Project: Graph Neural Networks for End-to-End Particle Identification with the CMS Experiment.

Participating Organizations: ***ML4SCI (Umbrella Organization), University of Alabama, Brown University, Carnegie Mellon University.***

Google Summer of Code '21

ANURAG GORKAR gorkaranurag@gmail.com

❖ *About Me:*

➢ Name and Contact information

- Name: Anurag Dilip Gorkar
- Time zone: IST (UTC +5:30)
- GitHub id: AnuragGorkar
- Email: gorkaranurag@gmail.com
- Medium: https://medium.com/@gorkaranurag
- LinkedIn: LinkedIn Anurag Gorkar
- Mobile: +91 7722089193 Education

➢ Education

- University: Savitribai Phule Pune University
- College: Pune Institute of Computer Technology
- Degree: B.E (Bachelor of Engineering)
- Field of Study: Computer Engineering
- Current Year: 3rd year (6th semester ongoing)
- CGPA: 9.4/10
- Expected Graduation: 2022

➢ Personal Background

Computers have fascinated me since early school days and I have been a programming geek since high school. I was introduced to **Machine Learning** in my college and that has excited me enough to explore more and more in the past 3 years. **Graph Convolutional Networks** and **Computer Vision** are especially my areas of interest. To start with, I have worked on many projects related to image classification and analysis.

I am the **Coding Department Head** at the PICT Robotics Club. We participate in the ABU Robocon competition each year. While working in this club I have gained substantial experience in collaborating with others to **write clean code using git and GitHub**.

I interned at the Defence Research and Development Organization (**DRDO**), where we developed a framework using **Flask** (**Python)** and **PyTorch** which leverages **machine learning steganalysis techniques** and **complex algorithmic analysis** to **detect malware** in image samples.

I have also co-authored a **Research Paper** "Intensive Image Malware Analysis and Least Significant Bit Matching Steganalysis" which has been published by **IEEE** at the 2020 IEEE International Conference on Big Data (Big Data) . This endeavour helped me gain experience in writing documentation in a proper format.

These experiences were very enriching and ingrained in me the importance of clean code and documentation. I am proficient in **C, C++, and Python, PyTorch and TensorFlow**.

➢ Achievements

I won the **first prize** in the PICT internal hackathon to represent in Smart India Hackathon (where more than 100 teams had participated).

After winning the internal hackathon in my college (where more than 100 teams participated), I was selected as **the top college representative in "Smart India Hackathon"**.

➢ Motivation

During my course study at college, I was introduced to Graphs as a Data Structure. I was fascinated by the fact that how such a **simple and primitive concept** of connecting points (nodes) which have a certain co-relation with each other with edges can help model and understand some of the most **complex relational sets from the real world**.

I have experience working with regular data structures like images / texts during my Machine Learning Projects and internship. However, I was always inquisitive to know how these concepts of machine learning and deep learning can be extended to **unstructured / irregularly structured data**.

As a unique **non-Euclidean data structure** for machine learning, graph analysis focuses on tasks such as **node classification, link prediction, and clustering**. Graph neural networks (GNNs) are deep learning-based methods that operate on graph domain.

Recently, deep learning on graphs has emerged to one of the hottest research fields in the deep learning community. Here, **Graph Neural Networks (GNNs)** aim to generalize classical deep learning concepts to **irregular structured data** (in contrast to images or texts) and to enable neural networks to reason about objects and their relations.

After experimenting with and reading about Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Attention in ML I would like to increase my knowledge spectrum by working on variants of GNNs such as **graph convolutional network** (GCN), **graph attention network** (GAT), **graph recurrent network** (GRN) which have demonstrated ground-breaking performances on many deep learning tasks.

❖*About the Project:*

➢ Abstract

▪ Description:

1. One of the important aspects of searches for new physics at the Large Hadron Collider (LHC) involves the identification and reconstruction of single particles, jets and event topologies of interest in collision events. The End-to-End Deep Learning (E2E) project in the CMS experiment focuses on the development of these reconstruction and identification tasks with innovative deep learning approaches.
2. This project will focus on the development of end-to-end graph neural networks for particle (tau) identification and CMSSW inference engine for use in reconstruction algorithms in offline and high-level trigger systems of the CMS experiment.

▪ Task Ideas:

1. Development of end-to-end graph neural networks for low-momentum tau identification.
2. Test and benchmarking of inference on GPUs.

▪ Expected Results:

1. Trained end-to-end graph neural network model for tau identification.
2. Benchmark of end-to-end GNN inference on GPU.

