Curvature Graph Network

Anonymous authors
Paper under double-blind review

Abstract
Graph-structured data is prevalent in many domains. Despite the widely celebrated success of deep neural networks, their power in graph-structured data is yet to be fully explored. We propose a novel network architecture that incorporates advanced graph structural information. In particular, we leverage discrete graph curvature, which measures how the neighborhoods of a pair of nodes are structurally related. The curvature of an edge \((x, y)\) defines the distance taken to travel from neighbors of \(x\) to neighbors of \(y\), compared with the length of edge \((x, y)\). It is a much more descriptive structural measure compared to previously ones that only focus on node specific attributes or limited topological information such as degree. Our curvature graph convolution network outperforms state-of-the-art on various synthetic and real-world graphs, especially the larger and denser ones.

1 Introduction
Despite the huge success of deep neural networks, it remains challenging to fully exploit their power on graph-structured data, i.e., data whose underlying structure is a graph, e.g., a social network, a telecommunication network, a biological network and a brain connectome. Inspired by the power of convolution on image data, convolutional networks have been proposed for graph-structured data. Existing works can be roughly divided into two categories, depending on whether convolution is applied to the spectral or spatial domain. Spectral approaches (Bruna et al., 2013; Defferrard et al., 2016; Henaff et al., 2015; Veličković et al., 2017) apply convolution to eigen-decomposed graph Laplacians and are generally efficient in both computation and memory. However, the learned convolution filters are graph-specific and cannot generalize to different graphs.

Spatial approaches execute “convolution” directly on the graph and operate on the neighborhood as defined by the graph topology. A spatial method iteratively updates the representation of each graph node by aggregating representations from its neighbors, i.e., adjacent nodes (Xu et al., 2018). Nonlinear transformations are applied to the representation passed from one node to another, called a message. These transformations have the same input/output dimension, i.e., the dimension of the node representation. They can be shared and learned across different nodes and even different graphs.

For spatial approaches, it is important to incorporate local structural information of the graph. Node degree has been used to reparametrize the nonlinear transformation of messages (Monti et al., 2017) or as an additional node feature (Hamilton et al., 2017). However, node degree is fairly limited; there can be different graph topologies with the same degree distribution. The limitation is illustrated in Figure 1. Nodes \(x\) and \(y\) have the same degree in three significantly different graphs: a tree, a grid graph and a clique. To effectively make use of graph structural knowledge, one would need a feature with more discriminative power; one that can distinguish these three scenarios in Figure 1.

In this paper, we propose a novel graph neural network that exploits advanced structural information. Notice that node degree only describes the number of neighbors of each node, but does not say how these neighbors are connected among themselves. We seek to use structural information characterizing how neighborhoods of a pair of nodes relate to each other. In Figure 1 the neighborhoods of \(x\) and \(y\) are well separated in a tree. In a grid graph, the two neighborhoods are within a parallel shift of each other. In a clique, they completely overlap. To quantify such pairwise structural information, we draw inspiration from recent study of graph curvature (Ollivier 2009; Lin et al., 2011; Weber et al., 2016).

Similar to the curvature in the continuous domain, e.g., the Ricci curvature of a Riemannian manifold, the discrete graph curvature measures how the geometry of a pair of neighborhoods deviates from a “flat” case, namely, the case of a grid graph. There are several definitions of discrete curvature for graphs. The most notable one is Ollivier’s Ricci curvature (Ollivier 2009). The edges of a (infinite)
Figure 1: Illustration of structural information. In all three graphs, the degrees of $x$ and $y$ are the same. However, the Ricci curvature of the edge $(x, y)$ is negative, zero, and positive, respectively. All edges have weight 1.

The grid graph have zero curvature. The curvature of an edge $(x, y)$ in a tree is negative and is positive in a complete graph. Intuitively, the graph curvature measures how well two neighborhoods are connected and/or overlap with each other. Such information is related to how information propagates in the neighborhood, and should be leveraged by a graph convolutional network.

We propose Curvature Graph Network (CurvGN), the first graph convolutional network built on advanced graph curvature information. In particular, we propose a novel network architecture that efficiently computes graph curvature and fully leverages such information in graph convolution. Using curvature information, CurvGN better adapts to different local structural scenarios and filter messages passed between nodes differently. Notice that the curvature information captures how easy information flows between the nodes. Within a well-connected community, neighborhoods of adjacent nodes have large overlap and many shortcuts. The corresponding curvature is positive and passing information between the nodes is easy. For edges bridging two clusters/cliques, the curvature is negative and information is hard to pass. A key to our success is that we choose to be agnostic on whether the curvature information should be used to block or accelerate the messages in graph convolution. We exploit the curvature in a data-driven manner and learn how to use it to reweight different channels of the message.

