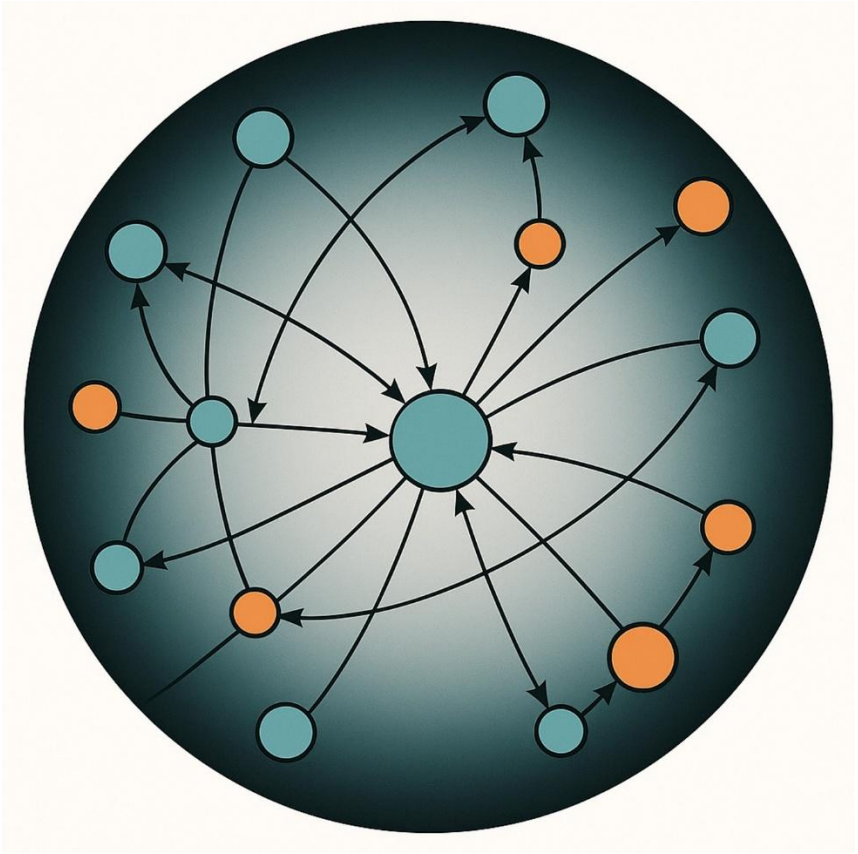
Analyzing Dynamic Neural Network Graph within Hyperbolic Space



Programmer's Manual

1.Overview -

This project analyzes dynamic (temporal) neural network graphs in hyperbolic space using advanced geometric deep learning techniques. It features data preparation, model training, anomaly detection, and visualization of temporal graph data, with a focus on citation networks.

This programmer's manual provides a comprehensive guide to the codebase for analyzing dynamic neural network graphs in hyperbolic space. It is intended for developers and researchers who wish to understand, use, or extend the project. The manual includes an explanation of the project structure, main components, setup instructions, and best practices for development and extension.

1.1 System Overview –

1.1.1 Architecture –

The system is designed around three main components:

- Data Preparation: Parses academic citation data, extracts features and constructs dynamic graph snapshots.

- Hyperbolic Neural Network Model: Implements models and operations over hyperbolic manifolds to learn node representations and detect anomalies.

- User Interface: Provides a graphical interface for data generation, experiment configuration, and visualization.

1.1.2 Dynamic Graph Processing and Embedding –

Utilizes paper metadata and references to generate temporal snapshots of citation networks.

Embeds nodes using hyperbolic neural models (e.g., HGCN) capturing both hierarchical and temporal relationships.

1.1.3 Anomaly Detection and Visualization –

Detects anomalous patterns in node activity over time using Isolation Forest and statistical analysis.

Offers a suite of visualization tools for examining anomaly scores, temporal patterns, and graph statistics

1.1.4 Threading and Workflow –

The UI leverages threading to keep the interface responsive during time-consuming operations like data generation and training.
Core workflow: DataPreparation → ModelTraining →Evaluation/Visualization.

