Transforming PageRank into an Infinite-Depth Graph Neural Network

Andreas Roth[0000−0002−0515−7635](✉) and Thomas Liebig[0000−0002−9841−1101]

Artificial Intelligence Group, TU Dortmund, Dortmund, Germany
{andreas.roth,thomas.liebig}@tu-dortmund.de

Abstract. Popular graph neural networks are shallow models, despite the success of very deep architectures in other application domains of deep learning. This reduces the modeling capacity and leaves models unable to capture long-range relationships. The primary reason for the shallow design results from over-smoothing, which leads node states to become more similar with increased depth. We build on the close connection between GNNs and PageRank, for which personalized PageRank introduces the consideration of a personalization vector. Adopting this idea, we propose the Personalized PageRank Graph Neural Network (PPRGNN), which extends the graph convolutional network to an infinite-depth model that has a chance to reset the neighbor aggregation back to the initial state in each iteration. We introduce a nicely interpretable tweak to the chance of resetting and prove the convergence of our approach to a unique solution without placing any constraints, even when taking infinitely many neighbor aggregations. As in personalized PageRank, our result does not suffer from over-smoothing. While doing so, time complexity remains linear while we keep memory complexity constant, independently of the depth of the network, making it scale well to large graphs. We empirically show the effectiveness of our approach for various node and graph classification tasks. PPRGNN outperforms comparable methods in almost all cases. [1]

Keywords: Machine Learning · Graph Neural Networks · PageRank

1 Introduction

Graph-structured data is found in many real-world applications ranging from social networks [26] to biological structures [28]. Steadily growing amounts of data lead to emerging solutions that can extract relevant information from these data types. Tasks like providing recommendations [41], predicting the state of traffic [8] or the classification of entire graphs into distinct categories [39] are some of the tasks of research interest. Approaches based on deep learning have found great success for grid-structured data, e.g., in image processing [20] and natural language processing [34]. Graph Neural Networks (GNNs) [21] adopt

[1] Our code is available at: https://github.com/roth-andreas/pprgnn

the ideas from convolutions in euclidean space for irregular non-euclidean domains. These methods directly consider the graph structure when performing convolution operations.

One of the challenges of GNNs is to capture long-range dependencies. Recently popular methods use an aggregation scheme, in which k layers of graph convolutions combine the information from k-hops around each node [21, 35]. Several issues, most dominantly over-smoothing [24, 38] and memory consumption [17, 6, 42] were found to prevent deep models, as in image processing [20]. Several recent efforts explore options to enable more layers and even formulate infinite-depth equations. However, previous work still only allows a limited depth [30, 38] or places hard constraints on the parameters [16] or the architecture [2].

As identified by [22], GNNs are closely related to PageRank [27], which in its basic version only depends on the graph structure, not on the initial distribution. Personalized PageRank [27] introduces a chance to reset PageRank to a teleportation vector, allowing the result to depend not only on the graph structure but also on an initial distribution. We show how the idea of personalization can be adopted to GNNs and propose the Personalized PageRank Graph Neural Network (PPRGNN), an infinitely deep GNN that adds a chance to reset the neighbor aggregation back to the initial state. In order to prove the convergence of PPRGNN to a unique solution when iterating infinitely many times, we modify the chance of resetting to be dynamic based on the distance to the root node. As in personalized PageRank, our approach does not suffer from over-smoothing and the locality of node features around their root nodes is preserved. Due to the large depths, far distant information still influences resulting node representations.

In addition, we provide rich theoretical intuition for the success of our formulation and our design choices. While the depth is theoretically always infinite, the practically effective depth is adaptive and varies depending on the learned parameters, the graph structure, and the observed features. We also provide a way to control the convergence rate since different levels of localization are effective for different types of graphs [1, 15]. Furthermore, contrary to previous infinite-depth approaches, we do not impose any constraints on parameters or the model's architecture. To allow scaling to large graphs despite the infinite depth, we design an efficient gradient computation that remains constant in memory and execution time. We validate our proposed approach against comparable methods on various inductive and transductive node and graph classification tasks. Our approach outperforms related methods in almost all cases by considerable margins, while most other approaches are within a competitive range only for individual tasks. The experimental execution time is also improved compared to previous infinite-depth approaches.

The rest of our work is structured as follows. Section 2 introduces our notation and relevant basics in personalized PageRank and GNNs. We describe recent related approaches in Section 3. Our method is detailed in Section 4, and a

comprehensive evaluation is presented in Section 5. We discuss our results and potential future directions in Section 6.

2 Preliminaries

We represent a graph $G = (V, E)$ as the tuple of n nodes $V = \{v_1, v_2, \ldots, v_n\}$ and a set of edges E between pairs of nodes. We construct an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ describing the connectivity between pairs of nodes from the E. Entries $a_{ij} \in \mathbf{A}$ indicate the strength of an edge between nodes v_i and v_j, a zero-entry indicates the absence of an edge. Our method assumes undirected edges, e.g., $a_{ij} = a_{ji}$, but it is straightforward to apply it to directed graphs. We use a normalized version $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ of the adjacency matrix, potentially with self-loops. Each node v_k has a set of F features $\mathbf{u}_k \in \mathbb{R}^F$ associated with them. The feature matrix $\mathbf{U} \in \mathbb{R}^{n \times F}$ contains all nodes' stacked feature vectors $\mathbf{u}_k$. We define the node neighborhood $N_i = \{v_j | \tilde{\mathbf{A}}_{ij} > 0\}$ as the set of all nodes connected to v_i.