To further investigate how curvature information affects graph convolution, we carried out extensive experiments with various synthetic graphs and real-world graphs. Our synthetic data are generated according to various well-established graph models, e.g., stochastic block model (Decelle et al., 2011), Watts–Strogatz network (Watts & Strogatz, 1998), Newman–Watts network (Newman & Watts, 1999) and Kleinberg’s navigable small world graph (Kleinberg, 2000). On these data, CurvGN outperforms vanilla graph network and networks using node degree information and self attention, demonstrating the benefit of curvature information in graph convolution. Such benefit is more apparent as the graph size increases. We hypothesize that graph convolution alone can adapt to any graph topology, at the cost of more convolutional layers and more training data. This is corroborated by our experiments on real-world graph. CurvGN outperforms state-of-the-art graph neural networks, especially on larger and denser graphs, which tend to have a large variation of local structures.

The success of CurvGN demonstrates how theoretical insights inspire better practical solutions. It encourages us to continue the endeavor in applying principles mathematics and theory in successful deployment of deep learning.

2 RELATED WORK

We briefly summarize previous works on graph convolution. In early works (Frasconi et al., 1998; Sperduti & Starita, 1997), recursive neural networks were applied on data whose underlying structures are directed acyclic graphs. In Graph Neural Networks (GNNs) (Gori et al., 2005; Scarselli et al., 2009), the recursive network framework was extended to general graphs. Li et al. (2015) introduced gated recurrent units into the framework in order to improve the performance. Since Convolutional Neural Networks (CNNs) have demonstrated strong performance in grid-like-structured data, various methods have been proposed to implement “convolution” on graph-structured data. The efforts can be roughly divided into spectral approaches and spatial approaches. Below we will review both categories in details.

Spectral approaches. Bruna et al. (2013) transformed the graph convolution into spectral domain multiplication by graph Fourier transform. This method is expensive due to the matrix eigen-decomposition. Furthermore, it cannot create spatially localized filters as in CNNs. Henaff et al. (2015) applied smooth coefficients on spectral filters to make them spatially localized. Defferrard et al. (2016) used Chebyshev expansion of the graph Laplacian to approximate the filters as a $k$-polynomial.
function. Kipf & Welling (2016) simplified those methods by reducing the polynomials to degree 1. Their method is essentially filtering the graph with a kernel whose receptive field is the 1-hop neighborhood of each node. The common limitation of spectral approaches is that the convolution filters depend on the Laplacian of each specific graph.

Spatial approaches. The main challenge of spatial approaches is to design an operator which applies to neighborhoods with different topology and still maintains shared filters. Monti et al. (2017) introduced a mixture model CNN (MoNet) that maps graph neighborhood into spatial neighborhood (with pseudo-coordinates) for spatial convolution. Hamilton et al. (2017) proposed GraphSAGE that samples fixed size neighbors and aggregates their representations. Velickovic et al. (2017) proposed Graph Attention Network (GAT), which uses self-attention mechanism to reweigh graph convolution. Recently, there have been other methods which studied graph neural networks from different perspectives, such as pooling (Gao & Ji, 2019) Ying et al. (2018).

Discrete graph curvature. Different proposals for discrete graph curvature have been introduced in recent years, including Ollivier’s Ricci curvature and Forman curvature (Ollivier 2009, Lin et al. 2011, Forman 2003). We focus on Ollivier’s Ricci curvature as it is more geometric in nature. Meanwhile, Forman’s definition of discrete curvature (Forman 2003) is more combinatorial and is faster to compute. Both curvatures have been applied to real-world graphs. Ni et al. (2015) showed that Ollivier’s Ricci curvature can be used to identify backbone edges of an Internet AS graph and is closely related to network vulnerability. In Ni et al. (2018) it was shown that using Ollivier’s Ricci flow, one can define a new metric that is more robust for network matching. Forman curvature is shown to have similar effect (Weber et al., 2017, 2016). To the best of our knowledge, graph curvature has not been used in graph neural networks.

3 CURVATURE GRAPH NETWORK

We first formulate the node label prediction problem of a graph, and explain the mechanism of a Graph Neural Network (GNN). Suppose we have an undirected graph \( G = (V, E) \) with features on the vertices \( H = (h_1, h_2, \ldots, h_n) \), \( h_i \in \mathbb{R}^F \). Here \( n = |V| \) is the number of nodes in the graph and \( F \) is the feature dimension of each node. Given labels of some nodes in \( V \), we would like to predict the labels of the remaining nodes. A GNN iteratively updates the graph \( G \)’s node representation and eventually predicts node labels. A GNN consists of multiple hidden layers that update node representation from lower level node representation \( H^t = \mathbb{R}^{n \times F_t} \) to high level representation \( H^{t+1} = \mathbb{R}^{n \times F_{t+1}} \). In particular, \( H^0 \) is the input feature, \( H \). Node representations of the last layer, \( H^T \), are fed to a fully connected layer or a linear classifier to predict node labels. The layers and their representations are illustrated in the top of Figure 2.

