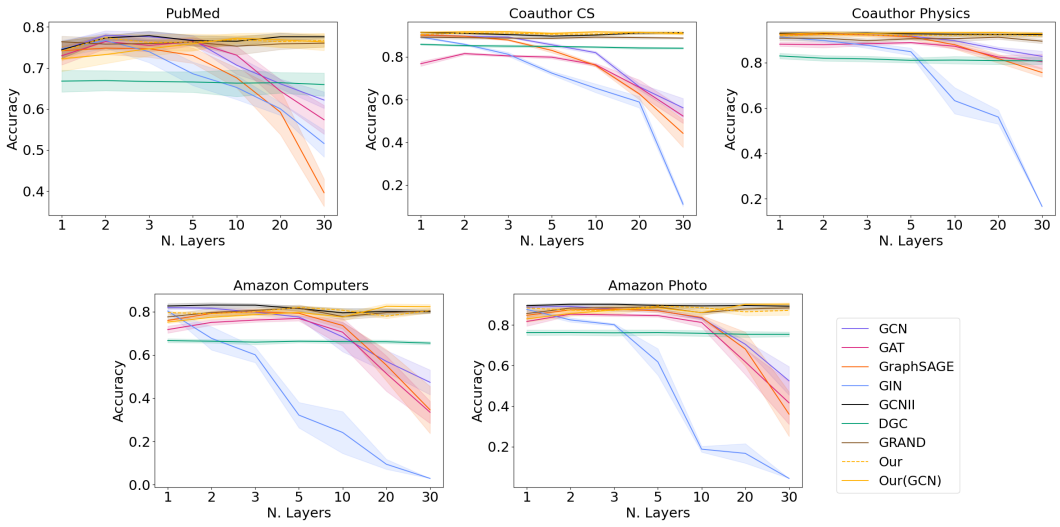

Figure 2: The test accuracy with respect to the number of layers on all the graph benchmark datasets. From the top left to the bottom, we show: PubMed, Coauthor CS, Coauthor Physics, Amazon Computers, and Amazon Photo. The accuracy is averaged over 5 random train/validation/test splits and 5 random weight initialization of the best configuration per split.

5 Conclusion

In this paper, we have presented *Anti-Symmetric Deep Graph Network* (A-DGN), a new framework for DGNs achieved from the study of ordinary differential equation (ODE) representing a continuous process of information diffusion on graphs. Unlike previous approaches, by imposing stability and conservative constraints to the ODE through the use of anti-symmetric weight matrices, the proposed framework is able to learn and preserve long-range dependencies between nodes. The A-DGN layer formulation is the result of the forward Euler discretization of the corresponding Cauchy problem on graphs. We theoretically prove that the differential equation corresponding to A-DGN is stable as well as non-dissipative. Consequently, typical problems of systems with unstable and lossy dynamics, e.g., no gradient explosion or vanishing, do not occur. Thanks to its formulation, A-DGN can be used to reinterpret and extend any classical DGN as a non-dissipative and stable ODEs.

Our experimental analysis, on the one hand, shows that when capturing long-range dependencies is important for the task, our framework largely outperform standard DGNs. On the other hand, it indicates the general competitiveness of A-DGN on several graph benchmarks. Overall, A-DGN shows the ability to effectively explore long-range dependencies and leverage dozens of layers without any noticeable drop in performance. For such reasons, we believe it can be a step towards the mitigation of the over-squashing problem in DGNs.

Looking ahead to future developments, we plan to extend the analysis presented in this paper to study DGN architectures resulting from alternative discretization methods of the underlying graph ODE, e.g., using adaptive multi-step schemes (Ascher & Petzold, 1998). Other future lines of investigation include extending the framework to dynamical graph structures, and evaluating its impact in the area of Reservoir Computing (Tanaka et al., 2019).

References

Uri Alon and Eran Yahav. 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.

U.M. Ascher, R.M.M. Mattheij, and R.D. Russell. *Numerical solution of boundary value problems for ordinary differential equations*. Classics in applied mathematics. Society for Industrial

and Applied Mathematics (SIAM), United States, unabridged, corr. republication. edition, 1995. ISBN 0-89871-354-4.

Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, USA, 1st edition, 1998. ISBN 0898714125.

Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, 2020. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2020.06.006. URL https://www.sciencedirect.com/science/article/pii/S0893608020302197.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

Ben Chamberlain, James Rowbottom, Maria I Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. Grand: Graph neural diffusion. In *International Conference on Machine Learning*, pp. 1407–1418. PMLR, 2021.

Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryxepo0cFX.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems*, 2020.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. NIPS'16, pp. 3844–3852, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, Peter W. Battaglia, Vishal Gupta, Ang Li, Zhongwen Xu, Alvaro Sanchez-Gonzalez, Yujia Li, and Petar Velickovic. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, pp. 3767–3776, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384469. doi: 10.1145/3459637.3481916. URL https://doi.org/10.1145/3459637.3481916.

Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

Moshe Eliasof, Eldad Haber, and Eran Treister. PDE-GCN: Novel architectures for graph neural networks motivated by partial differential equations. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=wWtk6GxJB2x.

Moshe Eliasof, Eldad Haber, and Eran Treister. pathGCN: Learning general graph spatial operators from paths. In *ICML*, 2022.

Claudio Gallicchio. Euler state networks. *arXiv preprint arXiv:2203.09382*, 2022.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1263–1272. JMLR.org, 2017.

Paul Glendinning. *Stability, Instability and Chaos: An Introduction to the Theory of Nonlinear Differential Equations*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 1994. doi: 10.1017/CBO9780511626296.

Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *CoRR*, abs/1705.03341, 2017. URL `http://arxiv.org/abs/1705.03341`.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HJlWWJSFDH`.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Zhewei Wei Ming Chen, Bolin Ding Zengfeng Huang, and Yaliang Li. Simple and deep graph convolutional networks. 2020.

Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M. Bronstein. Fake news detection on social media using geometric deep learning. *CoRR*, abs/1902.06673, 2019. URL `http://arxiv.org/abs/1902.06673`.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33014602. URL `https://doi.org/10.1609/aaai.v33i01.33014602`.

Galileo Mark Namata, Ben London, Lise Getoor, and Bert Huang. Query-driven active surveying for collective classification. In *Workshop on Mining and Learning with Graphs*, 2012. URL `http://linqs.cs.umd.edu/basilic/web/Publications/2012/namata:mlg12-wkshp/namata-mlg12.pdf`.

Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

T Konstantin Rusch, Benjamin P Chamberlain, James Rowbottom, Siddhartha Mishra, and Michael M Bronstein. Graph-coupled oscillator networks. *arXiv preprint arXiv:2202.02296*, 2022.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018. URL `http://arxiv.org/abs/1811.05868`.

Gouhei Tanaka, Toshiyuki Yamane, Jean Benoit Héroux, Ryosho Nakane, Naoki Kanazawa, Seiji Takeda, Hidetoshi Numata, Daiju Nakano, and Akira Hirose. Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123, 2019.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`. accepted as poster.

Yifei Wang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. Dissecting the diffusion process in linear graph convolutional networks. *CoRR*, abs/2102.10739, 2021. URL https://arxiv.org/abs/2102.10739.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6861–6871. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/wu19e.html.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 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.

Kaixiong Zhou, Xiao Huang, Daochen Zha, Rui Chen, Li Li, Soo-Hyun Choi, and Xia Hu. Dirichlet energy constrained learning for deep graph neural networks. *Advances in neural information processing systems*, 2021.

Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 06 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty294. URL https://doi.org/10.1093/bioinformatics/bty294.

A Proof of Proposition 1

Let us consider the ODE defined in Equation 2 and analyze the sensitivity of its solution to the initial conditions. Following (Chang et al., 2019), we differentiate both sides of Equation 2 with respect to $\mathbf{x}_u(0)$, obtaining:

$$\frac{d}{dt}\left(\frac{\partial \mathbf{x}_u(t)}{\partial \mathbf{x}_u(0)}\right) = \mathbf{J}(t)\frac{\partial \mathbf{x}_u(t)}{\partial \mathbf{x}_u(0)}. \tag{9}$$

Assuming the Jacobian does not change significantly over time, we can apply results from autonomous differential equations (Glendinning, 1994) and solve Equation 9 analytically as follows:

$$\frac{\partial \mathbf{x}_u(t)}{\partial \mathbf{x}_u(0)} = e^{t\mathbf{J}} = \mathbf{T}e^{t\mathbf{\Lambda}}\mathbf{T}^{-1} = \mathbf{T}\big(\sum_{k=0}^{\infty}\frac{(t\mathbf{\Lambda})^k}{k!}\big)\mathbf{T}^{-1}, \tag{10}$$

where $\mathbf{\Lambda}$ is the diagonal matrix whose non-zero entries contain the eigenvalues of $\mathbf{J}$, and $\mathbf{T}$ has the eigenvectors of $\mathbf{J}$ as columns. The qualitative behavior of $\partial\mathbf{x}(t)/\partial\mathbf{x}(0)$ is then determined by the real parts of the eigenvalues of $\mathbf{J}$. When $\max_{i=1,...,d} Re(\lambda_i(\mathbf{J}(t))) > 0$, a small perturbation of the initial condition (i.e., a perturbation on the input graph) would cause an exponentially exploding difference in the nodes representations, and the system would be unstable. On the contrary, for $\max_{i=1,...,d} Re(\lambda_i(\mathbf{J}(t))) \ll 0$ input perturbations would rapidly vanish, hence the nodes' representations would be insensitive to differences in the input graph, and the system would be dissipative. If $Re(\lambda_i(\mathbf{J}(t))) = 0$ for $i = 1, ..., d$ then the magnitude of $\partial\mathbf{x}(t)/\partial\mathbf{x}(0)$ is approximately constant over time, and the input graph information is effectively propagated through the successive transformations into the final nodes' representations, with at most moderate amplification or loss. In this last case, the system is hence both stable and non-dissipative.

