

|  |  |  |  |
| --- | --- | --- | --- |
| (a) LinkedIn connections | nodes: vertices |  | (b) Citation by authors |
| edges: links | - symmetric (a)  - directed (b) |
| graph: network | - Homogeneous (c)  - Heterogeneous (d) |
| (c) Homogeneous | - a social network is a graph consisting of people and their connections[[1]](#footnote-1) => (c)  - the graph encoding a marketplace will have buyer, seller, and product nodes that are connected via wants-to-buy, has-bought, is-customer-of, and is-selling edges => (d) | | (d) Heterogeneous |
|  |  | | Aij=1, is connected to |
| **Task 1: Node Classification (Transductive) [[2]](#footnote-2)**  - Features of all nodes in a graph are provided.  - Some node labels are provided.  - Goal: Predict labels of unknown nodes. | Does this person smoke? | | |
| **Task 2: Node Classification (Inductive)[[3]](#footnote-3)**  - Features of all nodes in one or more graphs are provided.  - Some or all node labels in those graphs are also provided.  - Goal: Predict labels of unseen nodes. | Is this molecule a suitable drug? | | |
| **Task 3: Link Prediction[[4]](#footnote-4)**  - Graph with missing edges is provided.  - Nodes may have labels/features.  - Edges may have labels/features  - Goal: Predict missing edges (possibly along with labels) | Next Netflix video? | | |

**NetworkX package in Python**

https://networkx.org/documentation/stable/tutorial.html

import networkx as nx

import matplotlib.pyplot as plt

#G = nx.Graph #Do thi vo huong

G = nx.DiGraph() #Do thi co huong

#Them cac node vao do thi

G.add\_nodes\_from(

[

(0,{"color": "blue", "size": 650}),

(1,{"color": "green", "size": 650}),

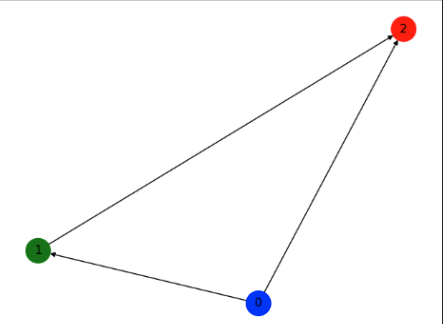
(2,{"color": "red", "size": 650}),

]

)

G.add\_edges\_from(

[

 (0,1),

(0,2),

(1,2)

]

)

#in ra man hinh danh sach cac node

for node in G.nodes(data=True):

print(node)

#ve cac node ra man hinh

node\_color = nx.get\_node\_attributes(G,"color").values()

