Curvature Graph Network

Anonymous Author(s)
Affiliation
Address
email

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 real-world graphs, especially the larger and denser ones.

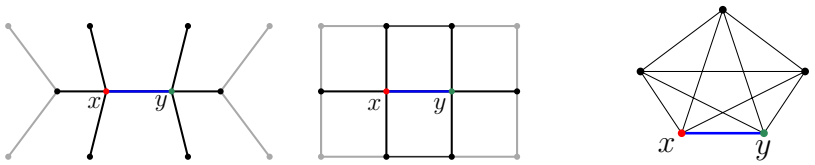
1 Introduction

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 and a biological network. Inspired by the power of convolution on image data, there are many attempts to extend convolutional networks for graph-structured data. For example, one category of graph neural networks, named spatial approaches, iteratively update each node representation by aggregating the information from its neighbors. For these 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 [9] or as an additional node feature [4]. 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. You*et al*. [20] calculates the shortest path between nodes. However this is not for node classification task and cannot extract complicate local information. 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 are focusing on designing 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 the recent study of *graph curvature* [12, 7, 18].

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 ones are Ollivier's Ricci curvature [12] and Forman curvature [18]. In both definitions, the edges of a (infinite) grid graph have zero curvature. The curvature of an edge (x, y) in a tree is negative and edges in a complete graph have positive curvature. Intuitively, the graph

Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.



(a) Tree: $\kappa(x, y) = -0.5$ (b) Grid: $\kappa(x, y) = 0$ (c) Clique: $\kappa(x, y) = +0.625$

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.

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)*, which is built on advance graph curvature information. In particular, the proposed CurvGN is 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 reweigh different channels of the message.

2 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 $H = (h_1, h_2, \cdots, h_n), h_i \in \mathbb{R}^F$. 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 consists of multiple hidden layers that update node representation from lower level node representation $H^t \in \mathbb{R}^{n \times F_t}$ to high level representation $H^{t+1} \in \mathbb{R}^{n \times F_{t+1}}$. 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 [5, 16, 19]. Denote by $\overline{\mathcal{N}}(x) = \mathcal{N}(x) \cup \{x\}$ the neighborhood of x including itself. We have $h_x^{t+1} = \sigma_t \left(\sum_{y \in \overline{\mathcal{N}}(x)} M_{y \to x}^t \right)$, in which σ_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 τ_{xy}^t whose purpose will be clear later. We have $M_{y \to x}^t = \tau_{xy}^t W^t h_y^t$, in which W^t is the linear transformation matrix learned in training. Formally, we have the representation updating equation

$$h_x^{t+1} = \sigma_t \left(\tau_{xx}^t W^t h_x^t + \sum\nolimits_{y \in \mathcal{N}(x)} \tau_{xy}^t W^t h_y^t \right) \tag{2.1}$$

It is crucial to obtain suitable reweighting parameter τ_{xy}^t since it is directly affecting how neighboring node information are passed to the node x. Some papers use node degree information as τ_{xy}^t [5, 9] and other work uses joint node features to compute the self attention as τ_{xy}^t [16]. We propose to use more advanced structural information, i.e., the Ricci curvature, to compute τ_{xy}^t. It is also worth mentioning that the reweighting parameter τ_{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 vector later on because it has more expressive power than a scalar and it is easier to train than a matrix.

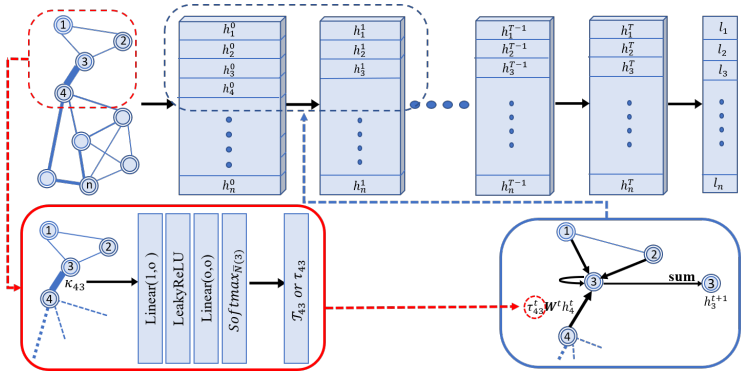


Figure 2: An overview of our Curvature Graph Network.