2.Project Structure –

```
├──src
│   code
│   config.py
│   generateContentFile
│   generateData.py
│   inits.py
│   main_run.py
│   requirements.txt
│   UI.py
│   visualFunction.py
├──data
│   │   dblpv13.cites
│   │   dblpv13.content
│   │   final_filtered_by_fos_and_reference.csv
│   └──generate_custom_output
│           ind.dblpv13.allx
│           ind.dblpv13.ally
│           ind.dblpv13.graph
│           ind.dblpv13.snapshot_graphs
```

```
│       ind.dblpv13.test.index
│       ind.dblpv13.tx
│       ind.dblpv13.ty
│       ind.dblpv13.x
│       ind.dblpv13.y
├──hgcn
│   │   _init_.py
│   ├──layers
│   │   │   hyplayers.py
│   │   │   _init_.py
│   │   └──_pycache_
│   │       │       hyplayers.cpython-310.pyc
│   │       │       hyplayers.cpython-311.pyc
│   │       │       _init_.cpython-310.pyc
│   │       │       _init_.cpython-311.pyc
│   ├──manifolds
│   │   │   base.py
│   │   │   poincare.py
│   │   │   _init_.py
│   │   └──_pycache_
│   │       │       base.cpython-310.pyc
│   │       │       poincare.cpython-310.pyc
│   │       │       _init_.cpython-310.pyc
│   ├──utils
│   │   │   math_utils.py
```

```
│   │   │   _init_.py
│   │   └──_pycache_
│   │           math_utils.cpython-310.pyc
│   │           _init_.cpython-310.pyc
│   └──_pycache_
│           _init_.cpython-310.pyc
│           _init_.cpython-311.pyc
├──model
│   │   AdiHS.py
│   │   _init_.py
│   └──_pycache_
│           AdiHS.cpython-310.pyc
│       AdiHS.cpython-311.pyc
│       _init_.cpython-310.pyc
│       _init_.cpython-311.pyc
├──utilis
│   │   data_utilis.py
│   │   _init_.py
│   └──_pycache_
│           data_utilis.cpython-310.pyc
│           data_utilis.cpython-311.pyc
│           data_utilis.cpython-313.pyc
│           _init_.cpython-310.pyc
│           _init_.cpython-311.pyc
│           _init_.cpython-313.pyc
```

└── _pycache_

config.cpython-310.pyc

visualFunction.cpython-310.pyc

3.Installation & setup –

3.1 Clone the repository:

git clone https://github.com/YotamG12/1.-Analyzing-Dynamic-neural-network-graph-within-Hyperbolic-Space.git

cd 1.-Analyzing-Dynamic-neural-network-graph-within-Hyperbolic-Space

3.2 Install dependencies:

- Python 3.10

- Required libraries:

pip install torch numpy pandas scikit-learn networkx matplotlib tabulate torch-geometric

- Additional dependencies may be required (refer to import statements in `src/`).

3.3 Prepare Data:

- Place the raw data CSV in `./data/final_filtered_by_fos_and_reference.csv`.

- Run data generation either via the UI or directly:

cd src

python generateData.py

4.Main Modules & Their Purpose –

4.1 Data Generation –

Prepares the dataset by reading the CSV file, splitting into train/val/test, extracting features and labels, constructing adjacency lists, and saving data for further use

Key steps:

- Loads papers and references from CSV.
- Builds feature matrixes and label matrixes.
- Constructs adjacency lists (graphs) based on references.
- Splits data by year into train/val/test sets.
- Builds dynamic graph snapshots for temporal analysis.
- Saves all processed data as pickles (.pkl files) in `data/generate_custom_output/`.

4.2 Main Experiment Runner –

Loads the generated data, initializes the model, trains it, tracks overfitting, and produces plots.

Key steps:

- Loads pickled data and meta-data.
- Prepares node features, labels, and edge lists.
- Initializes model (e.g., `AdiHs`) and optimizer.
- Handles training loop and anomaly tracking.
- Utilizes visualization utilities for anomaly detection and graph analysis.

4.3 User Interface –

A Tkinter-based GUI for easier interaction, allowing users to trigger data generation, set hyperparameters, run experiments, and view outputs.

Key Components:

- Data frame for time-slice input and data generation.

- Hyperparameter selection for model training.