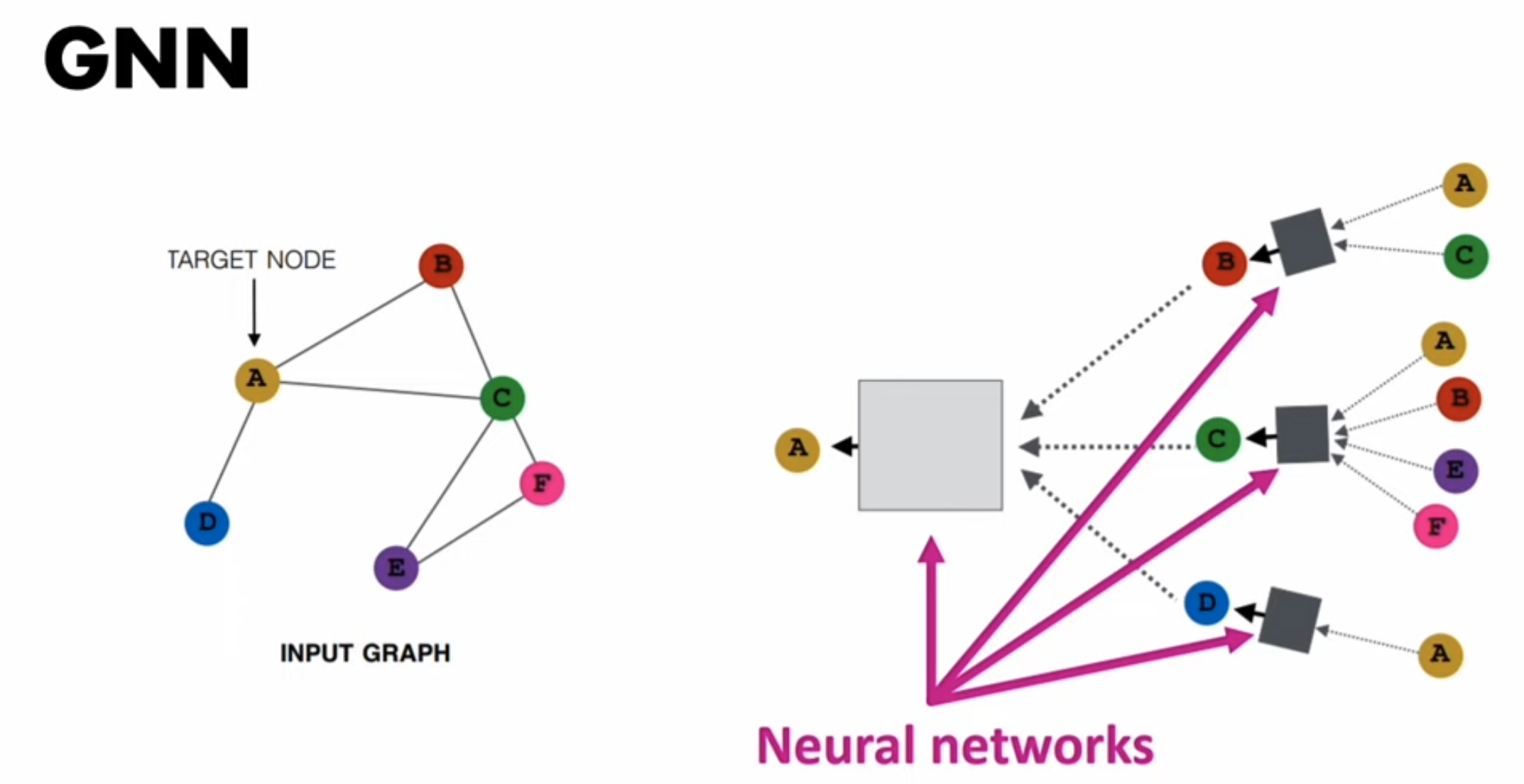
colors = list(node\_color)

node\_size = nx.get\_node\_attributes(G,"size").values()

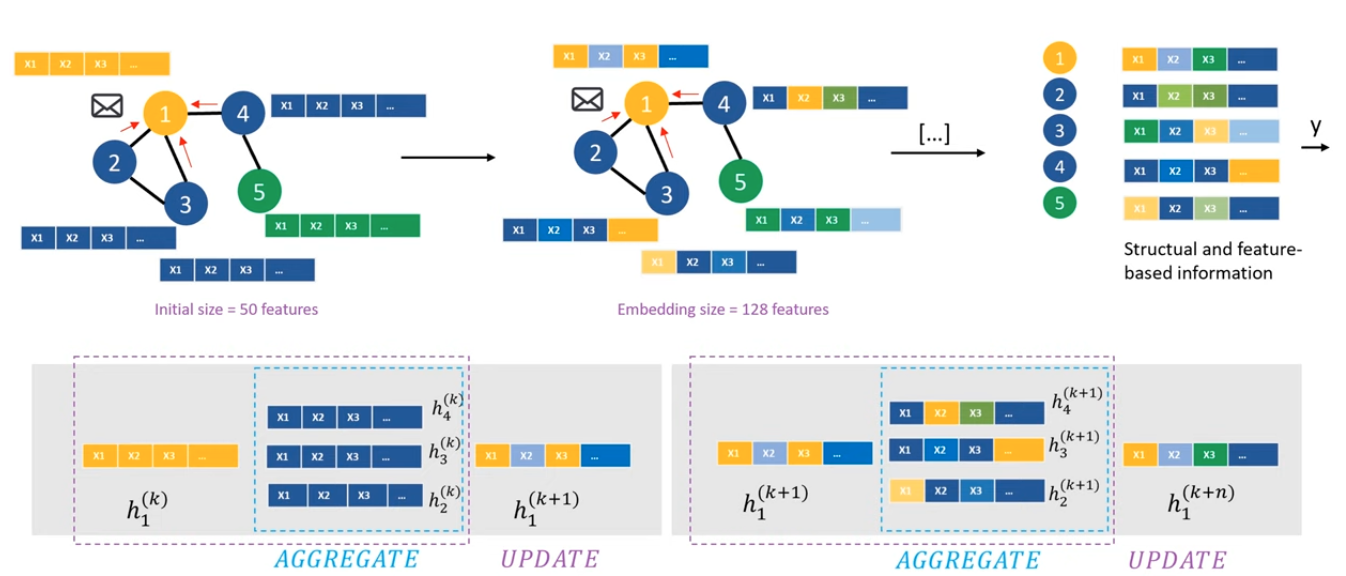
sizes = list(node\_size)