Now we explain how to construct hidden layers that update node representations from \( H^t \) to \( H^{t+1} \). We focus on spatial approaches and treat the convolution as a message passing scheme. The \((t+1)\)-th representation of node \( x \) is computed by aggregating messages passed from \( x \)’s neighbors. We also include the message from \( x \) to itself. There are several aggregation methods, such as mean, max and sum. We choose summation as it is a commonly used aggregation method (Kipf & Welling, 2016, Velickovic et al. 2017, Xu et al. 2018). Denote by \( \mathcal{N}(x) = \mathcal{N}(x) \cup \{x\} \) the neighborhood of \( x \) including itself. We have \( h_x^{t+1} = \sigma_t \left( \sum_{y \in \mathcal{N}(x)} M_{y \rightarrow x}^t h_y^t \right) \), in which \( \sigma_t \) is the non-linear transformation. A message passed from \( y \) to \( x \) is a linear transformation of \( y \)’s representation. We also introduce a weight \( \tau_{xy}^t \) whose purpose will be clear later. Formally, we have \( M_{y \rightarrow x}^t = \tau_{xy}^t W^t h_y^t \), in which \( W^t \) is the linear transformation matrix learned in training. Formally, we have the representation updating equation

\[
 h_x^{t+1} = \sigma_t \left( \sum_{y \in \mathcal{N}(x)} \tau_{xy}^t W^t h_y^t \right) \quad (3.1)
\]

It is crucial to obtain suitable reweighting parameter \( \tau_{xy}^t \) since it is directly affecting how neighboring node information are passed to the node \( x \). Some papers use node degree information as \( \tau_{xy}^t \) (Kipf & Welling, 2016, Monti et al., 2017) and other work uses joint node features to compute the self attention as \( \tau_{xy}^t \) (Velickovic et al. 2017). We propose to use more advanced structural information, i.e., the Ricci curvature, to compute \( \tau_{xy}^t \). It is also known that the reweighting parameter \( \tau_{xy}^t \) is not necessarily a scalar. It can also be anything between a scalar and a \( F^t \times F^t \) matrix. In fact, we choose \( F^t \) later on because it has more expressive power than a scalar and it is easier to train than a matrix.
Wasserstein distance (or Earth Mover distance) which finds the optimal mass-preserving transportation $M$ where $\alpha$ is a parameter within $[0,1]$. It is to keep probability mass of $\alpha$ at node $x$ itself and distribute the rest uniformly over the neighborhood. To compute the Wasserstein distance $W(m_x^\alpha, m_y^\alpha)$ between the probability measures around two end points $x, y$ of the edge $(x, y)$, the optimal transportation plan can be solved by the following linear programming:

$$\min_M \sum_{i,j} d(x_i, y_j) M(x_i, y_j) \text{ s.t. } \sum_j M(x_i, y_j) = m_x^\alpha(x_i), \forall i; \sum_i M(x_i, y_j) = m_y^\alpha(y_j), \forall j \quad (3.2)$$

where $M(x_i, y_j)$ is the amount of probability mass transported from node $x_i$ to $y_j$ along the shortest path with length $d(x_i, y_j)$. In the language of graph theory, if the Ollivier-Ricci curvature is negative
We choose to be agnostic on how the knowledge of edge curvature should be used. We resort to the assumption that different communities tend to have different representations/labels, a message should be reweighed according to the curvature of its edges. Next we expand the mapping to a multi-valued version, to incorporate more flexibility.

The scalar weight generated using curvature is not necessarily the same for different channels. To improve the expressing power of the message, i.e., $\tau_{xy}^t$ in Equation (3.1), we first explain how the mapping is learned end-to-end (CurvGN-1). Next we expand the mapping to a multi-valued version, to incorporate more flexibility in the model (CurvGN-n).

### CurvGN-1

As mentioned before, $\tau_{xy}^t$ can be anything between a scalar and a $F^t \times F^t$ matrix. We first assume $\tau_{xy}^t$ is a scalar. Then the mapping function can be defined as:

$$f^t : \kappa_{xy} \rightarrow \tau_{xy}^t$$

(3.3)

We create a multi-layer perceptron (MLP) to approximate the mapping function $f^t$ since MLP is proved to be a universal approximation machine and can be easily incorporated into our GNN model for end-to-end training. Denote the MLP at the $t$-th layer as $\text{MLP}^t$. As summation is used as the aggregation function in Equation (3.1), the messages may accumulate to an arbitrarily large value. To prevent a numerical explosion, we apply a softmax function, $S^t$, to $\text{MLP}^t(\kappa_{xy})$ of all neighbors of $x$ including itself, $y \in \mathcal{N}(x)$ nodes.

$$\tau_{xy}^t = S^t(\text{MLP}^t(\kappa_{xy}))$$

(3.4)

Figure 2 bottom shows how the MLP transforms a curvature and uses it to reweigh messages.