2.1 Personalized PageRank

Our approach inherits basic concepts and intuition from personalized PageRank [27], which we briefly describe here. PageRank [27] was originally introduced to score the importance of webpages for web searches. In their work, webpages represent individual nodes in a graph and links on these webpages are modeled as directed edges between these nodes. The solution to PageRank is the fixed point of the equation

$$\mathbf{r} = \mathbf{A}\mathbf{r}\,, \tag{1}$$

with $\mathbf{r} \in \mathbb{R}^n$ being the dominant eigenvector of $\mathbf{A}$. The vector $\mathbf{r}$ can be obtained by power iteration with an arbitrary initial $\mathbf{r}_0$ [27]. For an intuitive interpretation of Eq. (1), we can interpret $\mathbf{A}$ as the stochastic transition matrix over the graph, providing a connection to a random walk. Therefore the stationary probability distribution induced by a random walk is the same as $\mathbf{r}$ in the limit [27]. This also results in r only depending on the graph structure and not on prior information available for nodes. Therefore, the authors also introduce personalized PageRank [27]

$$\mathbf{r} = (1 - \alpha)\mathbf{A}\mathbf{r} + \alpha\mathbf{u} \tag{2}$$

that adds a chance α as a way to teleport back to a personalization vector $\mathbf{u} \in \mathbb{R}^n$ representing an initial distribution over all pages. The corresponding interpretation for a random walk is to introduce a chance to reset the random walk to the personalization vector [27].

2.2 Graph Neural Networks

Another concept we build upon are Graph Neural Networks (GNNs), specifically their subtype of Message-Passing Neural Networks (MPNNs) [13]. GNNs apply

permutation equivariant operations to graph structured data in order to identify task-specific features. Originating from spectral graph convolutions [18] as a localized first-order approximation, each message-passing operation updates the node states $\mathbf{h}_i$ by combining the information of the direct neighborhood N_i for each node v_i [36]. The general framework can be described as a node-wise update function

$$\mathbf{h}_i^{(l+1)} = \psi \left(\mathbf{h}_i^{(l)}, \bigoplus_{j \in N_i} \omega(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}) \right) , \tag{3}$$

for each state $\mathbf{h}_i$, using some functions ω and ψ and a permutation invariant aggregation function $\bigoplus$. In this work, we will demonstrate our approach using the very basic instantiation of this framework, namely the Graph Convolutional Network (GCN) [21]. Making use of the normalized adjacency matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{n \times n}$, the GCN can be expressed in matrix notation

$$\mathbf{H}^{(l+1)} = \phi \left(\tilde{\mathbf{A}} \mathbf{H}^{(l)} \mathbf{W}^{(l)} \right) \tag{4}$$

using a linear transformation $\mathbf{W} \in \mathbb{R}^{d \times d}$ and ϕ as an element-wise activation function. $\mathbf{H}^{(l)} \in \mathbb{R}^{n \times d}$ contains the node states $\mathbf{h}_i^{(l)}$ of all nodes after layer l. Each layer aggregates information only from direct neighborhood N_i for each node v_i. Thus, after k such layers, each node only has access to information a maximum of k hops away. Given this property, choosing any number k of these layers leads to information at $k+1$ hops away being impossible to be considered for making predictions. Moreover, even when the number of layers k can be selected to be sufficient for all potentially considered graphs, a large number k leads to various additional issues that we will describe next.

Over-smoothing. Recent work found that stacking many layers of Eq. (4) leads to a degradation of experimental performance that is caused by an effect called over-smoothing [21, 24, 38]. Li et al. [24] show that Eq. (4) is a special form of Laplacian smoothing leading to node representation becoming more similar the more layers are added. They prove that Laplacian smoothing converges to a linear combination of dominant eigenvectors. While some smoothing is needed to share information between nodes, representations eventually become indistinguishable with too much smoothing, thus making accurate data-dependant predictions harder [24].

On a similar note, [38] find a close connection between k layers of Eq. (4) and a k-step random walk. They find that both to converge the limit distribution of the random walk. In the limit, a random walk becomes independent of the root nodes and therefore loses the locality property of individual nodes. Therefore, representations become independent of the starting node and initial node features, thus becoming indistinguishable [38]. In practice, the performance of Eq. (4) already degrades with more than two layers in many cases [21].

Memory Complexity. Another reason that prevents GNNs from being deep models is the memory complexity. Graphs can quickly surpass a million nodes, which leads to out-of-memory issues due to the linear memory requirements $\mathcal{O}(kn)$ in the number of layers k and the number of nodes n. Several approaches try to lower the memory complexity by only considering samples of nodes from a local neighborhood [17]. Due to an effect known as the neighborhood explosion, the number of nodes in the k-hop neighborhood $\mathcal{O}(d^k)$ explodes, with d being the average node degree. Thus, for a large number of layers k, the benefit vanishes. Other approaches cluster the graph into subgraphs and use these for training [6, 42], but cannot leverage the full potential of the entire graphs relationships. Therefore, this issue needs to be considered when designing deep graph neural networks.

3 Related Work

Several approaches aim to increase the depth of MPNNs and simultaneously deal with over-smoothing and memory consumption. Rong et al. [30] found over-smoothing to occur faster for nodes with many incoming edges and propose DropEdge as the equivalent to dropout in regular neural networks. They randomly sample edges to remove during each training epoch and show that the effect of over-smoothing gets slowed down. Klicpera et al. [23] propose a diffusion process that they find to be beneficial for semi-supervised node classification tasks for graphs with high homophily but encounters problems with complex graphs. Zhu et al. [45] further discuss the issue of settings with low homophily. Li et al. [24] co-train a random walk model that explores the global graph topology. Inspired by the findings from ResNet [20], Chen et al. [4] propose GCNII that makes use of residual connections in two ways. In each layer, they add an initial residual connection to the input state $\mathbf{H}^{(0)}$ and an identity mapping to the weights, which were shown to have beneficial properties [19]. Xu et al. [38] combine the results of all intermediate iterations in JKNet. Other works find a rescaling of the weights to alleviate the over-smoothing problem [44, 25]. While these approaches help reduce the effect of over-smoothing, they are limited in practical depth and the issue still arises.