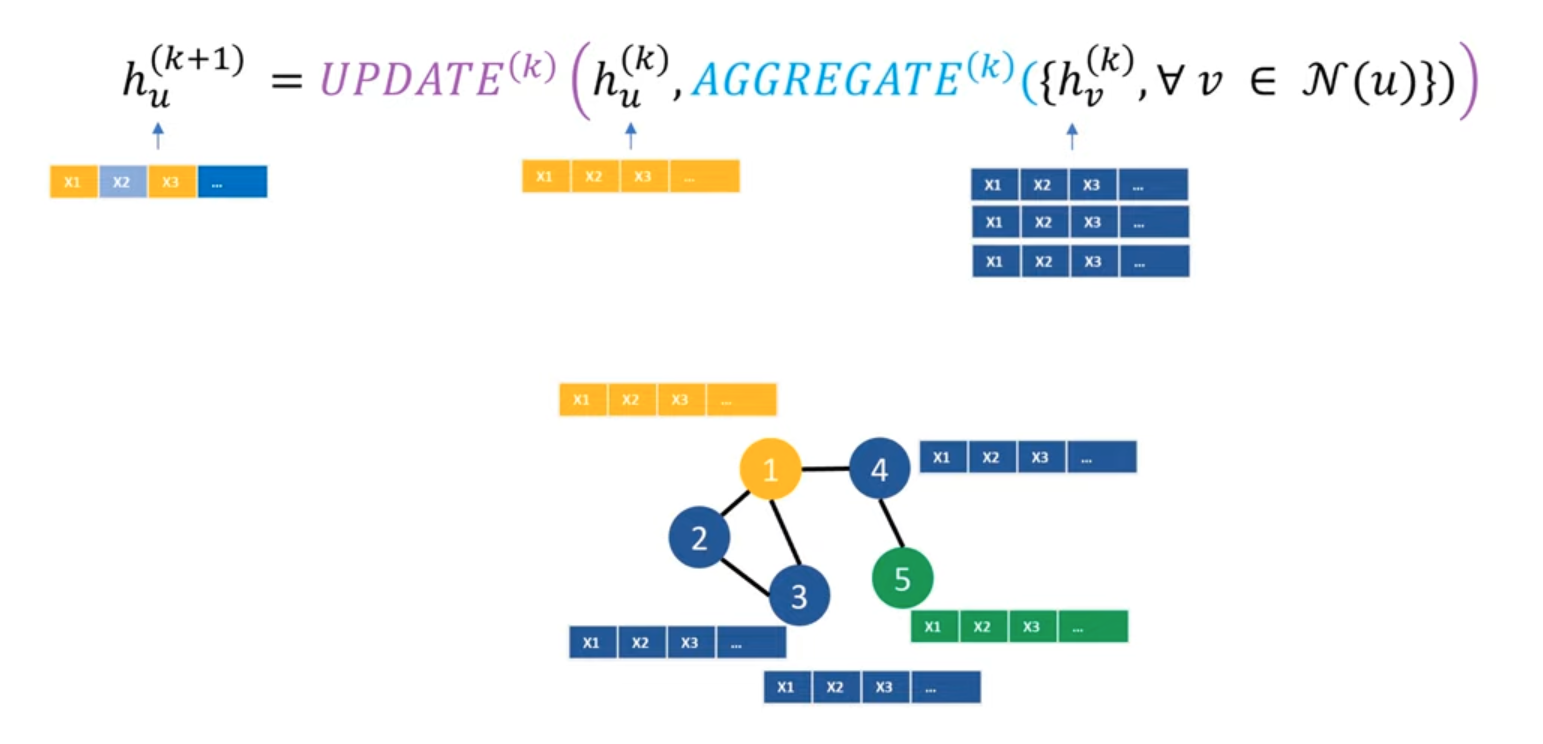
nx.draw(G,with\_labels = True, node\_color=colors, node\_size=sizes)