To illustrate how we build curvature convolution layer in Equation (2.1), we define Ricci Curvature in the context of graph (Section 2.1). We explain how to compute τ_{xy}^t from the curvature in Section 2.2.

2.1 Ricci Curvature

In Riemannian geometry, curvature measures how a smooth object deviates from being flat, or being straight in the case of a line. Similar concepts can be extended to non-smooth setting for discrete objects. In particular, curvature has been studied for metric-measure space in [1, 2, 8, 15], Markov chain by Ollivier [12] and general graphs in [7]. The definitions of curvatures that are easier to generalize in a discrete graph setting are sectional curvature and Ricci curvature.

To generalize Ricci curvature to discrete spaces, Ollivier [12] takes a coarse approach that represents S_x as a probability measure m_x of mass 1 around x. Thus the distance can be measured by Wasserstein distance (or Earth Mover distance) which finds the optimal mass-preserving transportation plan between two probability measures. Then the coarse Ricci curvature $\kappa(x, y)$ on edge (x, y) is defined by comparing the Wasserstein distance $W(m_x, m_y)$ to the distance $d(x, y)$, formally, $\kappa_{xy} = 1 - W(m_x, m_y)/d(x, y)$. For an undirected graph $G = (V, E)$, denotes the set of neighboring nodes of a node $x \in V$ as $N(x) = \{x_1, x_2, \ldots, x_k\}$. Then we can define a probability measure m_x^α at x: $m_x^\alpha(x_i) = \delta_{x_i=x} \times \alpha + \delta_{x_i \in N(x)} \times (1 - \alpha)/k$ where α is a parameter within $[0, 1]$ and δ is the indicator function. It is to keep probability mass of α 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:

$$\min_M \sum_{i,j} d(x_i, y_j) M(x_i, y_j), \tag{2.2}$$

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

where $M(x_i, y_j)$ is the amount of probability mass transported from node x_i to y_i along the shortest path with length $d(x_i, y_j)$. Following existing work [11], we set $\alpha = 0.5$.

2.2 Curvature-Driven Graph Convolution

Next we present how Ricci curvature is used in our graph convolutional network. The usage of curvature should depend on the problem and the data. Intuitively, curvature measures how easy a message flows through an edge, and should be used to control messages in convolution. For example, an edge with negative curvature is likely to be a bridge connecting two different communities. If we assume different communities tend to have different representations/labels, a message should be blocked on this edge. Meanwhile, an edge with positive curvature tends to be intra-community and thus should have accelerated message flow. However, the intuition may be invalid if the community structure is not correlated with node representation/labels.

We choose to be agnostic on how the knowledge of edge curvature should be used. We resort to a data-driven strategy and learn a mapping function that maps Ricci curvature κ_{xy} to the weight of messages, i.e., τ_{xy}^t in Equation (2.1). 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, τ_{xy}^t can be anything between a scalar and a $F^t \times F^t$ matrix. We first assume τ_{xy}^t is a scalar. In this case, the mapping function can be defined as: $f^t : \kappa_{xy} \to \tau_{xy}^t$.

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 MLP^t. As summation is used as the aggregation function in Equation (2.1), the messages may accumulate to an arbitrarily large value. To prevent a numerical explosion, we use softmax to normalize outputs of MLP over all neighbors. This gives us the eventual weight, τ_{xy}^t. Figure 2 bottom shows how the MLP transforms a curvature and uses it to reweigh messages.

CurvGN-n. Messages $M_{y\to x}^t$ 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 τ_{xy}^t, we create a similar mapping function as f^t. 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. More details can be found in the appendix.

3 Experiments

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 commonly compared baselines GCN and GAT, we also compare CurvGN-1 and CurvGN-n with multilayer perceptron (MLP), MoNet [9] and GraphSAGE with mean aggregation (GS-mean) [4]. 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 [13]. 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 in [14]: Coauthor CS, Coauthor Physics, Amazon Computers and Amazon Photos. 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. Descriptions and statistics for all datasets in our experiments can be found in the Appendix.

