CECS

CENTER FOR EMBEDDED & CYBER-PHYSICAL SYSTEMS

System Level CPS design using Non-Euclidean Training method

Jiang Wan

Arquimedes Canedo

Mohammad Abdullah Al Faruque

Center for Embedded and Cyber-Physical Systems

University of California, Irvine

Irvine, CA 92697-2620, USA

{jiangwan, alfaruqu}@uci.edu

arquimedes.canedo@siemens.com

CECS Technical Report TR#18-02

June 4, 2018

Abstract

For the purpose of improving the design automoation of Cyber-Physical system from system level to detailed design stage. A novel non-euclidean training based method for categorizing functions from engineering data is proposed in this paper. In order to achieve the training task, a Structured Graph Convolutional Neural Network (SGCNN) is developed to perform graph invariant learning tasks at graph and subgraph level. The proposed method enables the first step toward the design automation of CPS at system level. The preliminary results show that we can achieve an accuracy of around 85% for machines to learn the functionality of a CPS.

1 Introduction

One of the key challenges in design automation of CPS is that some of the system level critical design decisions still require engineers' manual efforts. The reason is that most of these critical decisions requires creative thinking rather than logical behaviors which can be automatically explored by machines. Specifically, when a functionality of part of a CPS is given, engineers usually need to manually make system-level decisions such as choose the architectures of the system or choose the physical domains to implement the functionality [24, 4, 25]. Fortunately, thanks to the recent development of machine learning technology, the engineering knowledge can be learned by machines to enable design automation of CPS from its functional perspective.

Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed [19]. Artificial neural network (ANN) is one of the most commonly used models to implement the machine learning techniques. With the growth of computational power in the past few years, training an ANN model with many layers is possible. This method of training a multi-layered ANN is also known as deep learning. Deep learning has achieved a great success in solving problems such as speech recognition, image processing and computer vision. This success, however, are for clustering and classification of grid-structured Euclidean data (e.g. 1-D time-series signals, 2-D images, etc.). For example, Convolutional Neural Network (CNN) is a special ANN which specifically works on 2-D images, which can be considered as functions on the Euclidean space, sampled on a grid [2].

However, many of the real-world problems require learning tasks based on data which does not have a underlying Euclidean structure, defined as non-Euclidean data. Graphs and geometric manifolds are the two most common types of non-Euclidean data. For example, in social networks, the data is modeled as a social graph in which the users' information are embedded on the vertices, and the relations between users are embedded on the edges [7, 16, 22]. In Cyber Physical Systems (CPS) or Internet of Things (IoT), system components such as sensors, actuators and controllers can also be modeled as a system graph, and the status of these components can be modeled as time-dependent signals on the vertices. In information system and software system analysis, a data flow diagram (DFD) is also a non-Euclidean graphical representation of the "flow" of data through the system [3]. In electronic design, a netlist is a special type of graph describing the connectivity of an electronic circuit, which consists of a list of the electronic components in a circuit and a list of the nodes they are connected to. In computer graphics and vision, 3D objects are modeled as Riemannian manifolds (surfaces) which defines various geometric notions such as angles, lengths of curves, areas (or volumes), curvature, gradients of functions and divergence of

vector fields. [15]. In molecular biology, Protein Graph Repository (PGR) aims to explore the protein 3D-structure within a graph perspective where a protein is seen as a network of amino acids [9].

To replicate the same success of machine learning on Euclidean data, training the non-Euclidean data faces many challenges such as: lack of common system of coordinates, no vector space structure, shift-variance, and rotation-variance. As a result, the basic operations in ANN like convolution and pooling need to be redefined on non-Euclidean domains. And eventually, a new type of ANN needs to be developed for non-Euclidean data structure. However, to solve the different types of non-Euclidean data (e.g. Graph and Geometric manifolds) training problems may require different types of ANNs. In this paper, we are trying to a make step forward toward design automation of CPS with machine learning technique, and we are only targeting the problem of learning non-Euclidean data in the engineering domain. We assume the underlying data structure is a graph. Thus, we propose a Structured Graph Convolutional Neural Network (SGCNN) model that can be used to learn graph structured data. Nevertheless, the proposed model may also be applied in domains which require learning graph-structured data.