### CurvGN-n

Messages $M^t_{y \rightarrow x}$ are usually multi-channeled. In particular, they are $F^{t+1}$-dimensional. The scalar weight generated using curvature is not necessarily the same for different channels. To improve the expressing power of $\tau_{xy}^t$, we create a similar mapping function as $f^t$ in Equation (3.3).

But the new mapping generates a reweighing vector $\mathcal{T}_{xy}^t \in \mathbb{R}^{F^{t+1}}$. In other words, we learn to reweigh different message channels differently. In principle, we can even learn an $\mathbb{R}^{F^{t+1}} \times \mathbb{R}^{F^{t+1}}$ to reweigh the message. However, a vector has significantly less parameters to train and is found to be sufficient in practice.

Using the same strategy as CurvGN-1, the vector $\mathcal{T}_{xy}^t$ is calculated by applying a $\text{MLP}^t$ with $F^{t+1}$ outputs. Then, we apply a channel-wise softmax function, $S^t$, that normalizes the MLP outputs separately on each message channel: $\mathcal{T}_{xy}^t = S^t(\text{MLP}^t(\kappa_{xy}))$

Substituting $\mathcal{T}_{xy}^t$ into Equation (3.1), we have the convolution of CurvGN-n:

$$h_{x}^{t+1} = \sigma_t \left( \sum_{y \in \mathcal{N}(x)} \text{diag}(\mathcal{T}_{xy}^t) W^t h_y^t \right)$$

(3.5)

Here $\text{diag}(\mathcal{T}_{xy}^t)$ is a matrix whose diagonal entries are entries of $\mathcal{T}_{xy}^t$. For details of $\text{MLP}^t$, please refer to Appendix A.1

### Design details of the network

In practice, we use a two-convolutional-layer CurvGN model. The first layer is a linear transform layer that produces an output feature vector paired with a three layer MLP that computes reweighing vector. The output feature is pushed into an exponential linear unit layer to add non-linearity. The second layer is for classification, with the same structure as the first layer except that the output feature is now at length of class number. The hyperparameters are similar to GAT implemented in [Veličković et al. 2017]. For synthetic experiments, the hidden layer output is reduced to 8 dimensions.
4 EXPERIMENTS
We evaluate our method on both synthetic and real-world graphs. Our method outperforms the state-of-the-art methods, especially on larger and denser graphs, which tend to have heterogeneous topology. In addition to proving the prediction power, we use different graph theoretical models in synthetic experiments and different parameter settings to gain insights of how curvature information helps graph convolution. We focus on node classification task, while our method easily generalizes to graph classification task.

4.1 SYNTHETIC EXPERIMENTS ON DIFFERENT GRAPH THEORETICAL MODELS
We generate synthetic data using different graph theoretical models. We start with the Stochastic Block Model (SBM) (Holland et al., 1983), which assumes a partition of the graph into communities. We create random graphs, each with 1000 nodes and equally partition the node set into five disjoint communities. Nodes in the same community have the same class label. Edges are randomly sampled with an intra-community probability, \( p \), if they are within the same community. They are sampled with an inter-community probability, \( q \), if they are cross-community, e.g., bridges connecting different communities. We randomly create 100 graphs with \( p \) ranging in \{0.05, 0.07, \ldots, 0.23\} and \( q \) ranging in \{0.0, 0.005, \ldots, 0.045\}. For each generated graph, we randomly select 400 nodes as training set, another 400 nodes as validation set and the remaining 200 nodes as test set. We assign each node with a randomly generated feature of dimension 10 and use them as uninformative input of our CurvGN.

Baselines. We use five different methods. They include two popular state-of-the-arts: GCN (Kipf & Welling, 2016) and GAT (Veličković et al., 2017). For these methods, we use the exact same setting as for Cora dataset mentioned in Veličković et al. (2017); Kipf & Welling (2016), except that the output of hidden layer is a vector of length 8. We also use a baseline method which aggregates messages without reweighing Vanilla GN. We also apply the two proposed networks, CurvGN-1 and CurvGN-n. Compared with the Vanilla GN, GCN reweighs messages using node degrees. GAT reweighs messages using self attention map computed using node representations. CurvGN-1 and CurvGN-n reweigh messages using scalar and vector computed by Ricci curvature.

We run all the methods on 100 random graphs. For each graph, we run the training and inference task for 10 times and take the average accuracy. For each training, we run 200 epochs and use validation set for early stopping. Figure 3 shows the heat maps for all methods. The title of each heatmap also includes the max and average performance over all parameter settings. In Figure 3(c), we run the same experiments on graphs with different sizes and report the average accuracy.