➢ Proposed Solution

▪ Software Tools/Libraries

1. Deep Graph Library:

i. Deep Graph Library (DGL) is a Python package built for easy implementation of graph neural network model family, on top of existing DL frameworks (currently supporting **PyTorch, MXNet and TensorFlow**).

ii. DGL offers a versatile control of message passing, speed optimization via auto-batching and highly tuned sparse matrix kernels, and **multi-GPU/CPU training to scale** to graphs of hundreds of millions of nodes and edges.

iii. The nodes and edges of a **DGLGraph** can have several user-defined named features for storing **graph-specific properties** of the nodes and edges. These features can be accessed via the **ndata** and **edata** interface.

2. PyTorch Geometric Library:

i. PyTorch Geometric is a **geometric deep learning extension** library for **PyTorch.**

ii. It consists of various methods for deep learning on graphs and other irregular structures, also known as geometric deep learning, from a variety of **published papers**.

iii. PyTorch Geometric consists of an easy-to-use mini-batch loader for many small and single giant graphs, a large number of common **benchmark datasets** and helpful transforms, both for learning on arbitrary graphs as well as on **3D meshes or point clouds**.

▪ Papers referred / Model Architectures implemented:

1. Kipf Et al. (ICLR 2017) Semi-Supervised classification with graph convolutional networks:

i. This present a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks which operate directly on graphs.

ii. It motivates the choice of convolutional architecture via a localized first-order approximation of spectral graph convolutions.

iii. The model scales linearly in the number of graphs edges and learns hidden layer representations that encode both local graph structure and features of nodes.

Following is a code which performs Node Classification Task on the Cora dataset according to the GCN architecture mention in the above-mentioned paper:

Node Classification with DGL

GNNs are powerful tools for many machine learning tasks on graphs. In this notebook we use GNNs for node classification, i.e. predicting the category of a node in a graph.

```
import dgl
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
DGL backend not selected or invalid.  Assuming PyTorch for now.
Setting the default backend to "pytorch". You can change it in the ~/.dgl/config.json fi
Using backend: pytorch
```

Overview of Node Classification with GNN

One of the most popular and widely adopted tasks on graph data is node classification, where a model needs to predict the ground truth category of each node. Before graph neural networks, many proposed methods are using either connectivity alone (such as DeepWalk or node2vec), or simple combinations of connectivity and the node's own features. GNNs, by contrast, offers an opportunity to obtain node representations by combining the connectivity and features of a *local neighborhood*.

`Kipf et al., <https://arxiv.org/abs/1609.02907>`__ is an example that formulates the node classification problem as a semi-supervised node classification task. With the help of only a small portion of labeled nodes, a graph neural network (GNN) can accurately predict the node category of the others.

Here build such a GNN for semi-supervised node classification with only a small number of labels on the Cora dataset, a citation network with papers as nodes and citations as edges. The task is to predict the category of a given paper. Each paper node contains a word count vector as its features, normalized so that they sum up to one, as described in Section 5.2 of `the paper <https://arxiv.org/abs/1609.02907>`__.

Loading Cora Dataset

```
import dgl.data

dataset = dgl.data.CoraGraphDataset()
print('Number of categories:', dataset.num_classes)
```

```
Downloading /root/.dgl/cora_v2.zip from https://data.dgl.ai/dataset/cora_v2.zip...
Extracting file to /root/.dgl/cora_v2
Finished data loading and preprocessing.
  NumNodes: 2708
  NumEdges: 10556
  NumFeats: 1433
  NumClasses: 7
  NumTrainingSamples: 140
  NumValidationSamples: 500
  NumTestSamples: 1000
Done saving data into cached files.
Number of categories: 7
```

A DGL Dataset object may contain one or multiple graphs. The Cora dataset consists of one single graph.

```
g = dataset[0]
```

A DGL graph can store node features and edge features in two dictionary-like attributes called `ndata` and `edata`. In the DGL Cora dataset, the graph contains the following node features:

- `train_mask`: A boolean tensor indicating whether the node is in the training set.
- `val_mask`: A boolean tensor indicating whether the node is in the validation set.
- `test_mask`: A boolean tensor indicating whether the node is in the test set.
- `label`: The ground truth node category.
- `feat`: The node features.

```
print('Node features')
print(g.ndata)
print('Edge features')
print(g.edata)
```

```
Node features
{'train_mask': tensor([ True,  True,  True,  ..., False, False, False]), 'val_mask': ter
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.]])}
Edge features
{}
```

Defining a Graph Convolutional Network (GCN)

We build a two-layer `Graph Convolutional Network (GCN) <http://tkipf.github.io/graph-convolutional-networks/>__`. Each layer computes new node representations by aggregating neighbor information.