2 Related Works

Recently, researchers have been trying to apply CNN for non-Euclidean structured data. Briefly, we classify two types of non-Euclidean data learning problems in the existing works: (1) Learning and extracting features about the structure of the data; (2) Learning and analyzing the functions of the signals of the data. Most of the existing works for geometric manifold learning are trying to solve the first problem [21, 18, 1, 12, 5, 14, 8, 17]. They try to project a local high dimension structure into a low dimension space (e.g. a local coordinate system, an eigenvector based vector space) and extract features from the low dimension space for learning purposes.

In this paper, for the engineering domain problem, our interest falls into the second type of the problems. **Specifically, we are trying to classify existing CPS design data, which is represented in the form of graphs, into useful functions.** By learning the functions from existing CPS designs, the machine can extract knowledge from human design experience which is usable for high level design automation.

There have been multiple attempts to apply CNN on graphs. In general, there are two different approaches to achieve it. One is based on spectral domain analysis, the other is based on spatial/vertex domain analysis. In [8, 17], a Fourier transformation on graphs is proposed to project the high dimension signals, which lives on the vertex of the graph, to low dimension space constructed by the eigenbasis of the graph Laplacian operator [6]. Approximation of the transformation is also proposed in the current work to reduce the complexity. However, the spectral domain approach has a major limitation: it is not graph invariant. This is because all the spectral domain approaches rely on the Laplacian matrix of the graph. In other words, the Fourier transform on different graph will be different (due to the eigenvectors of Laplacian matrix being graph-dependent), thus a CNN trained for one graph can't be applied on another one.

On the other hand, most of the vertex domain approaches are based on the aggregation of neighbor nodes' information for every node in the graph [13, 20, 11], thus they are graph invariant. For example, in the recent GraphSAGE work [13], the authors propose to train a function to sample and aggregate a node's local neighborhood to the center node. In

this work, the samples are generated in a breadth-first search manner. Similarly, there also exists other works which differ mostly at the sampling methods. For example, in [11], the authors proposed to use a factor to tune the sampling ratio between breadth-first search and depth-first search. The vertex domain approach has shown the effectiveness on node-level clustering and classification. However, it also has the limitation, which is that it only works on node-level. And in many real-world problem, we want to be able to cluster or classify a whole graph or subgraph instead of a single node (e.g. 3D CAD model, social networks). Making an analogy to image classification, we want to classify whole images rather than pixels.

In addition to CNN on graph, traditionally graph kernels have been used in structure mining to measure the similarity of pairs of graphs [23]. Graph kernel can be used to classify or cluster graphs and subgraphs, however, it only considers the structure similarity between pairs of graphs. In engineering domain, it is not necessary for two subgraphs with different structures to be in different clusters. For example, two 3D-CAD models of an automotive system may have different structures but they will still have the same function and be in the same cluster labeled as automotive.

3 Contributions

In this paper, we propose a Structured Graph Convolutional Neural Network (SGCNN) that is able to perform graph invariant learning tasks at graph and subgraph level. The proposed SGCNN will be used to help the CPS design automation in the engineering domain. The major contribution of SGCNN are as follows:

- A novel SGCNN architecture that is able to perform training tasks at graph/subgraph level.
- A new path-based neighbor nodes aggregation method that aggregates vertex domain neighbor nodes information onto subgraphs.
- A novel subgraph convolution kernel that can perform graph invariant convolution operations on graph or subgraph.
- A downsampling/pooling algorithm based on the local structure of the subgraph.

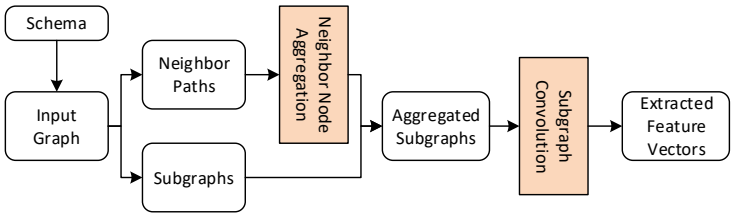

Figure 1: SGCNN Architecture

4 SGCNN Architecture

We define a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges. The graph edges can be weighted and directed. However, for simplicity, we only consider unweighted graph in this paper. For each $v_i \in \mathcal{V}$, we define the features to be f_i. And we define the adjacency matrix of $\mathcal{G}$ to be $\bar{A}$. A subgraph is defined as $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$, where $\mathcal{V}_s \in \mathcal{V}$ and $\mathcal{E}_s \in \mathcal{E}$.