Discussion. Looking at the heatmaps, we observe that Vanilla GN, GCN and GAT are not better than random guessing. This implies that using node degree information is not enough. Meanwhile, we
observe improved performance by CurvGN-1, confirming the power of reweighing with curvature in graph convolution. In addition, CurvGN-n outperforms CurvGN-1, suggesting that the multi-channel reweighing based on curvature is beneficial. Furthermore, looking at the results with different graph sizes, we observe that the benefit of curvature increases as the graph size increases. We hypothesize that the graph convolution is sufficient in small graph setting to fully explore the graph structure. Only with larger graph and more diverse topology, advanced structural information becomes significant.

We also visualize the prediction results on one particular graph generated at \((p, q) = (0.21, 0.025)\) in Figure 4. We observe that CurvGN-1 and CurvGN-n make high quality predictions except for a small portion of data in a few communities. Meanwhile, other baselines can completely mix different communities and results are unsatisfactory.

Figure 4: One SBM result. Small nodes are training set. Larger nodes are testing set.

**Further studies on different graph theoretical models.** To further examine the expressive power of Ricci curvature, we run synthetic experiments using several well-accepted graph theoretical models: Watts–Strogatz network (Watts & Strogatz, 1998), Newman–Watts network (Newman & Watts, 1999) and Kleinberg’s navigable small world graph (Kleinberg, 2000). Watts-Strogatz network randomly rewires edges of a ring graph. Newman-Watts network randomly adds new edges to the ring. The Kleinberg’s model also adds random edges, but the probability of a new edge is inversely proportional to the geodesic distance between the nodes. In all these settings, we partition the ring into different communities and design experiments similar to the Stochastic Block Model. More details can be found in the Appendix A.2.

We compare all five methods on these different graph models and report the average accuracy over different parameter settings in Table 1. The standard deviations on these results are generally large as we are averaging over all different parameter settings. We observe consistently better performance of CurvGN-n than other methods. This confirms that curvature information is beneficial in a broad spectrum of graphs. We also note that GCN and GAT are indeed doing well for Watts-Strogatz and Newman-Watts models. This is because, in these networks, edge rewiring and addition create difference in node degrees. Bridges crossing different communities tend to have higher node degree. Therefore, node degrees carry useful structural information and can help with graph convolution. We do not observe the same benefit of node degree information in SBM and Kleinberg’s model as in

<table>
<thead>
<tr>
<th></th>
<th>GCN</th>
<th>GAT</th>
<th>Sum</th>
<th>CurvGN-1</th>
<th>CurvGN-n</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBM</td>
<td>24%</td>
<td>23%</td>
<td>28%</td>
<td>31%</td>
<td>36%</td>
</tr>
<tr>
<td>Watts–Strogatz</td>
<td>32%</td>
<td>30%</td>
<td>26%</td>
<td>29%</td>
<td>32%</td>
</tr>
<tr>
<td>Newman-Watts</td>
<td>32%</td>
<td>30%</td>
<td>27%</td>
<td>29%</td>
<td>33%</td>
</tr>
<tr>
<td>Kleinberg</td>
<td>23%</td>
<td>22%</td>
<td>28%</td>
<td>27%</td>
<td>31%</td>
</tr>
</tbody>
</table>

Table 1: Average prediction accuracy on four different graph models.
these models, node degree is not correlated with the locations of bridges. More heatmap results can be found in the Appendix A.2.

4.2 Real-World Benchmarks

Our real-world benchmarks include two families of datasets: small sparse graphs and large dense graphs. We compare our networks CurvGN-1 and CurvGN-n with several strong baselines. Aside from GCN and GAT that have been used in the synthetic experiments, we also compare CurvGN-1 and CurvGN-n with multilayer perceptron (MLP), MoNet (Monti et al. 2017), WSCN (Morris et al. 2019) and GraphSAGE with mean aggregation (GS-mean) (Hamilton et al. 2017). Our method is on par with state-of-the-art methods on relatively small graphs and greatly outperforms state-of-the-art methods on large and dense graphs, which tend to have heterogeneous topology.

Datasets. We use three popular citation network benchmark datasets: Cora, citeseer and PubMed (Sen et al. 2008). We categorize Cora and citeseer into the first family since both Cora and citeseer graphs are relatively small and sparse. They have thousands of nodes and edges with an average node degree below 2. We also use four extra datasets: Coauthor CS and Coauthor Physics which are co-authorship graphs based on the Microsoft Academic Graph from the KDD Cup 2016 challenge; Amazon Computers and Amazon Photos which are segments of the Amazon co-purchase graph in McAuley et al. (2015). These graphs, together with PubMed, are large and dense graphs. Those graphs have more than 10 thousands node and 200 thousands edges with an average node degree as high as 20. We use the exact data splitting as in semi-supervised learning setting used in Kipf & Welling (2016); Velickovic et al. (2017): using 20 nodes per class for training, 500 nodes for validation and 1000 nodes for testing. Descriptions and statistics for all datasets in our experiments can be found in the Appendix A.3.