To build a multi-layer GCN we can simply stack `dgl.nn.GraphConv` modules, which inherit `torch.nn.Module`.

```
from dgl.nn import GraphConv

class GCN(nn.Module):
    def __init__(self, in_feats, h_feats, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GraphConv(in_feats, h_feats)
        self.conv2 = GraphConv(h_feats, num_classes)

    def forward(self, g, in_feat):
        h = self.conv1(g, in_feat)
        h = F.relu(h)
        h = self.conv2(g, h)
        return h

# Create the model with given dimensions
model = GCN(g.ndata['feat'].shape[1], 16, dataset.num_classes)
```

Training the GCN

Training this GCN is similar to training other PyTorch neural networks.

```
def train(g, model):
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
    best_val_acc = 0
    best_test_acc = 0

    features = g.ndata['feat']
    labels = g.ndata['label']
    train_mask = g.ndata['train_mask']
    val_mask = g.ndata['val_mask']
    test_mask = g.ndata['test_mask']
    for e in range(100):
        # Forward
        logits = model(g, features)

        # Compute prediction
```

```
        pred = logits.argmax(1)

        # Compute loss
        # Note that you should only compute the losses of the nodes in the training set.
        loss = F.cross_entropy(logits[train_mask], labels[train_mask])

        # Compute accuracy on training/validation/test
        train_acc = (pred[train_mask] == labels[train_mask]).float().mean()
        val_acc = (pred[val_mask] == labels[val_mask]).float().mean()
        test_acc = (pred[test_mask] == labels[test_mask]).float().mean()

        # Save the best validation accuracy and the corresponding test accuracy.
        if best_val_acc < val_acc:
            best_val_acc = val_acc
            best_test_acc = test_acc

        # Backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if e % 5 == 0:
            print('In epoch {}, loss: {:.3f}, val acc: {:.3f} (best {:.3f}), test acc: {:.3f}
                e, loss, val_acc, best_val_acc, test_acc, best_test_acc))
model = GCN(g.ndata['feat'].shape[1], 16, dataset.num_classes)
train(g, model)
```

```
In epoch 0, loss: 1.946, val acc: 0.072 (best 0.072), test acc: 0.074 (best 0.074)
In epoch 5, loss: 1.898, val acc: 0.516 (best 0.516), test acc: 0.529 (best 0.529)
In epoch 10, loss: 1.822, val acc: 0.682 (best 0.682), test acc: 0.708 (best 0.708)
In epoch 15, loss: 1.722, val acc: 0.718 (best 0.718), test acc: 0.733 (best 0.733)
In epoch 20, loss: 1.598, val acc: 0.692 (best 0.718), test acc: 0.726 (best 0.733)
In epoch 25, loss: 1.452, val acc: 0.710 (best 0.718), test acc: 0.729 (best 0.733)
In epoch 30, loss: 1.289, val acc: 0.714 (best 0.718), test acc: 0.734 (best 0.733)
In epoch 35, loss: 1.118, val acc: 0.720 (best 0.720), test acc: 0.744 (best 0.737)
In epoch 40, loss: 0.951, val acc: 0.732 (best 0.732), test acc: 0.754 (best 0.754)
In epoch 45, loss: 0.795, val acc: 0.740 (best 0.740), test acc: 0.759 (best 0.757)
In epoch 50, loss: 0.658, val acc: 0.746 (best 0.746), test acc: 0.767 (best 0.767)
In epoch 55, loss: 0.541, val acc: 0.748 (best 0.750), test acc: 0.770 (best 0.770)
In epoch 60, loss: 0.445, val acc: 0.750 (best 0.750), test acc: 0.779 (best 0.770)
In epoch 65, loss: 0.366, val acc: 0.748 (best 0.752), test acc: 0.778 (best 0.780)
In epoch 70, loss: 0.303, val acc: 0.752 (best 0.752), test acc: 0.776 (best 0.780)
In epoch 75, loss: 0.252, val acc: 0.752 (best 0.752), test acc: 0.776 (best 0.780)
In epoch 80, loss: 0.210, val acc: 0.758 (best 0.758), test acc: 0.779 (best 0.779)
In epoch 85, loss: 0.177, val acc: 0.764 (best 0.766), test acc: 0.775 (best 0.777)
In epoch 90, loss: 0.150, val acc: 0.762 (best 0.766), test acc: 0.772 (best 0.777)
In epoch 95, loss: 0.128, val acc: 0.756 (best 0.766), test acc: 0.772 (best 0.777)
```

Training on GPU

Training on GPU requires to put both the model and the graph onto GPU with the `to` method, similar to what you will do in PyTorch.