The proposed SGCNN Architecture is shown in Figure 1. We assume a *schema* is available to generate subgraphs from the input graph. The *schema* should be given by the user as an input, which can be used for graph query [26]. It has two major steps (highlighted in Figure 1): (1) Neighbor Nodes Aggregation implements a path-based ANN to aggregate neighbor nodes information into the target subgraph. (2) Subgraph Convolution Kernel implements a graph invariant CNN on the target subgraph to extract feature vectors. The details will be presented in the following sections.

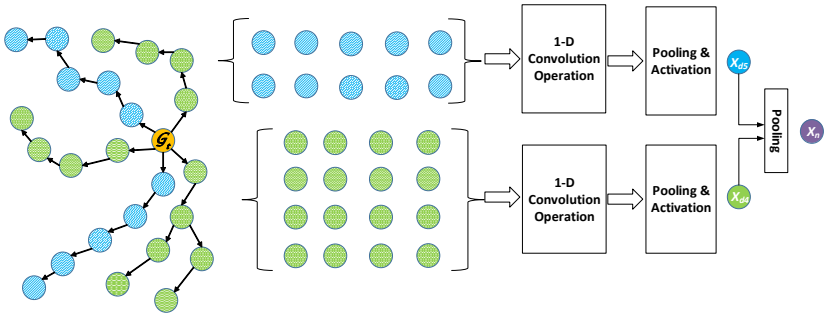

Figure 2: Neighbor Nodes Aggregation Example: Yellow node is a simple representation of a subgraph $\mathcal{G}_t$; Blue nodes represent length 5 paths; Green nodes represents length 4 paths.

4.1 Neighbor Nodes Aggregation

Similar to the vertex domain approach in [13], given a graph $\mathcal{G}$, we propose to aggregate the neighbor nodes features of the target subgraph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$. By considering that there are two methods to collect neighbor nodes in a graph: (1) Breadth first search; (2) Depth first search, we proposed a generalized path-based approach that integrates both of these two methods in engineering domain data. Specifically, we have developed the Neighbor Nodes Aggregation Layer which serves as hidden layers in SGCNN. For an Neighbor Nodes Aggregation Layer l_{ag}, we define two parameters: d as the depth to search, and n as the number of paths to be computed.

For all $v_i \in \mathcal{V}_t$, we search $\mathcal{G}$ to find all length d paths $\mathcal{P}_i^d$, which includes v_i, but doesn't include any other nodes in $\mathcal{V}_t$. Then we get all the length d neighbor paths of $\mathcal{G}_t$ as $\mathcal{P}^d = \{\mathcal{P}_0^d \mathcal{P}_1^d, ...\}$. From $\mathcal{P}^d$, we randomly select s paths (e.g. we can uniformly draw s samples), where s is a input parameter, and use each path as a row to form a neighbor feature matrix $\bar{N}$. Thus $\bar{N}$ is a n by d matrix with each element being a feature vector of a neighbor node. An example of this step is shown in Figure 2. Notice that in case the number of paths found in $\mathcal{P}^d$ is smaller than n, we can simply pad the $\mathcal{P}^d$ to make $\bar{N}$ at least have n number of rows/paths.

Algorithm 1: Neighbor Node Aggregation Process.

Input: An input Graph: G
Input: A schema for query: $Schema$
Input: A list of depth to search: $D = d_1, d_2, ..., d_n$
Input: A list of sample numbers per depth: $S = s_1, s_2, ..., s_n$
Output: A feature vector: x_n

Query G with $Schema$ to generate subgraphs G_t
foreach $d_i \in D$ **do**
  foreach $v_j \in \mathcal{V}_t$ **do**
    Find all length d_i paths $P_j^{d_i}$
    Remove paths contains nodes in $\mathcal{V}_t$ from $P_j^{d_i}$
    Add $P_j^{d_i}$ into P^{d_i}
  Construct $\bar{N}$ by randomly select s_i paths from P^{d_i}
  Extract feature x_{di} from $\bar{N}$
$x_n = f_{pool}^d(x_{d1}, x_{d1}, ..., x_{dn})$
return x_n