3.1 Infinite-Depth Graph Neural Networks

Evaluating the option of repeating iterations infinitely many times have been analyzed in various approaches [14, 11, 22, 16]. These methods iterate some graph convolution until convergence by employing weight-sharing and ensuring the convergence of their formulations. When using an equation for an infinite-depth GNN, the result needs to converge to a unique solution. We summarize this under the following definition of well-posedness.

Definition 1. *(Well-posedness). Given an input matrix* $\mathbf{X} \in \mathbb{R}^{N \times D}$*, an equation* $\mathbf{Y} = g(\mathbf{X})$*, with* g *being an infinitely recursive function is well-posed, if*

1. *The solution* $\mathbf{Y}$ *is unique*
2. $g(\mathbf{X})$ *converges to the unique solution* $\mathbf{Y}$.

While the GCN (Eq. (4)) is not generally well-posed, our work proposes a similar equation that we prove to be well-posed. We start by reviewing two recent approaches to infinitely deep graph neural networks that serve as the starting point for our contribution. The first [22] is derivated from the PageRank [27] algorithm, the other is the fixed-point solution of an equilibrium equation [16].

APPNP. Klicpera et al. [22] propose a propagation scheme derived from personalized PageRank [27]. They identify the connection between the limit distribution of MPNNs and PageRank, with both losing focus on the local neighborhood of the initial state. As personalized PageRank was introduced as a solution to this issue for PageRank [27], they adopt this idea for MPNNs. They set the personalization vector $\mathbf{r}$ from Eq. (2) to the hidden state of all nodes $\mathbf{H}^{(0)}$. A chance α to teleport back to the root node preserves the local neighborhood with the tunable parameter. Klicpera et al. [22] transfer this idea to MPNNs with Approximate Personalized Propagation of Neural Predictions (APPNP) [22]

$$\mathbf{H}^{(l+1)} = (1-\alpha)\tilde{\mathbf{A}}\mathbf{H}^{(l)} + \alpha\mathbf{H}^{(0)} \tag{5}$$

that repeatedly, potentially infinitely many times, aggregates the neighborhood. They also add a chance of going back to the initial state $\mathbf{H}^{(0)} = f_\theta(\mathbf{U})$, that is be the output of previous layers f_θ. They show that Eq. (5) is well-posed for any $\alpha \in (0, 1], \mathbf{H}^{(0)} \in \mathbb{R}^{N\times D}, \tilde{\mathbf{A}} \in \mathbb{R}^{N\times N}$ with $\det(\tilde{\mathbf{A}}) \leq 1$. Typical normalizations $\tilde{\mathbf{A}}$ of the adjacency matrix satisfy this property. Notably, Eq. (5) does not utilise any learnable parameters. They rather propose to separate the propagation scheme in Eq. (5) from the learnable part, by making $\mathbf{H}^{(0)} = f_\theta(\mathbf{U})$ as node-wise application of a MLP. This method is proposed only for semi-supervised node classification tasks, with a softmax activation employed to transform the output of the last iteration $\mathbf{H}^{(K)}$ of Eq. (5) to class predictions.

Implicit Graph Neural Networks. Independently, Gu et al. [16] propose the Implicit Graph Neural Network (IGNN) by adapting the general implicit framework [10] for graph convolutions. They obtain the fixed-point solution of a non-linear equilibrium equation

$$\mathbf{X} = \phi(\mathbf{W}\mathbf{X}\tilde{\mathbf{A}} + f_\theta(\mathbf{U})) \tag{6}$$

by iterating it until convergence. While not being well-posed in general, they prove the well-posedness of Eq. (6) for the specific case that $\lambda_{pf}(|\mathbf{A}^T \otimes \mathbf{W}|) < 1$ with λ_{pf} being the Perron-Frobenius (PF) eigenvalue. They make use of the Kronecker product $\otimes$ and the Perron-Frobenius theory [3]. Since $\tilde{\mathbf{A}}$ is fixed, the matrix of parameters $\mathbf{W}$ needs to be strictly constrained to fulfill $\lambda_{pf}(|\mathbf{A}^T \otimes \mathbf{W}|) < 1$. The set $\mathcal{M}$ of allowed matrices $\mathbf{W}$ forms an $\mathcal{L}_1$-ball, with any weight matrix outside the ball not leading to convergence. Remaining inside this ball

cannot be guaranteed by regular gradient descent. Instead, after each step of regular gradient descent, they project the result to the closest point on the ball using projected gradient descent, for which efficient algorithms exist [9]. While their inspiring work shows great experimental results, we identify a couple of shortcomings with. Many weight matrices cannot be used given the strict constraint on $\mathbf{W}$, hindering the model capacity. Further, the projection onto the $\mathcal{L}_1$-ball changes the direction of the gradient update away from the steepest descent. Therefore optimization steps are less effective in reducing the models' loss. The strict constraint and the resulting projection step also add complexity to the method's theoretical derivation and practical implementation. Considering different neighborhood sizes was found to be important when applying graph algorithms to varying graph types [1, 15], not having a way to control the effective depth of the model is also unsatisfying.