Let us now consider a loss function $\mathcal{L}$, and observe that its sensitivity to the initial condition (i.e., the input graph) $\partial\mathcal{L}/\partial\mathbf{x}_u(0)$ is proportional to $\partial\mathbf{x}_u(t)/\partial\mathbf{x}_u(0)$. Hence, in light of the previous considerations, if $Re(\lambda_i(\mathbf{J}(t))) = 0$ for $i = 1, ..., d$, then the magnitude of $\partial\mathcal{L}/\partial\mathbf{x}_u(0)$, which is the longest gradient chain that we can obtain during back-propagation, stays constant over time. The backward propagation is then stable and non-dissipative, and no gradient vanishing or explosion can occur during training.

B PROOF OF PROPOSITION 2

The Jacobian matrix of Equation 4 is given by:

$$\mathbf{J}(t) = \text{diag}\left[\sigma'\left((\mathbf{W}-\mathbf{W}^T)\mathbf{x}_u(t) + \Phi(\mathbf{X}(t), \mathcal{N}_u) + \mathbf{b}\right)\right](\mathbf{W}-\mathbf{W}^T). \tag{11}$$

Following (Chang et al., 2019; 2018), we can see the right-hand side of Equation 11 as the result of a matrix multiplication between an invertible diagonal matrix and an anti-symmetric matrix. Specifically, defining $\mathbf{A} = \text{diag}\left[\sigma'\left((\mathbf{W}-\mathbf{W}^T)\mathbf{x}_u(t) + \Phi(\mathbf{X}(t), \mathcal{N}_u) + \mathbf{b}\right)\right]$ and $\mathbf{B} = \mathbf{W} - \mathbf{W}^T$, we have $\mathbf{J}(t) = \mathbf{AB}$.

Let us now consider an eigenpair of $\mathbf{AB}$, where the eigenvector is denoted by $\mathbf{v}$ and the eigenvalue by λ. Then:

$$\begin{aligned} \mathbf{ABv} &= \lambda\mathbf{v}, \\ \mathbf{Bv} &= \lambda\mathbf{A}^{-1}\mathbf{v}, \\ \mathbf{v}^*\mathbf{Bv} &= \lambda(\mathbf{v}^*\mathbf{A}^{-1}\mathbf{v}) \end{aligned} \tag{12}$$

where $*$ represents the conjugate transpose. On the right-hand side of Equation 12, we can notice that the $(\mathbf{v}^*\mathbf{A}^{-1}\mathbf{v})$ term is a real number. Recalling that $\mathbf{B}^* = \mathbf{B}^T = -\mathbf{B}$ for a real anti-symmetric matrix, we can notice that $(\mathbf{v}^*\mathbf{Bv})^* = \mathbf{v}^*\mathbf{B}^*\mathbf{v} = -\mathbf{v}^*\mathbf{Bv}$. Hence, the $\mathbf{v}^*\mathbf{Bv}$ term on the left-hand side of Equation 12 is an imaginary number. Thereby, λ needs to be purely imaginary, and, as a result, all eigenvalues of $\mathbf{J}(t)$ are purely imaginary.

C DATASETS DESCRIPTION AND STATISTICS

In the graph property prediction (GPP) experiments, we employed the same generation procedure as in Corso et al. (2020). Graphs are randomly sampled from several graph distributions, such as Erdős–Rényi, Barabasi-Albert, and grid. Each node have random identifiers as input features. Target values represent single source shortest path, node eccentricity, and graph diameter.

PubMed is a citation network where each node represents a paper and each edge indicates that one paper cites another one. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The class labels represent the papers categories.

Amazon Computers and Amazon Photo are portions of the Amazon co-purchase graph, where nodes represent goods and edges indicate that two goods are frequently bought together. Node features are bag-of-words encoded product reviews, and class labels are given by the product category.

Coauthor CS and Coauthor Physics are co-authorship graphs extracted from the Microsoft Academic Graph[4] where nodes are authors, that are connected by an edge if they co-authored a paper. Node features represent paper keywords for each author's papers, and class labels indicate most active fields of study for each author.

Table 3 contains the statistics of the employed datasets, sorted by graph density. The density of a graph is computed as the ratio between the number of edges and the number of possible edges, i.e., $d = \frac{|\mathcal{E}|}{|\mathcal{V}|(|\mathcal{V}|-1)}$.