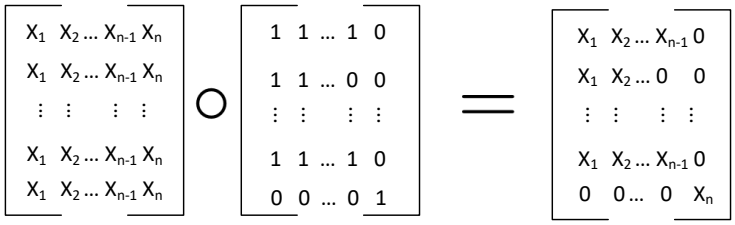

Figure 3: Attribute Matrix Example

Next task is to extract feature vectors from $\bar{N}$ as the output of this layer. As discussed in [13], the non-Euclidean data such as graph has no nature ordering. Thus, the feature extraction needs to be applied over an unordered set of paths. Or in other words, the rows of matrix $\bar{N}$ can be arbitrarily exchanged, and the extracted features should remain unchanged. In this paper, We apply a general 1-D convolution operation with a trainable 1 by d weight matrix $\bar{W}$ on $\bar{N}$ firstly, then we use a symmetric pooling function to extract the neighbor nodes feature vectors x_n as follows:

$$x_n = \sigma(f_{pool}(\bar{W} \circledast \bar{N}) + b) \tag{1}$$

Where, b is a bias variable, σ is an activation function (e.g. ReLU function), and f_{pool} is a pooling function which is invariant to permutations of rows in $\bar{N}$. For example, f_{pool} can be a mean operator over all elements in the matrix, or over all the rows in the matrix. Similarly, f_{pool} can also be a max operator as well. We argue that different f_{pool} functions/operators may be suitable in different domains, it can be a configurable parameter during the training process.

Notice that, we can aggregate paths with different lengths. For example, as shown in Figure 2, features from both length 5 and length 4 paths are extracted. In general, assume we extract features from paths with lengths $\{d_1, d_2, d_3, ..., d_k\}$ as $\{x_{d1}, x_{d2}, x_{d3}, ..., x_{dk}\}$. A pooling function f_{pool}^d can also be applied to reduce the dimensions of extracted features as:

$$x_n = f^d_{pool}(\{x_{d1}, x_{d2}, x_{d3}, ..., x_{dk}\}) \tag{2}$$

Finally, we will aggregate x_n to $\mathcal{G}_t$ by concatenate all the feature vectors of $v_i \in \mathcal{V}_t$ as $x_{agg} = \{x_i, x_n\}$. Algorithm 1 summaries our Neighbor Nodes Aggregation process.

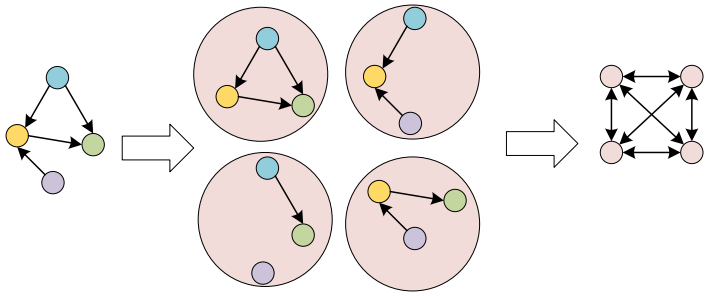

Figure 4: Convolution Kernel Example: Given input graph having 4 vertices, a 3-by-3 convolution kernel is applied. 4 convolution candidates are generated and the convolution results a new graph with 4 vertices

Algorithm 2: Graph Convolution Kernel.

Input: An input graph: G with n vertices
Input: An convolution kernel: $\bar{W}^k$
Input: Sample size: s
Output: An output feature graph: G'