.. code:: python

g = g.to('cuda') model = GCN(g.ndata['feat'].shape[1], 16, dataset.num_classes).to('cuda') train(g, model)

2. Weihua Hu Et al. (ICLR 2020) Strategies for Pre-Training Graph Neural Networks:

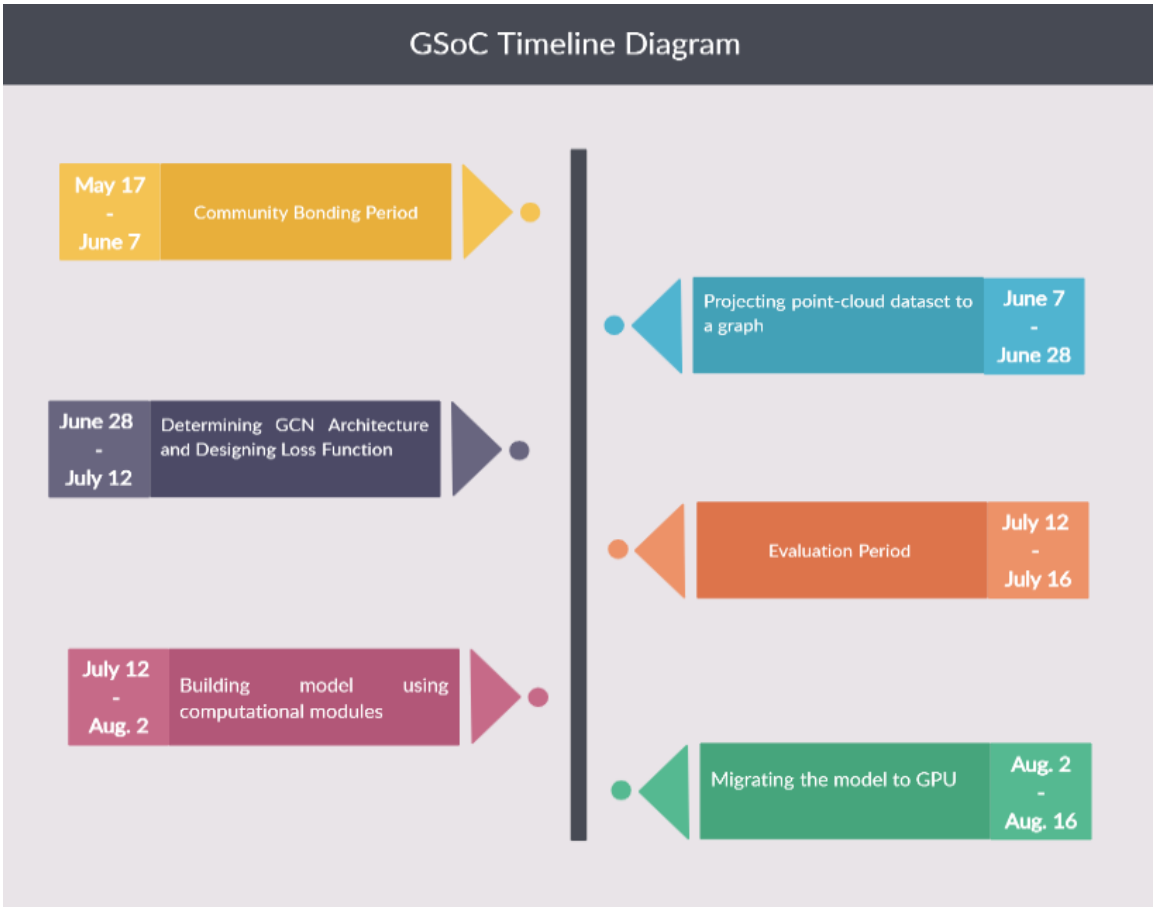
i. In this paper, the authors develop a new strategy and self-supervised methods for pre-training Graph Neural Networks (GNNs).

ii. The key to the success of our strategy is to pre-train an expressive GNN at the level of individual nodes as well as entire graphs so that the GNN can learn useful local and global representations simultaneously.

iii. The author systematically studies pre-training on multiple graph classification datasets.

iv. This strategy avoids negative transfer and improves generalization significantly across downstream tasks, leading up to 9.4% absolute improvements in ROC-AUC over non-pre-trained models and achieving state-of-the-art performance for molecular property prediction and protein function prediction.