D EXPLORED HYPER-PARAMETER SPACE

In Table 4 and Table 5 we report the grids of hyper-parameters employed in our experiments by each method. We recall that the hyper-parameters ϵ and γ refer only to our method.

E COMPLETE RESULTS

In Table 6 we report the complete results for the graph property prediction benchmark, including all the versions and cofiguration of our A-DGN. We reported the average time per epoch (measured

[4]https://www.kdd.org/kdd-cup/view/kdd-cup-2016/Data

Table 3: Datasets statistics ordered by graph density.

	Nodes	**Edges**	**Features**	**Classes**	**Density**
GPP	25 - 35	22 - 553	2	—	0.0275 - 0.5
Amazon Computers	13,752	491,722	767	10	$2.6e^{-3}$
Amazon Photo	7,650	238,162	745	8	$4.1e^{-3}$
PubMed	19,717	88,648	500	3	$2.3e^{-4}$
Coauthor Physics	34,493	495,924	8,415	5	$4.2e^{-4}$
Coauthor CS	18,333	163,788	6,805	15	$4.9e^{-4}$

Table 4: The grid of hyper-parameters employed during model selection for the graph property prediction task.

	Configs
optimizer	Adam
learning rate	0.003
weight decay	10^{-6}
n. layers	1, 5, 10, 20
embedding dim	10, 20, 30
σ	tanh
ϵ	1, 10^{-1}, 10^{-2}, 10^{-3}
γ	1, 10^{-1}, 10^{-2}, 10^{-3}

in seconds) for each model in the graph property prediction task in Table 7. Each model in the evaluation has 20 layers and an embedding dimesion equal to 30.

Table 5: The grid of hyper-parameters employed during model selection in the graph benchmark.

	Configs
optimizer	AdamW
learning rate	10^{-2}, 10^{-3}, 10^{-4}
weight decay	0.1
n. layers	1, 2, 3, 5 ,10, 20, 30
embedding dim	32, 64, 128
σ	tanh
ϵ	1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}
γ	1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}

Table 6: Mean test set $log_{10}(\text{MSE})$ and std averaged over 4 random weight initializations for each configuration. The subscript *ws* stands for weight sharing, while *ldw* for layer dependent weights. The lower the better.

	Diameter	**SSSP**	**Eccentricity**
GCN	0.7424±0.0466	$0.9499 \pm 9.18 \cdot 10^{-5}$	0.8468±0.0028
GAT	0.8221±0.0752	0.6951±0.1499	0.7909±0.0222
GraphSAGE	0.8645±0.0401	0.2863±0.1843	0.7863±0.0207
GIN	0.6131±0.0990	-0.5408±0.4193	0.9504±0.0007
GCNII	0.5287±0.0570	-1.1329±0.0135	0.7640±0.0355
DGC	0.6028±0.0050	-0.1483±0.0231	0.8261±0.0032
GRAND	0.6715±0.0490	-0.0942±0.3897	0.6602±0.1393
Our_{ws}	$-0.5188_{\pm 0.1812}$	$-3.2417_{\pm 0.0751}$	$0.4296_{\pm 0.1003}$
Our_{ldw}	$\mathbf{-0.5455}_{\pm 0.0328}$	$\mathbf{-3.4020}_{\pm 0.1372}$	$\mathbf{0.3046}_{\pm 0.1181}$
Our_{ws}(GCN)	$0.2646_{\pm 0.0402}$	$-1.3659_{\pm 0.0702}$	$0.7177_{\pm 0.0345}$
Our_{ldw}(GCN)	$0.2271_{\pm 0.0804}$	$-1.8288_{\pm 0.0607}$	$0.7235_{\pm 0.0211}$

Table 7: Average time per epoch (measured in seconds) and std, averaged over 4 random weight initializations. Each time is obtained by employing 20 layers and an embedding dimension equal to 30. The subscript *ws* stands for weight sharing, *ldw* for layer dependent weights.

	Diameter	**SSSP**	**Eccentricity**
GCN	32.45±2.54	17.44±3.85	11.78±2.43
GAT	20.20±5.18	26.41±8.34	17.28±1.92
GraphSAGE	13.12±2.99	13.12±2.99	8.20±0.75
GIN	6.63±0.28	21.16±2.33	14.22±3.17
GCNII	13.13±6.85	14.96±7.17	15.70±3.92
DGC	8.97±9.07	12.54±1.62	7.21±11.10
GRAND	133.84±42.57	109.15±27.49	202.46±85.01
Our_{ws}	8.42±2.71	7.86±2.11	13.18±9.07
Our_{ldw}	14.59±8.67	10.47±6.95	14.04±11.60
Our_{ws}(GCN)	13.08±5.49	28.74±10.92	16.26±4.58
Our_{ldw}(GCN)	40.50±16.45	26.72±17.98	24.43±19.10