Generate adjacency matrix $\bar{A}$ from G
Using the same vertices order to generate list of features $\mathcal{X}$
Create feature matrix $\bar{X}$ with n rows, and each row being $\mathcal{X}$
$\bar{Ar} = \bar{X} \circ (\bar{A} + \bar{I})$
$m = \binom{n}{k}$
$CombList$ = Enumerating choice of $n - k$ elements from $1, 2, ..., n$
foreach $comb \in CombList$ **do**
    $\bar{Ar}_{comb}$ = remove rows and columns list in $cand$ from $\bar{Ar}$
    Add $\bar{Ar}_{comb}$ into $CandList$
if $m > s$ **then**
    Down-sample $CandList$ to s elements
else if $m < s$ **then**
    Pad $CandList$ to s elements
foreach $cand \in CandList$ **do**
    $x^k = \bar{W}^k \circledast cand + b$
    Add new vertex v_k into G'
    Add feature vector x^k on v_k
    Connect v_k based on $cand$'s connection in G
return G'

4.2 Subgraph Convolution Kernel

The task of our proposed Subgraph Convolution Kernel is to extract feature vectors from graph or subgraph. First, we define the attribute matrix of a graph or subgraph as follows:

Given a target graph $\mathcal{G}$ with n number of vertices, by listing all the feature vectors x_i , which lives in $v_i \in \mathcal{V}$, in any given order, we can get the list of feature vector $\mathcal{X}$. Then, we repeat $\mathcal{X}$ by n times to form the feature matrix $\bar{X}$ with each row being $\mathcal{X}$. With the adjacency matrix $\bar{A}$, we define the attribute matrix $\bar{A}r$ of $\mathcal{G}$ as the Hadamard Product between $\bar{X}$ and $\bar{A} + \bar{I}$ as follows:

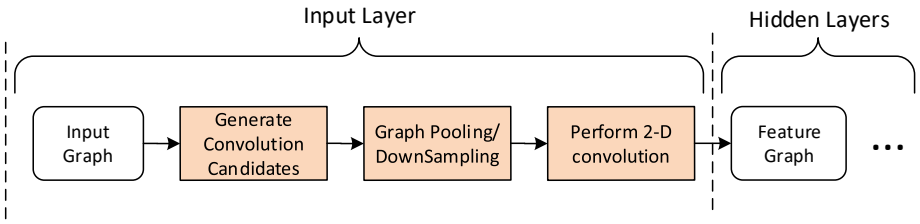

Figure 5: Subgraph Convolution Layer Architecture

$$\bar{A}r = \bar{X} \circ (\bar{A} + \bar{I}) \tag{3}$$

Algorithm 3: Graph Pooling Algorithm.

Input: An input graph: G
Input: The list of the candidate nodes combinations: $Comb$
Input: Sample size: s
Output: The list of downsampled canndidate nodes: $Comb'$
Generate adjacency matrix $\bar{A}$ from G
foreach $c \in Comb$ **do**
    $d_c = 0$
    foreach $n \in c$ **do**
        Calculate Degree of n as d_n
        $d_c = d_c + d_n$
    foreach $c' \in Comb$ **do**
        if c' *is connected with* c *in* $\bar{A}$ **then**
            $d_c = d_c + 1$
Keep the s number of nodes with highest degrees and store in $Comb'$
return $Comb'$;

where $\bar{I}$ is the identity matrix. For the purpose of maintaining the feature vectors of every vertex without loosing their own information [17], we add a self-loop to each node by the addition of $\bar{I}$. Notice that although we discuss only about unweighted graph in this paper, the proposed work can be expanded to weighted graph by applying a weighted adjacency matrix $\bar{A}$. An example of attribute matrix is shown in Figure 3.

Taking $\bar{A}r$ as the inputs for convolution operation on graph, we define a graph convolution kernel to be a k by k weight matrix $\bar{W}^k$. Then, we will apply convolution between $\bar{W}^k$ and $\bar{A}r$. However, different from a grid-structured 2D convolution between matrices (which the kernel will slide following a top-down, left to right order), we propose a new definition of convolution operation in graph data structure.

Since each row or column in $\bar{A}r$ is actually corresponding to a vertex in $\mathcal{G}_t$, thus, removing the i row and i column equals to remove the vertex i from $\mathcal{G}_t$. Assume n is bigger than k, we propose to remove $n - k$ number of vertex from $\mathcal{G}_t$, and the left over subgraph has a new

k by k attribute matrix $\bar{A}r^k$. And there is $\binom{n}{k}$ number of possible $\bar{A}r^k$. Thus, for all the possible $\bar{A}r^k$, we apply a simple convolution operation to extract feature vectors from this leftover subgraph as follows:

$$x^k = \bar{W}^k \circledast \bar{A}r^k + b \tag{4}$$

We will have m extracted feature vectors as: x_1^k, x_2^k, ..., x_m^k, and $m = \binom{N}{K}$ if we consider all possibilities. However, the complexity of this operation will be $O(\binom{n}{k})$, which is unacceptable in practice. To relax it, in this paper, instead of considering all possibilities, we will only pick s number of $\bar{A}r^k$ as convolution candidate which reduces the complexity to $O(s)$. Pick s samples of $\bar{A}r^k$ is a pooling/down-sampling operation which we will explain the details in Section 5. Finally, we consider the extracted feature vectors x_1^k, x_2^k, ..., x_s^k as a new feature graph $\mathcal{G}'$ with s number of vertices, and x_i^k is the feature vector for node i. An example is presented in Figure 4.

$\mathcal{G}'$ can be used as input to another Subgraph Convolution Layer in the proposed SGCNN architecture to form deep SGCNN model. However, notice that for $\mathcal{G}'$, we need to recalculate the adjacency matrix as well as the attribute matrix. Algorithm 2 summaries our Graph Convolution Kernel.

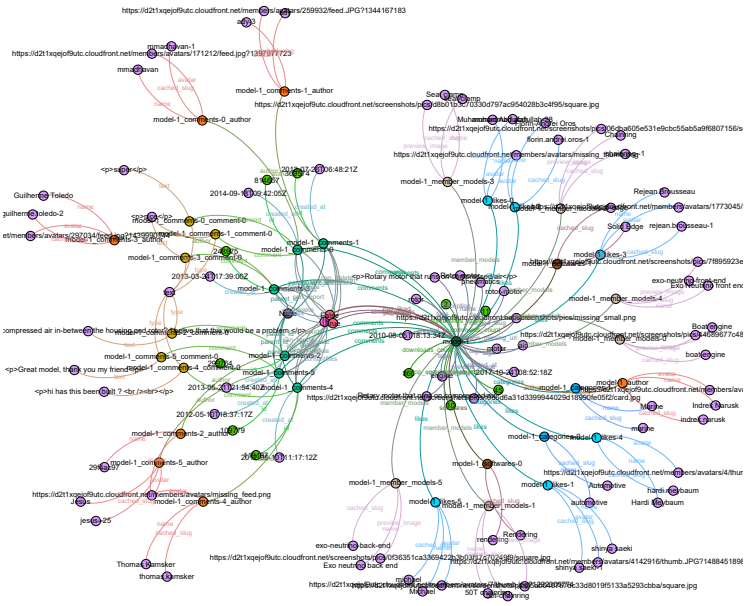

Figure 6: Example of a subgraph constructed from GrabCAD model

5 Graph Pooling Algorithm

In order to form deep structures for SGCNN, we propose a pooling operation before the convolution operations of each layer. The necessity of a pooling operation is because the combination of all possible convolution candidates is huge which has already explained in

previous section. Thus, with the a pooling operations inserted before convolution, the structure of our proposed subgraph convolution layer is presented in Figure 5

A nature idea of good downsampling/pooling operation is to remove the samples with less significant features. Thus, we consider to remove the subgraphs that has less total degrees compared to other samples for graph structured data. Algorithm 3 describes our proposed downsampling/pooling algorithm for graph.