4 PageRank Graph Neural Network

The solution of PageRank is the stationary probability distribution that is independent of the input. Given the close relation between PageRank (Eq. (1)) and MPNNs (Eq. (4)), the locality of the data and the influence of the input features also diminish with a MPNN, as identified by [22]. As personalized PageRank was introduced to prevent the loss of focus for PageRank [27], we introduce the Personalized PageRank Graph Neural Network (PPRGNN) based on personalized PageRank, that similarly assures the locality of the node states in the limit. Using the initial state $f_\theta(\mathbf{U})$ as personalization matrix for teleportation [22], PPRGNN can be understood as repeatedly applying graph convolutions with a chance to teleport back to this initial state. We assure the convergence of PPRGNN to a unique solution, so our method allows an arbitrary amount of layers - potentially infinitely many - without suffering from over-smoothing. Practically, we iterate graph convolutions until further iterations have negligible impact and our solution is close to the limit distribution. In this work, we adopt GCNs [21], which are the basic version of MPNNs, but these are directly replaceable by more advanced types.

We denote the chance of traversing the graph further by α_l. Rewriting the formulation of the GCN in a similar way to personalized PageRank, we come up with our formulation

$$\mathbf{H}^{(l+1)} = \phi\left(\alpha_l \tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W} + f_\theta(\mathbf{U})\right) \tag{7}$$

with $\mathbf{H}^{(0)} = \mathbf{0}$ that utilizes shared and unconstrained parameters $\mathbf{W}$. Due to the recursive nature and no constraints, exponential growth in $\mathbf{W}$ prevents well-posedness for any fixed α_l. The issue with having no guarantees for convergence is that the furthest distant nodes are multiplied with the highest exponential of $\mathbf{W}$, which potentially dominates the result. As in PageRank, these only depend on the graph structure and not on the node features, leading to the loss of locality of the resulting node features.

Our core idea to guarantee convergence of Eq. (7) without constraining the parameters as in [16] is to reduce the chance of expanding further α_l with the distance to the root node. As the message-passing formulation is connected to a random walk on the graph, another interpretation is to increase the chance of resetting the random walk with the number of steps taken. When n is the number of steps taken in that walk, we find using a decay of $\alpha_n = \frac{1}{n}$ to be sufficient for converging to a unique solution. The recursive nature of our formulation leads to a multiplication of all α_n, resulting in the influence to decay by $\frac{1}{n!}$. Because the recursive application of $\mathbf{W}$ only leads to an exponential $\mathbf{W}^n$ growth, the equation converges. For control over the effective depth, i.e., the speed of convergence and numerical stability, we use $\frac{1}{1+n\epsilon}$ and formally prove its convergence for any $\epsilon > 0$ later. We set the value for teleporting back to $f_\theta(\mathbf{U})$ fixed to 1 because in the limit the chance $(1 - \alpha_l)$ would become very small for close neighbors, leading to the same issues of over-smoothing that we described in section 2.2.

Setting α_l in Eq. (7) accordingly to our findings, the following issue arises: The most distant nodes are processed first in Eq. (7), and the direct neighbors are processed in the last iteration. This results from recursively applying the adjacency matrix $\tilde{\mathbf{A}}$ on the input, leading to the initial state being transformed k times by $\tilde{\mathbf{A}}$. Thus, for calculating $\mathbf{H}^{(1)}$, the expansion factor α_0 needs to be minimal, which is the opposite of using the iteration l as n.

In case we are interested in a fixed number k of total iterations, we can directly set $\alpha_l = \frac{1}{1+(k-l-1)\epsilon}$ for each layer l. When using a fixed number of iterations, this approach is ready for usage directly. Since we are interested in the case when $k \to \infty$, starting with α_0 poses a challenge.

For a theoretical analysis of the convergence of Eq. (7), an equation that can be iterated infinitely-deep independently of k is desired. We achieve this by setting the index variable to $n = k - l$ resulting in the flipped equation

$$\mathbf{G}^{(n)} = \phi\left(\beta_n \tilde{\mathbf{A}}\mathbf{G}^{(n+1)}\mathbf{W} + f_\theta\left(\mathbf{U}\right)\right) \tag{8}$$

with $\beta_n = \alpha_{k-l-1}$ that is semantically unchanged, i.e., $\mathbf{G}^{(0)} = \mathbf{H}^{(k)}$ for any k used for both $\mathbf{G}$ and $\mathbf{H}$. Calculating $\mathbf{G}^{(n)}$ from a given $\mathbf{G}^{(n+1)}$ can be performed without knowing k beforehand, helping us in the theoretical analysis by expanding the recursive equation infinitely deep without the need to set a fixed value for k. It also leads to a cleaner proof of convergence, which we will provide next. We further simplify our notation by denoting $\mathbf{G}^{(l;k)}$ as the result of k iterations performed by setting $\mathbf{G}^{k+l+1} = \mathbf{0}$, resulting in $\mathbf{G}^{(l)}$.

Theorem 1. *The result of* $\mathbf{G}^{(l;k)}$ *using the equation* $\mathbf{G}^{(n)} = \phi\left(\beta_n \tilde{\mathbf{A}}\mathbf{G}^{(n+1)}\mathbf{W} + \mathbf{B}\right)$ *with* $\beta_n = \frac{1}{1+n\epsilon}$ *converges to a unique solution when* $k \to \infty$ *for any* $l \in \mathbb{R}$ $\mathbf{W} \in \mathbb{R}^{d\times d}, \tilde{\mathbf{A}} \in \mathbb{R}^{n\times n}, \mathbf{B} \in \mathbb{R}^{d\times n}, n \in \mathbb{N}, \epsilon > 0$*, any Lipschitz continuous activation function* ϕ*.*

We refer to the supplementary material for all proofs.