During training stage, we set $L_2$ regularization with $\lambda = 0.0005$ for all datasets. Also, all the models are initialized by Glorot initialization and trained by minimizing cross-entropy loss using Adam SGD optimizer with learning rate $r = 0.005$. We apply an early stopping strategy with the help of the validation set’s accuracy with a patience of 100 epochs. We compute curvature exactly following Eq. (3.2) for all datasets but one. For the Amazon Computer dataset, we use an approximation scheme for efficiency (Ni et al. 2018). The linear programming problem is solved using an interior point solver (ECOS).

We report the mean and standard deviation of classification accuracy on test nodes on 100 runs and re-use the metrics reported by Monti et al. (2017); Shchur et al. (2018); Velickovic et al. (2017) for other state-of-the-art methods. The results are reported in Table 2. Our method is on par with state-of-the-art performance for relatively small graph and achieves superior performance on large and dense graphs. This is consistent with our conclusion from synthetic experiments: when graph is large and has heterogeneous topology, advanced structural information becomes critical in graph convolution.

Table 2: Performance on Real-World Benchmarks

<table>
<thead>
<tr>
<th>Method</th>
<th>Cora</th>
<th>Citeseer</th>
<th>PubMed</th>
<th>Coauthor CS</th>
<th>Coauthor Physics</th>
<th>Amazon Computer</th>
<th>Amazon Photo</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>58.2</td>
<td>59.1</td>
<td>70.0±2.1</td>
<td>88.3±0.7</td>
<td>88.9±1.1</td>
<td>44.9±5.8</td>
<td>69.6±3.8</td>
</tr>
<tr>
<td>MoNet</td>
<td>81.7</td>
<td>71.2</td>
<td>78.6±2.3</td>
<td>90.8±0.6</td>
<td>92.5±0.9</td>
<td>83.5±2.2</td>
<td>91.2±1.3</td>
</tr>
<tr>
<td>GS-mean</td>
<td>79.2</td>
<td>71.2</td>
<td>77.4±2.2</td>
<td>91.3±2.8</td>
<td>93.0±0.8</td>
<td>82.4±1.8</td>
<td>91.4±1.3</td>
</tr>
<tr>
<td>WSCN</td>
<td>78.9±0.9</td>
<td>67.4±0.8</td>
<td>78.1±0.6</td>
<td>89.1±0.7</td>
<td>90.7±0.9</td>
<td>67.6±3.7</td>
<td>82.1±1.2</td>
</tr>
<tr>
<td>GCN</td>
<td>81.5±0.5</td>
<td>70.9±0.5</td>
<td>79.0±0.3</td>
<td>91.1±0.5</td>
<td>92.8±1.0</td>
<td>82.6±2.4</td>
<td>91.2±1.2</td>
</tr>
<tr>
<td>GAT</td>
<td>83.0±0.7</td>
<td>72.5±0.7</td>
<td>79.0±0.3</td>
<td>90.5±0.6</td>
<td>92.5±0.9</td>
<td>78.0±19.0</td>
<td>85.1±20.3</td>
</tr>
<tr>
<td>CurvGN-1</td>
<td>82.6±0.6</td>
<td>71.5±0.8</td>
<td>78.8±0.6</td>
<td><strong>92.9±0.4</strong></td>
<td>94.1±0.3</td>
<td>86.3±0.7</td>
<td><strong>92.5±0.5</strong></td>
</tr>
<tr>
<td>CurvGN-n</td>
<td>82.7±0.7</td>
<td>72.1±0.6</td>
<td><strong>79.2±0.5</strong></td>
<td>92.8±0.3</td>
<td><strong>94.3±0.2</strong></td>
<td><strong>86.5±0.7</strong></td>
<td><strong>92.5±0.5</strong></td>
</tr>
</tbody>
</table>

5 Conclusion

We introduce a novel graph convolution network to leverage advanced graph structural information, namely, the graph curvature. The curvature information effectively helps achieve superior performance on synthetic and real-world datasets, especially on larger and denser graphs. This shows how principled mathematics and theory help the deployment of deep learning and encourages us to continue the endeavor in bridging the gap between graph theoretical foundation and neural networks.
REFERENCES


A APPENDIX

A.1 THE DETAILS OF MLP^T

We describe the details of MLP^t for the t-th convolutional layer, which maps a curvature κ_{xy} to the weight vector T_{xy}^t ∈ ℝ^F_{t+1}. MLP^t has three layers: an input layer, followed by a non-linear transformation layer and an output layer. The input layer linearly transforms the Ricci curvature κ_{xy} into an output vector with same size of message M_{y→x}^t ∈ ℝ^F_{t+1}. We use LeakyReLU function in our non-linear layer. For output layer Out, we use a transformation matrix with size F_{t+1} × F_{t+1} to compute reweighing vector T_{xy}^t. Formally,

MLP^t = Out(LeakyReLU(In))  (A.1)

Recall a node also passes a message to itself. To generate its weight vector T_{xy}^t, we set κ_{xx} = 0, as if the edge (x, x) is a grid edge. For the case when we reweigh the message using a single scalar τ_{xy} (e.g., CurvGN-1 network), we change the size of transformation matrix of output layer into F_{t+1} × 1.