- Validation frame for noise injection settings.

- Console output for logs and messages.

- Buttons to trigger generation and main run.

4.4 Manifolds & Geometry –

Defines abstract base classes for manifold operations in hyperbolic neural networks. Any new manifold should implement these interfaces.

Key Classes:

-Manifold: Abstract class for geometric operations (distance, projection, exp/log maps, Mobius addition, etc.).

- ManifoldParameter: Subclass of torch.nn.Parameter ,to store manifold and curvature info for Riemannian optimization.

4.5 Initializations –

Contains initialization functions for model parameters (e.g., Xavier, also known as Glorot) and data preparation helpers.

5. Data & File Format –

Input data - data/final_filtered_by_fos_and_reference.csv (must include columns: `id`, `abstract`, `references`, `year`, `fos.name`, etc.)

Output data - Pickle files saved in data/generate_custom_output/, including:

- x, y: Features and labels for train.
- tx, ty: Features and labels for test.
- allx, ally: Features/labels for train+val.
- graph: Adjacency list.
- snapshot_graphs: Dynamic graph snapshots.
- ind.dblpv13.test.inde: List of test indices.

6. Development Workflow –

6.1 Data Preparation:

- Ensure the CSV is present.
- Run `generateData.py` (via UI or CLI).

6.2 Experimentation:

- Adjust hyperparameters via the UI or directly in code/config.
- Run main_run.py for training, evaluation, and visualization.

6.3. Visualization:

- Check the `plots/` directory for generated analysis plots.

6.4 Model/Manifold Development:

- To add new geometric operations, inherit from `Manifold` and implement required methods.

7.Core Algorithms –

The algorithms in this project are designed to analyze dynamic (temporal) citation networks using hyperbolic neural network models. The approach captures structural and temporal changes in the graph, detects anomalies, and generates meaningful node embeddings.

Dynamic Graph Processing Pipeline -

1. Dynamic Graph Snapshot Creation

Splits citation data into time slices (snapshots) based on publication year or user-defined intervals.

For each time slice, constructs an adjacency list and feature/label matrices for all papers (nodes).

2. Hyperbolic Embedding Generation

Trains a hyperbolic neural network model (e.g., HGCN) on each time slice or across the temporal sequence.

Outputs node embeddings that capture both the hierarchical and temporal structure of the citation network.

3. Anomaly Detection

Calculates anomaly scores for nodes over time using methods like Isolation Forest and statistical measures (e.g., standard deviation of node activity).Identifies nodes or time periods with unusual behavior.

4. Visualization and Analysis

Generates plots to visualize anomaly distributions, sharp changes, and embedding trajectories over time.

Dynamic Graph Snapshot Creation (Algorithm 1)

- Iteratively constructs a sequence of graph snapshots:

1. Groups papers by time (e.g., year).

2. For each group, builds a graph where nodes are papers and edges are citation relationships.

3. Extracts features (e.g., via text vectorization of abstracts) and encodes research fields as labels.

- Stops when all time intervals are processed.

Hyperbolic Neural Network Embedding (Algorithm 2)

Learns node embeddings that preserve both the local and global (hierarchical) structure of each graph snapshot.

1. Initializes node features, adjacency matrix, and labels.

2. Applies layers of neural operations defined over a hyperbolic manifold.

3. Uses Riemannian optimization for training (with manifold-aware gradients and projections).

4. Outputs node embeddings for each time slice.

Anomaly Detection and Scoring (Algorithm 3)

Identifies unusual node or edge patterns in temporal graphs.

1. Computes anomaly scores for each node at each time slice (using Isolation Forest on node features or embedding trajectories).

2. Aggregates scores to identify significant anomalies or sharp changes.

3. Applies thresholding and ranking to highlight significant nodes and periods.

Visualization and Temporal Analysis (Algorithm 4)

Interprets and displays results for insight and validation.

1. Plots distributions of anomaly scores across time slices.
2. Highlights nodes with sharpest anomaly changes.
3. Visualizes embedding trajectories and dynamic graph properties.
4. Supports interactive exploration via the UI or exported plots.

These core algorithms work together to support end-to-end dynamic network analysis, from data preparation and representation learning to anomaly detection and result visualization.