Practically, for either Eq. (7) or Eq. (8) processing starts at the furthest distant nodes, for which k needs to be known. This a challenge, because we do

not know beforehand when our convergence criterion is satisfied. As our interest is in the limit state $\mathbf{G}^{(0;k)}$ when $k \to \infty$, we make use of a convergence threshold ϵ to identify the number of required iterations

$$k = \min\{M \mid \mathbf{G}^{(0;k)} - \mathbf{G}^{(0;k+1)} < \epsilon\} \tag{9}$$

until our solution is close to the limit and iterating further has negligible impact. Because even the initial iteration $\mathbf{G}^{(k-1;1)} \neq \mathbf{G}^{(k;1)}$ is different, intermediate results from $\mathbf{G}^{(0;k-1)}$ cannot be reused for computing $\mathbf{G}^{(0;k)}$. A full recalculation is needed, which requires $k!$ iterations.

Instead, we take a different route to determine k. Determining at which iteration the difference of expanding further on the graph becomes negligible is approximately the same as determining how far the influence of nodes in the graph reach using our message passing scheme. To determining this, we ignore the teleportation term and estimate the influence of the initial state $f_\theta(\mathbf{U})$ on the result of l iterations $\mathbf{G}^{(0;l)}$ with the equation

$$\mathbf{E}^{(l+1)} = \phi(\alpha_{l+1}\tilde{\mathbf{A}}\mathbf{E}^{(l)}\mathbf{W}) \tag{10}$$

by setting $\mathbf{E}^{(0)} = f_\theta(\mathbf{U})$. Unlike in Eq. (7) where we reversed the equation, the result $\mathbf{E}^{(m)}$ is equal for $\alpha_l = \frac{1}{1+(m-l-1)\epsilon}$ and $\alpha_l = \frac{1}{1+l\epsilon}$ when we use ReLU as ϕ. Note, that we start with α_{l+1} because this is the first α that is applied to the teleportation matrix $f_\theta(\mathbf{U})$. Eq. (10) converges for similar reasons as Eq. (8), only towards $\mathbf{0} \in 0^{d\times n}$, which we proof with the following theorem.

Theorem 2. *The equation* $\mathbf{E}^{(l+1)} = \phi\left(\alpha_l\tilde{\mathbf{A}}\mathbf{E}^{(l)}\mathbf{W}\right)$ *with* $\alpha_l = \frac{1}{1+l\epsilon}$ *converges to* $\mathbf{0} \in 0^{d\times n}$ *for any* $\mathbf{W} \in \mathbb{R}^{d\times d}, \tilde{\mathbf{A}} \in \mathbb{R}^{n\times n}, l \in \mathbb{N}, \epsilon > 0$*, any initial* $\mathbf{E}^{(0)}$ *and the ReLU activation function* ϕ*. The solution can be obtained by iterating the equation. For any fixed number of iterations* m*, the solution* $\mathbf{E}^{(m)}$ *is the same as using* $\alpha_l = \frac{1}{1+(m-l-1)\epsilon}$*.*

Since we can evaluate Eq. (10) directly by iterating until our convergence criterion is met, we find the required number of steps with

$$k' = \min\{l \mid \mathbf{E}^{(l)} < \epsilon\}\,. \tag{11}$$

At this point the effect of the initial state on nodes of distance l is negligible. We use k' as k in Eq. (8) and execute the forward pass. The result $\mathbf{H}^{(k')}$ gets passed onto the next operation in our model, as with other graph convolutions.

4.1 Efficient Optimization

While we do not use Eq. (10) for gradient computation, even tracking only Eq. (7) with autograd software would still lead to memory consumption that is linear in the number of layers. Similarly as in the forward pass, the gradients converge to $\mathbf{0}$ for distant nodes. We iterate the computation of gradients until the same convergence criterion is met. Because of faster converge in the backward pass,

this allows the optimization of the model with reduced memory consumption. We will further limit the iterations to guarantee constant complexity, independently of the number of iterations performed.

For calculating derivatives we use the reformulation from Eq. (8). We introduce additional notation and set $\hat{\mathbf{Y}} = f_\theta(\mathbf{G}^{(0)})$ as the output of our model, $\mathbf{Y}$ as the target, and $\mathcal{L} = l(\hat{\mathbf{Y}}, \mathbf{Y})$ to be our loss calculated with any differentiable loss function l. We are interested in the partial derivatives of our loss $\mathcal{L}$ with respect to the parameters $\mathbf{W}$ and the input state $\mathbf{B}$. We let autograd solve the derivation $\frac{\partial L}{\partial \mathbf{G}^{(0)}}$ and apply the chain rule for other partial derivatives. To simplify our notation for the application of the chain rule, we further define $\mathbf{Z}^{(n)} = \alpha_n \tilde{\mathbf{A}} \mathbf{G}^{(n+1)} \mathbf{W} + \mathbf{B}$ and $\mathbf{G}^{(n)} = \phi(\mathbf{Z}^{(n)})$. All further partial derivatives can be calculated by using the following equations:

$$\frac{\partial L}{\partial \mathbf{G}^{(n)}} = \alpha_n \tilde{\mathbf{A}}^T \frac{\partial L}{\partial \mathbf{Z}^{(n-1)}} \mathbf{W}^T \tag{12}$$

$$\frac{\partial L}{\partial \mathbf{Z}^{(n)}} = \phi' \left(\alpha_n \tilde{\mathbf{A}} \mathbf{G}^{(n+1)} \mathbf{W} + \mathbf{B} \right) \odot \frac{\partial L}{\partial \mathbf{G}^{(n)}} \tag{13}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{n=0}^{\infty} \alpha_n \left(\tilde{\mathbf{A}} \mathbf{G}^{(n+1)} \right)^T \frac{\partial L}{\partial \mathbf{Z}^{(n)}} \tag{14}$$

$$\frac{\partial L}{\partial \mathbf{B}} = \sum_{n=0}^{\infty} \frac{\partial L}{\partial \mathbf{Z}^{(n)}} \tag{15}$$