A.2 DIFFERENT NETWORK MODELS FOR SYNTHETIC EXPERIMENTS

All three models, Watts-Strogatz, Newman-Watts and Kleinberg’s model, are created by randomly modifying/adding edges to a ring graph.\[1\] A ring graph has n nodes embedded on a circle, with each node connected to its k nearest neighbors. Figure 5(a) is an example ring graph with n = 20 and k = 4. To create communities, we partition the nodes into 5 equal-size sets according to their locations on the circle. In addition, we remove the edges cross different communities. Next we explain how edges of the ring graph are randomly changed for Watts-Strogatz, Newman-Watts and Kleinberg’s model, respectively.

**Watts-Strogatz Network.** Watts-Strogatz Network (Watts & Strogatz, 1998) is created by randomly rewiring edges of the ring graph with a predefined probability, p. See Figure 5(b) for an example of Watts-Strogatz network.

In our experiments, we generate 100 random Watts-Strogatz graphs of size n = 1000 using different parameter combinations of k and p: k ∈ {5, 10, · · · , 50} and p ∈ {0.02, 0.04, · · · , 0.2}. For each graph, the 5 communities correspond to nodes with 5 different labels. We randomly generate a 10-dimensional feature for each node, as in Stochastic Block Model experiments. The training set is created by randomly sampling 400 nodes in one graph. The validation set and testing set are create in the same way with size 400 and 200, respectively. For each graph, we run the experiment 10 times with 200 epochs each time and report the average.

Figure 6(a) shows the results of all five methods (GCN, GAT, Vanilla GN, CurvGN-1 and CurvGN-n). We observe that CurvGN-n has the best performance compared with others. It suggests that edge curvature information is crucial in prediction: a rewired edge has a high probability to be a bridge with negative curvature. Curvature information can effectively distinguish bridges and intra-community edges, and therefore help graph convolution. It is also worth mentioning that GCN also has good performance. We hypothesize that this is because rewired edges (likely bridges) tend to have higher curvature information is crucial in prediction: a rewired edge has a high probability to be a bridge with negative curvature. Curvature information can effectively distinguish bridges and intra-community edges, and therefore help graph convolution. It is also worth mentioning that GCN also has good performance. We hypothesize that this is because rewired edges (likely bridges) tend to have higher curvature information.

**Newman-Watts Network.** The Newman-Watts network (Newman & Watts, 1999) is similar to the Watts-Strogatz model except that it adds random edges on the ring graph with probability p, instead of rewiring existing edges. We run the experiments in the same setting as Watts-Strogatz model. The results are shown in Figure 6(b). We observe similar effects as Watts-Strogatz.

**Kleinberg’s Navigable Small World Graph.** Instead of randomly generating edges with a fixed probability p, Kleinberg’s model (Kleinberg, 2000) adds a fixed number of additional long-range edges to the ring graph. For each node u, add ε_l random edges (u, v) with v picked with a probability proportional to 1/d(u, v), in which d(u, v) is the distance between u and v in the circle.\[2\] We slightly modify the original definition by making all edges undirected and removing self-loops. Figure 5(c) shows an example graph of Kleinberg’s model with 100 nodes. We observe much fewer long range (cross-community) connections and more intra-community connections than the other models.

\[1\]Note these models can be built on any d-dimensional grid. Ring is a special case when d = 1.

\[2\]In general, the probability could be proportional to 1/d(u, v)^m. We choose m to be 1.
(a) An example ring graph with $n = 20$, $k = 4$.

(b) An example Watts-Strogatz network with 100 nodes.

(c) An example Kleinberg’s graph model demonstration with 100 nodes.

Figure 5: Example graphs.
Figure 6: Results on different graph models.
We generate 100 different graphs using different combinations of parameters $e_s$ and $e_l$. Here $e_s \in \{ \lfloor 2.5 \rfloor, \lfloor 5 \rfloor, \ldots, \lfloor 25 \rfloor \}$ controls the distance upperbound for short-range neighbors; any nodes within distance $e_s$ of $x$ is connected with $x$. $e_s$ is very similar to $k$ in Wattz-Strogatz graph. Similar to previous models, we run experiments on each graph 10 times and use the validation set for early stopping.

Since the added edge has a probability proportional to $1/d(u,v)$, there can be long distance inter-community edges. And the node degree can no longer capture the structural information introduced by the added edges. The GCN behaves similar to random guessing in this case. However, the Ricci Curvature is still negative on those inter-community edges and it can still predict communities. Figure 6(c) shows the heat maps of five different algorithms. CurvGN-n outperforms other methods by a large margin.