In Table 1, we report the mean and standard deviation of classification accuracy on test nodes on 100 runs and re-use the metrics reported by [9, 14, 16] for other state-of-the-art methods.

Table 1: Performance on Real-World Benchmarks

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

Discussion. Our method is on par with state-of-the-art performance for relatively small graphs. Meanwhile, it achieves superior performance on large and dense graphs. It is clear curvature information provides richer information than node degrees, at least on large and dense graphs. We also observe that CurvGN-n is generally better than CurvGN-1. This means multi-channel reweighting provides a better mechanism in leveraging curvature information. These observations are consistent with our synthetic experiments, which are not included due to page limitation. We conduct experiments on synthetic datasets generated according to various well-established graph models, e.g., stochastic block model [3], Watts-Strogatz network [17], Newman-Watts network [10] and Kleinberg's navigable small world graph [6]. On these data, CurvGN consistently outperforms GAT and GCN, demonstrating the benefit of curvature information.

References

[1] Anca-Iuliana Bonciocat. A rough curvature-dimension condition for metric measure spaces. *Open Mathematics*, 12(2):362–380, 2014.

[2] Anca-Iuliana Bonciocat and Karl-Theodor Sturm. Mass transportation and rough curvature bounds for discrete spaces. *Journal of Functional Analysis*, 256(9):2944–2966, 2009.

[3] Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E*, 84(6):066106, 2011.

[4] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[5] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[6] Jon M Kleinberg. Navigation in a small world. *Nature*, 406(6798):845, 2000.

[7] Yong Lin, Linyuan Lu, and Shing-Tung Yau. Ricci curvature of graphs. *Tohoku Mathematical Journal, Second Series*, 63(4):605–627, 2011.

[8] John Lott and Cédric Villani. Ricci curvature for metric-measure spaces via optimal transport. *Annals of Mathematics*, pages 903–991, 2009.

[9] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[10] Mark EJ Newman and Duncan J Watts. Renormalization group analysis of the small-world network model. *Physics Letters A*, 263(4-6):341–346, 1999.

[11] Chien-Chun Ni, Yu-Yao Lin, Jie Gao, and Xianfeng Gu. Network alignment by discrete ollivier-ricci flow. In *International Symposium on Graph Drawing and Network Visualization*, pages 447–462. Springer, 2018.

[12] Yann Ollivier. Ricci curvature of markov chains on metric spaces. *Journal of Functional Analysis*, 256(3):810–864, 2009.

[13] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[14] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[15] Karl-Theodor Sturm et al. On the geometry of metric measure spaces. *Acta mathematica*, 196(1):65–131, 2006.

[16] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[17] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440, 1998.

[18] Melanie Weber, Jürgen Jost, and Emil Saucan. Forman-ricci flow for change detection in large dynamic data sets. *Axioms*, 5(4):26, 2016.

[19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[20] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. *arXiv preprint arXiv:1906.04817*, 2019.

A Appendix

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

Substituting $\mathcal{T}_{xy}$ into Equation (2.1), we have the convolution of CurvGN-n:

$$h_x^{t+1} = \sigma_t\left(\sum\nolimits_{y\in\overline{\mathcal{N}}(x)} \text{diag}(\mathcal{T}_{xy}^t)W^t h_y^t\right) \tag{A.1}$$

Here diag$(\mathcal{T}_{xy}^t)$ is a matrix whose diagonal entries are entries of $\mathcal{T}_{xy}^t$.

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. input layer that transforms curvature information into 64 dimension feature vector and a linear layer followed by a LeakyReLU layer that generates a reweighting vector of dimension 64. 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 [16].

Statistical detail of benchmarks We describe the statistical details of all datasets in Table 2. Cora and Citeseer are considered as small and sparse graphs while PubMed, Coauthors and Amazons are considered as large and dense graphs.

Table 2: Statistic details of all datasets.

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

Data splitting and Hyper-parameters We use the exact data splitting as in semi-supervised learning setting used in [5, 16]: using 20 nodes per class for training, 500 nodes for validation and 1000 nodes for testing. During training stage, we set L_2 regularization with $\lambda = 0.0005$ for all datasets. 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 based on the validation set's accuracy with a patience of 100 epochs.