The partial derivatives $\frac{\partial L}{\partial \mathbf{W}}$ and $\frac{\partial L}{\partial \mathbf{B}}$ converge for similar reasons as before, so we iterate Eq. (14) and Eq. (15) until our convergence criterion is met. The convergence rate turns out to be much faster than the convergence rate of the forward pass, which results in reduced practical memory consumption. To theoretically guarantee constant memory consumption, we only consider a fixed amount n of elements in the sum, similarly to the effectiveness of Truncated Backpropagation Through Time (TBPTT) [33] for sequential data. This also reduces the time complexity of the backward pass to be constant. We found this restriction to have negligible impact even for small values of N. Depending on available memory, we either store the intermediate results for gradient computation or use gradient checkpointing [5] with a few additional forward iterations. Note, that for the backward step the solutions of $\mathbf{G}^{(0)}, \ldots, \mathbf{G}^{(n)}$ are needed explicitly. We assure the convergence of all used $\mathbf{G}^{(i)}$ by using the fact that $\mathbf{G}^{(n;k)} < \mathbf{G}^{(0;k)}$ and therefore compute $\mathbf{G}^{(0;k+n)}$ instead of $\mathbf{G}^{(0;k)}$ in the initial forward pass.

5 Experiments

We evaluate the effectiveness of PPRGNN on various public benchmark datasets and compare these results to popular methods and other infinite-depth approaches. We evaluate our approach on an inductive node classification task, a transductive node classification task, and five graph classification tasks. Table 1

Table 1: Properties of datasets used for evaluation.

Dataset	# of Graphs	Avg. # of nodes	# of classes
Amazon	1	334 863	58
PPI	22	2373	121
MUTAG	188	17.9	2
PTC	344	25.5	2
COX2	467	41.2	2
PROTEINS	1113	39.1	2
NCI1	4110	29.8	2

Table 2: Micro-F_1-Scores for PPI.

Method	Micro-F_1-Score
MLP	46.2
GCN	59.2
SSE	83.6
GAT	97.3
IGNN	97.6
APPNP	44.8
PPRGNN	**98.9**

shows detailed properties of all used datasets. We closely follow the experimental settings of IGNN [16] and inherit their architectures, only replacing their formulation directly with ours. Thus the number of parameters is the same, so the comparison with their approach is the most meaningful for us. We apply APPNP to all tasks using their setup with 10 iterations. We further compare PPRGNN with a series of other popular methods for the tasks of node classification and graph classification and reuse the results reported in [16]. Due to the increased modeling capacity, we use gradient clipping and weight decay. Additionally, we tune ϵ, the learning rate and whether self-loops are taken into account for $\tilde{\mathbf{A}}$ for the three different tasks. We set $n = 5$ for the backward pass. We reduce the learning rate when the training loss plateaus. All experiments are executed on a single Nvidia Tesla P100.

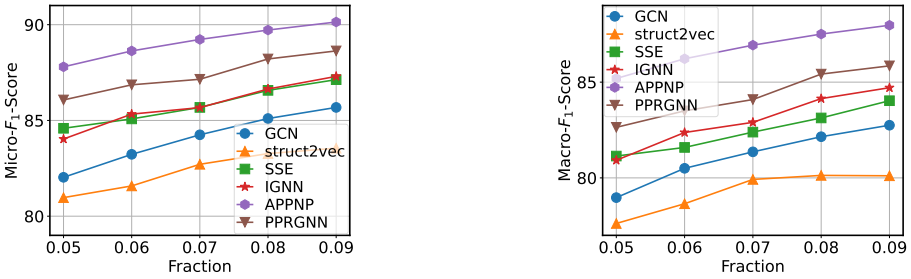
PPI We consider the task of role prediction of proteins in graphs of protein-protein interactions (PPI) [17]. In this inductive node classification task, we use 18 graphs for training our model, 2 for validation, and 2 for testing. Our data split matches that in previous work [17]. As taken over from IGNN, our model consists of 5 stacked layers, each iterating until convergence. We set $\epsilon = 0.25$ and find self-loops detrimental to our approach. In addition to IGNN and APPNP, reference methods are a MLP, GCN [21], SSE [7], GAT [35]. The Micro-F_1-Scores for all considered approaches are presented in Table 2. PPRGNN outperforms all of these approaches and reduces the error by more than 50% compared to IGNN. Our trained PPRGNN uses a total of 82 message passing iterations in testing, while GCN and GAT use a maximum of 3 iterations. We also compare the time needed for PPRGNN to surpass the Micro-F_1-Score of IGNN in Figure 3. PPRGNN needs fewer iterations and also takes less time per Epoch. This comes from accurate gradient descent steps without projection and being able to adjust the speed of convergence with ϵ. We find APPNP to underfit the data due to the limited modeling capacity, even when the number of parameters uses all available memory.

Table 3: Time and epochs needed until PPRGNN surpasses the best epoch of IGNN on the validation set.