A.3 Statistical detail of benchmarks

We describe the statistical details of all datasets in Table 3. Cora and Citeseer are considered as small and sparse graphs while PubMed, Coauthors and Amazons are considered as large and dense graphs.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>#Classes</th>
<th>#Nodes</th>
<th>#Edges</th>
<th>#Features</th>
<th>#Training</th>
<th>#Edges/#Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>7</td>
<td>2708</td>
<td>5429</td>
<td>1433</td>
<td>140</td>
<td>2.0</td>
</tr>
<tr>
<td>Citeseer</td>
<td>6</td>
<td>3327</td>
<td>4732</td>
<td>3703</td>
<td>120</td>
<td>1.42</td>
</tr>
<tr>
<td>PubMed</td>
<td>3</td>
<td>19717</td>
<td>44338</td>
<td>500</td>
<td>60</td>
<td>2.25</td>
</tr>
<tr>
<td>Coauthor CS</td>
<td>15</td>
<td>18333</td>
<td>100227</td>
<td>6805</td>
<td>300</td>
<td>5.47</td>
</tr>
<tr>
<td>Coauthor Physics</td>
<td>5</td>
<td>34493</td>
<td>282455</td>
<td>8415</td>
<td>100</td>
<td>8.19</td>
</tr>
<tr>
<td>Amazon Computers</td>
<td>10</td>
<td>13381</td>
<td>259159</td>
<td>767</td>
<td>200</td>
<td>19.37</td>
</tr>
<tr>
<td>Amazon Photo</td>
<td>8</td>
<td>7487</td>
<td>126530</td>
<td>745</td>
<td>160</td>
<td>16.90</td>
</tr>
</tbody>
</table>

A.4 Complexity of Curvature Computation

Exactly calculating the curvatures is somewhat time-consuming. We need to solve a linear programming (LP) problem for each edge. For an edge $(u,v)$, the LP problem has $d_u \times d_v$ variables and $d_u + d_v$ linear constraints, in which $d_u$ and $d_v$ are the degrees of the nodes $u$ and $v$. Take the interior point solver (ECOS) as an example, the complexity is $O((d_u \times d_v)^w)$, in which $w$ is the exponent of matrix multiplication (the best known is $2.373$).

However, in practice, there are many approximation methods to accelerate the computation of optimal transport distance, among which the best known is the Sinkhorn algorithm (Cuturi, 2013). It is known to have near-linear time complexity \cite{Altschuler2017SinkhornSR}. So for small datasets we can keep using the ECOS to get exact solution while for large datasets (e.g. Computers) we may use advanced approximation methods. In Table 4, we show the computation time for the curvatures with two 18-core CPUs. Note that this computation is only needed once for data pre-processing, and the values can be reused in various downstream tasks, so it is feasible and practical.

Table 4: Time for curvature computation. "*" indicates the usage of approximation method.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>2.38</td>
</tr>
<tr>
<td>Citeseer</td>
<td>2.04</td>
</tr>
<tr>
<td>PubMed</td>
<td>28.76</td>
</tr>
<tr>
<td>CS</td>
<td>49.51</td>
</tr>
<tr>
<td>Physics</td>
<td>215.24</td>
</tr>
<tr>
<td>Computers*</td>
<td>1873.5</td>
</tr>
<tr>
<td>Photo</td>
<td>1237.55</td>
</tr>
</tbody>
</table>

A.5 Necessity of Multi-channel MLP

The usage of multi-channel MLP is essential in the success of using the curvature information. To illustrate this, in Table 5 we report the performance of our model if we directly use the curvature to weight messages, i.e., $\tau_{xy} = \kappa_{xy}$ for all layers. We call this baseline **CurvGN-κ**.

We observe that curvature alone cannot provide the best reweighting scheme for graph convolution. Only learning from the data helps us determine how weights on different channels should be dependent on $\kappa$. In Figure 7 we show for one example graph in CurvGN-n how $\tau_{xy}$ at different channels change
Table 5: Performance of a baseline which directly reweights messages with $\kappa_{xy}$

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Cora</th>
<th>Citeseer</th>
<th>PubMed</th>
<th>CS</th>
<th>Physics</th>
<th>Computers</th>
<th>Photo</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurvGN-$\kappa$</td>
<td>29.8</td>
<td>52.7</td>
<td>71.2</td>
<td>69.6</td>
<td>87.3</td>
<td>45.8</td>
<td>70.2</td>
</tr>
</tbody>
</table>

according to $\kappa_{xy}$. We observe weights of different channels are changing very differently w.r.t. $\kappa_{xy}$; some are increasing w.r.t. $\kappa_{xy}$, others are decreasing. MLP is necessary in learning such dependence from data. Using MLP enhances the model flexibility and model capacity. But also note that curvature $\kappa_{xy}$ is the essential input that the MLP relies on.

Figure 7: Illustration of different channels of the MLP output ($\tau_i$) v.s. the input curvature ($\kappa$). The model was trained on the synthetic SBM graph generated at $(p, q) = (0.21, 0.025)$ (visualized in Figure 4).