❖ *Timeline:*



➢ Community Bonding Period (May 17 – June 7):

The most important task I will be committing to during this time is getting familiar with the codebase and community. I will be reading a survey paper Jie Zhou Et al.: Graph neural networks: A review of methods and applications which proposes a general design pipeline for GNN models and discusses the variants of each component, systematically categorize the applications, and propose four open problems for future research. I will be implementing some architectures performing node classification for graphs by going through the state-of-the-art research papers. I am already familiar with the **PyTorch** component. In order to better understand the data set and interrelation between different data points to construct an accurate graph representation of the data I am planning to read about particle physics briefly.

I will constantly stay in touch with the mentors and the ML4SCI community and will make sure I clear my doubts regarding any issues I face that time as well as discuss the factors that can improve my project.

By the end of this period, my aims are to:

- Learn about the codebase of the ML4SCI and contribute at least one example.
- Discuss with the mentors about the Milestones to be set for the coding period.
- Align on the ways of working and project tracking with mentors.
- Align on the ways of working and project tracking with mentors.
- Learn thoroughly about developing and implementing Graph Convolutional Network models and the best practices used.

➢ Coding Period (June 7 – April 16):

Coding Period Begins

- **Week 1-Week 3: (June 7 - June 28):**
  Projecting point-cloud dataset to a set of interconnected nodes and edges.
  - Studying the low momentum tau dataset and CMSSW inference engine.

  - Determining the algorithm to accurately project the data to a graph such that it accurately represents the relations between tau jets.

  - Determining the type of graph (Heterogenous/Homogenous, Weighted/Unweighted, Directed/Un-directed).

    1. Directed/Undirected Graphs:

       Edges in directed graphs are all directed from one node to another, which provide more information than undirected graphs. Each edge in undirected graphs can also be regarded as two directed edges.

2. Homogeneous/Heterogeneous Graphs:

   Nodes and edges in homogeneous graphs have same types, while nodes and edges have different types in heterogeneous graphs. Types for nodes and edges play important roles in heterogeneous graphs and should be further considered.

- Determining the node and edge features to accurately and efficiently map the low-momentum tau data to graph data structure.

- **Week 4-Week 5: (June 28 – July 12):**
  Determining GCN Architecture and Designing Loss Function

  - For node level classification of tau particles, I will implement state of the art GCN algorithms which performs node classification to categorize nodes into several classes, and node regression predicts a continuous value for each node. Node clustering aims to partition the nodes into several disjoint groups, where similar nodes should be in the same group.

Evaluation Period (July 12 – July 16)

- **Week 6-Week 8: (July 12 – August 2):**
  Building model using computational modules

  - Propagation Module:

    In propagation modules, the convolution operator and recurrent operator are usually used to aggregate information from neighbours while the skip connection operation is used to gather information from historical representations of nodes and mitigate the over-smoothing problem.

  - Sampling Module:

    Sampling modules are usually needed to conduct propagation on graphs. The sampling module is usually combined with the propagation module.

- Pooling Module:
  Pooling modules are needed to extract information from nodes.

- **Week 9-Week 10: (August 2 – August 16):**
  Migrating the model to GPU

  - Migrating the graph to a GPU by passing the node and edge data as GPU tensors during construction.
  - Use APIs provided by DGL and PyTorch Geometric to copy the graph to a GPU, which copies the graph structure as well as the feature to the given device.
  - Test and benchmarking of inference on GPUs.
  - In this period, I will be testing the model and if good results are obtained. I will be discussing it with my mentor to further expand the model and API.
  - Documentation will be revised and modified as per the comments from my mentors.

Coding Period Ends

Apart from the above-mentioned schedule, I will be:

- Pushing code to my fork daily so that my mentors can evaluate and keep track of whatever work I am doing.
- Blogging every week about the progress and related experiences in the said week so that mentors and others can get an overall summary of my week's work.
- Sending a detailed Pull Request to the main master branch as soon as the code is ready and cleaned-up. I will make sure it is well-documented and has comments wherever necessary.
- Maintaining constant communication with my mentors and updating them about the work done regularly.

❖ *GSoC:*

- **Have you participated previously in GSoC? When? Under which project?**

  No, I have not participated in GSoC before. This is the first time I am participating in GSoC.

- **Are you also applying to other projects?**

  No, I am not applying for any other project in any other organization.

- **Commitments**

  I have my end-semester practical exams from around May 10 (tentative due to Coronavirus Pandemic) and would end before the coding period starts so it will not be a problem. Around a week into August, I would have my first in-semester exam for a week so I will not be available around that period. I will compensate for that period before-hand. Even if there is a change in the exam timetables, I will notify my mentors and shift my schedule accordingly without hampering my progress in the project.

- **Eligibility**

  I am eligible to participate in the Google Summer of Code. For any queries, clarifications or further explanations, feel free to contact me at l’

  .

****