Dataset	Method	Epochs	Avg. Time per Epoch	Total Time
Amazon (0.05)	IGNN	872	14s	3h 21m
	PPRGNN	**175**	**11s**	**32m**
PPI	IGNN	58	26s	25m
	PPRGNN	**47**	**18s**	**14m**

Amazon To test the scalability of our approach, we apply it to the Amazon product co-purchasing network data set [40]. Following the settings from [7], product types with at least 5000 different products are selected. This results in 334 863 nodes representing products and 925 872 edges representing products that have been purchased together. The task is to predict the correct product type for each node. Nodes do not have any features, so predictions are made solely based on the graph structure. We use the same data split as [7], leading to a fraction of nodes used for training varying between 5% and 9%. A fixed set consisting of 10% of the nodes is used for training, the rest for validation. The main challenge of this task is not the prediction complexity but rather dealing with the sparsity of the available labels. Our architecture consists of our PPRGNN layer combined with a linear operation before and after. We compare our results with APPNP and reuse the result found in [7, 16] for IGNN [16], SSE [7], struct2vec [29] and GCN [21]. Micro-F_1-Scores and Macro-F_1-Scores are shown in Figure 1 for varying fractions of labels used. While we outperform IGNN, SSE, struct2vec and GCN across all settings by at least 1%, APPNP performs the best. We find the low modeling capacity of APPNP to be better suited for generalizing in this scenario. Again, we compare the execution time needed for PPRGNN to outperform IGNN (Figure 3) and find PPRGNN to converge in fewer epochs, with each epoch executing faster. This further adds to our point of benefiting from applying gradient descent without projection and controlling convergence speed.

Graph Classification We now evaluate our approach for the task of graph classification on five open graph datasets, namely MUTAG, PTC, COX2, PROTEINS, and NCI1. Following the same setup from previous work, we conduct a 10-fold cross-validation for each dataset and report the mean and standard deviation of the folds validation sets. We integrate our formulation with $\epsilon = 1$ into the architecture from IGNN, consisting of 3 stacked iterations until convergence. For regularization, we add a weight decay of 1e−6 and gradient clipping of 25 to all datasets. For NCI1, we find removing self-loops to be helpful for generalization. For comparison, we use several graph kernel approaches (GK [32], RW [12], WL [31]) and GNN approaches (DGCNN [43], GCN [21], GIN [37]) in addition to IGNN and APPNP. We reuse reported results from [16]. PPRGNN outperforms all other approaches across 4 out of 5 datasets by at least 1% and is the second-



(a) Micro-F_1-Scores.

(b) Macro-F_1-Scores.

Fig. 1: Comparison of results on the Amazon dataset. The fraction of labels used for optimization varies between 0.05 and 0.09.

Table 4: Comparison of accuracies on various graph classification tasks.

Dataset	MUTAG	PTC	COX2	PROTEINS	NCI1
GK	81.4 ±1.7	55.7 ±0.5	-	71.4 ±0.3	62.5 ±0.3
RW	79.2 ±2.1	55.9 ±0.3	-	59.6 ±0.1	-
WL	84.1 ±1.9	58.0 ±2.5	83.2 ±0.2	74.7 ±0.5	**84.5** ±0.5
DGCNN	85.8	58.6	-	75.5	74.4
GCN	85.6 ±5.8	64.2 ±4.3	-	76.0 ±3.2	80.2 ±2.0
GIN	89.0 ±6.0	63.7 ±8.2	-	75.9 ±3.8	82.7 ±1.6
IGNN	89.3 ±6.7	70.1 ±5.6	86.9 ±4.0	77.7 ±3.4	80.5 ±1.9
APPNP	87.7 ±8.6	64.5 ±5.1	82.2 ±5.5	78.7 ±4.8	65.9 ±2.7
PPRGNN	**90.4** ±7.2	**75.0** ±5.7	**89.1** ±3.9	**80.2** ±3.2	83.5 ±1.5

best performing model with competitive accuracy on the fifth dataset. Despite using the same $\epsilon = 1$ across all experiments, the effective depth ranges from 22 to 41 for different datasets. Depth is adaptive even within individual datasets, depending on learned parameters, the examined graph and present node features. These results further demonstrate the effectiveness of our approach and the potential to create deeper models on a wide variety of datasets.

6 Conclusion

We introduced PPRGNN, a reformulation of MPNNs based on personalized PageRank that assures localization and prevents over-smoothing of node features even when using infinitely many layers. Theoretically based on the personalized version of PageRank which allows teleporting back to the initial state, we adopt this idea for MPNNs, specifically for the basic type GCNs [21]. Starting from the classic algorithm, we follow intuitive steps to introduce learnable parameters and still converge to a limit distribution. Compared to previous infinite-depth

GNNs, our approach has a higher modeling capacity as we do not place any constraints. Our empirical evaluation on tasks for graph classification, and inductive and transductive node classification confirm our theoretical base. Against regular GCNs that have no way to teleport back to the initial state, we find large improvements across all datasets. We even outperform other comparable approaches, including previous infinite-depth models, across almost all datasets by decent margins. Despite the theoretical infinite-depth, we introduced a path for efficient optimization, running linearly in the number of layers and only using constant memory. Our formulation allows controlling the convergence rate, leading to considerable improvements in experimental execution time compared to IGNN, a previous infinite-depth model. While we show that our formulation allows infinitely many layers, even fixed sized models should benefit from adopting our idea. Our approach is directly applicable to other types of MPNNs, for which our proofs of convergence should hold.

Acknowledgements Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) - project number 124020371 - within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", DFG project number 124020371, SFB project B4. The authors are funded by the German Federal Ministry of Education and Research (BMBF) in the course of the 6GEM research hub under grant number 16KISK038.