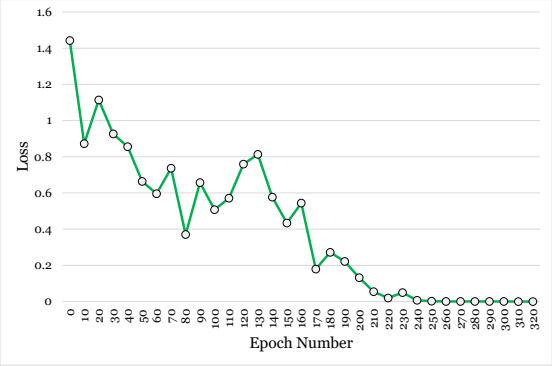

Figure 7: Loss value during the training process

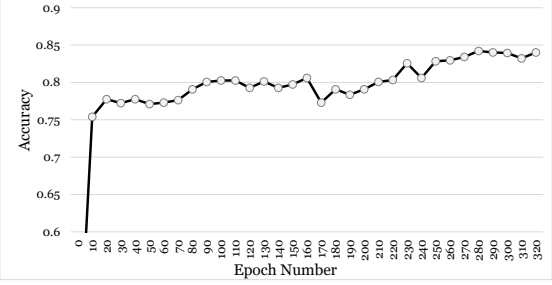

Figure 8: Accuracy results during the training process

In Line 1, we generate the adjacency matrix $\bar{A}$ from the input graph G. Lines 2-9 compute the total degrees of each candidate nodes combination $Comb_l$, which are combinations of the nodes in G that are generated by the convolution kernel and these combinations will serve as the new nodes of the output feature graph after the graph convolution kernel. Specifically,

lines 2-6 computes the total degrees inside the combination, and lines 7-9 computes the degrees in between different combinations. Finally, we will keep the s number of nodes combinations which have the highest degree.

6 Experimental Results

To evaluate the proposed non-euclidean training method, we applied our method on a dataset extracted from some samples in GrabCAD. GrabCAD is a free cloud-based collaboration solution that helps engineering teams manage, view and share CAD files [10]. GrabCAD consists of over 2 million open source engineering models, and we took 6090 models as training samples and 1523 models as testing samples for experimental purpose. From the GrabCAD website, we extracted the information (including model's name, model's author, model's parts names, model's tags, and model's comments, etc.) from the models and constructed graph from them. Notice that, the extracted information contains both social network data and engineering data. Nevertheless, we can use these data to construct graph and evaluate our method. One example of the constructed graph for one model is shown in Figure 6.

We classify all the samples into 4 different categories labeled as: (1) Car: 2237 samples; (2) Engine: 1623 samples; (3) Gear: 1731 samples ; (4) Robotic Arm: 2022 samples. A two layers SGCNN model is created for this learning task. The first layer has a kernel size of 16 by 16 and second layer has a kernel size of 4 by 4. The training results are shown in Figure 7 and Figure 8. The x-axis is the number of training echoes and the y-axis is the loss value and accuracy respectively.

The results show that, with the current dataset, we could converge to an accuracy of around 85%. This preliminary test results demonstrate that we can train machines to learn functionality of a engineering system using proposed non-euclidean training algorithms. Further testing with more engineering data from different sources will be conducted in future.

7 Conclusion

This paper proposes a novel non-euclidean training method for engineering data. A Structured Graph Convolutional Neural Network (SGCNN) is developed to perform graph invariant learning tasks at graph and subgraph level. The proposed method enables the design automation of CPS at system level with the help of machine learning technology. The preliminary results show that we can achieve an accuracy of around 85% for machines to learn the functionality of a CPS.

References

[1] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[2] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[3] P. D. Bruza and T. Van der Weide. *The semantics of data flow diagrams*. Citeseer, 1989.

[4] A. Canedo, J. Wan, and M. A. Al Faruque. Functional modeling compiler for system-level design of automotive cyber-physical systems. In *Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on*, pages 39–46. IEEE, 2014.

[5] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 891–900. ACM, 2015.

[6] F. R. Chung. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[7] D. J. Cook and L. B. Holder. *Mining graph data*. John Wiley & Sons, 2006.

[8] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[9] A. B. Diallo and W. Dhifli. Pgr: A novel graph repository of protein 3d-structures. *Journal of Data Mining in Genomics & Proteomics*, 6(02), 2015.

[10] GrabCAD. `https://grabcad.com/`.

[11] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[12] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 1735–1742. IEEE, 2006.

[13] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.

[14] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[15] J. Jost and J. Jost. *Riemannian geometry and geometric analysis*, volume 42005. Springer, 2008.

[16] U. Kang, C. E. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 229–238. IEEE, 2009.

[17] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[18] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[19] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[20] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[21] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[22] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*, 2011.

[23] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

[24] J. Wan, A. Canedo, and M. A. Al Faruque. Cyber–physical codesign at the functional level for multidomain automotive systems. *IEEE Systems Journal*, 2015.

[25] J. Wan, A. Canedo, and M. A. Al Faruque. Functional model-based design methodology for automotive cyber-physical systems. *IEEE Systems Journal*, 2015.

[26] P. T. Wood. Query languages for graph databases. *ACM SIGMOD Record*, 41(1):50–60, 2012.