References

1. Abu-El-Haija, S., Perozzi, B., Al-Rfou, R., Alemi, A.A.: Watch your step: Learning node embeddings via graph attention. Advances in neural information processing systems **31** (2018)
2. Bai, S., Kolter, J.Z., Koltun, V.: Deep equilibrium models. Advances in Neural Information Processing Systems **32** (2019)
3. Berman, A., Plemmons, R.J.: Nonnegative matrices in the mathematical sciences. SIAM (1994)
4. Chen, M., Wei, Z., Huang, Z., Ding, B., Li, Y.: Simple and deep graph convolutional networks. In: International Conference on Machine Learning. pp. 1725–1735. PMLR (2020)
5. Chen, T., Xu, B., Zhang, C., Guestrin, C.: Training deep nets with sublinear memory cost. arXiv preprint arXiv:1604.06174 (2016)
6. Chiang, W.L., Liu, X., Si, S., Li, Y., Bengio, S., Hsieh, C.J.: Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 257–266 (2019)
7. Dai, H., Kozareva, Z., Dai, B., Smola, A., Song, L.: Learning steady-states of iterative algorithms over graphs. In: International conference on machine learning. pp. 1106–1114. PMLR (2018)
8. Derrow-Pinion, A., She, J., Wong, D., Lange, O., Hester, T., Perez, L., Nunkesser, M., Lee, S., Guo, X., Wiltshire, B., et al.: Eta prediction with graph neural networks in google maps. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management. pp. 3767–3776 (2021)

9. Duchi, J., Shalev-Shwartz, S., Singer, Y., Chandra, T.: Efficient projections onto the l 1-ball for learning in high dimensions. In: Proceedings of the 25th international conference on Machine learning. pp. 272–279 (2008)
10. El Ghaoui, L., Gu, F., Travacca, B., Askari, A., Tsai, A.: Implicit deep learning. SIAM Journal on Mathematics of Data Science **3**(3), 930–958 (2021)
11. Gallicchio, C., Micheli, A.: Fast and deep graph neural networks. In: Proceedings of the AAAI conference on artificial intelligence. vol. 34, pp. 3898–3905 (2020)
12. Gärtner, T., Flach, P., Wrobel, S.: On graph kernels: Hardness results and efficient alternatives. In: Learning theory and kernel machines, pp. 129–143. Springer (2003)
13. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: International conference on machine learning. pp. 1263–1272. PMLR (2017)
14. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. In: Proceedings. 2005 IEEE international joint conference on neural networks. vol. 2, pp. 729–734 (2005)
15. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864 (2016)
16. Gu, F., Chang, H., Zhu, W., Sojoudi, S., El Ghaoui, L.: Implicit graph neural networks. Advances in Neural Information Processing Systems **33**, 11984–11995 (2020)
17. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. Advances in neural information processing systems **30** (2017)
18. Hammond, D.K., Vandergheynst, P., Gribonval, R.: Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis **30**(2), 129–150 (2011)
19. Hardt, M., Ma, T.: Identity matters in deep learning. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings (2017)
20. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
21. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
22. Klicpera, J., Bojchevski, A., Günnemann, S.: Predict then propagate: Graph neural networks meet personalized pagerank. arXiv preprint arXiv:1810.05997 (2018)
23. Klicpera, J., Weißenberger, S., Günnemann, S.: Diffusion improves graph learning. In: Conference on Neural Information Processing Systems (NeurIPS) (2019)
24. Li, Q., Han, Z., Wu, X.M.: Deeper insights into graph convolutional networks for semi-supervised learning. In: Thirty-Second AAAI conference on artificial intelligence (2018)
25. Oono, K., Suzuki, T.: Graph neural networks exponentially lose expressive power for node classification. In: International Conference on Learning Representations (2020)
26. Otte, E., Rousseau, R.: Social network analysis: a powerful strategy, also for the information sciences. Journal of information Science **28**(6), 441–453 (2002)
27. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab (1999)
28. Pavlopoulos, G.A., Secrier, M., Moschopoulos, C.N., Soldatos, T.G., Kossida, S., Aerts, J., Schneider, R., Bagos, P.G.: Using graph theory to analyze biological networks. BioData mining **4**(1), 1–27 (2011)

29. Ribeiro, L.F., Saverese, P.H., Figueiredo, D.R.: struc2vec: Learning node representations from structural identity. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 385–394 (2017)
30. Rong, Y., Huang, W., Xu, T., Huang, J.: Dropedge: Towards deep graph convolutional networks on node classification. In: International Conference on Learning Representations (2020)
31. Shervashidze, N., Schweitzer, P., Van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. Journal of Machine Learning Research **12**(9) (2011)
32. Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., Borgwardt, K.: Efficient graphlet kernels for large graph comparison. In: Artificial intelligence and statistics. pp. 488–495. PMLR (2009)
33. Sutskever, I.: Training recurrent neural networks. University of Toronto Toronto, ON, Canada (2013)
34. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
35. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. International Conference on Learning Representations (2018)
36. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems **32**(1), 4–24 (2020)
37. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? arXiv preprint arXiv:1810.00826 (2018)
38. Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.i., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. In: International Conference on Machine Learning. pp. 5453–5462. PMLR (2018)
39. Yanardag, P., Vishwanathan, S.: Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. pp. 1365–1374 (2015)
40. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. Knowledge and Information Systems **42**(1), 181–213 (2015)
41. Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. pp. 974–983 (2018)
42. Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.: Graphsaint: Graph sampling based inductive learning method. In: International Conference on Learning Representations (2020)
43. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: Thirty-second AAAI conference on artificial intelligence (2018)
44. Zhao, L., Akoglu, L.: Pairnorm: Tackling oversmoothing in gnns. In: International Conference on Learning Representations (2020)
45. Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., Koutra, D.: Beyond homophily in graph neural networks: Current limitations and effective designs. Advances in Neural Information Processing Systems **33**, 7793–7804 (2020)