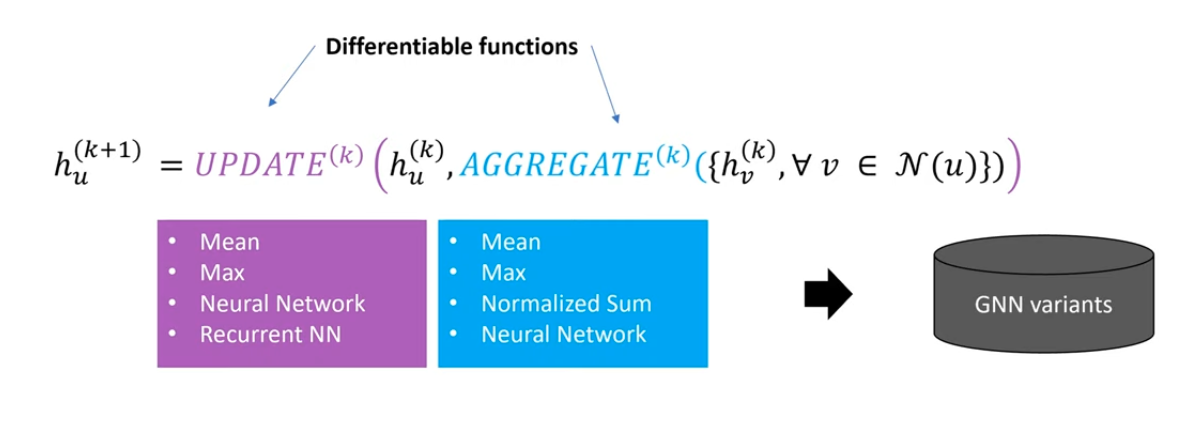
plt.waitforbuttonpress()

[[5]](#footnote-5)

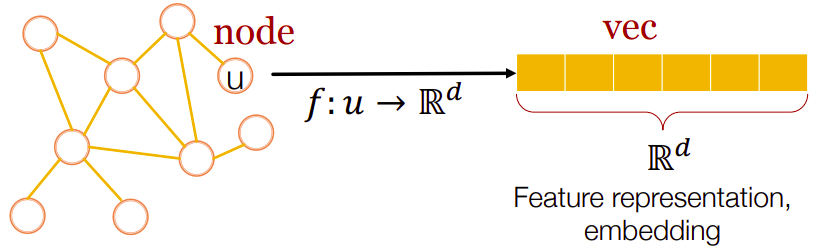












**Build Simple GCN**

Planetoid dataset: https://graphsandnetworks.com/the-cora-dataset/

The Cora dataset consists of 2708 scientific publications classified into one of seven

classes. The citation network consists of 5429 links. Each publication in the dataset is

described by a 0/1-valued word vector indicating the absence/presence of the

corresponding word from the dictionary. The dictionary consists of 1433 unique words.

The Cora dataset consists of Machine Learning papers. These papers are classified into one of the following seven classes:

Case\_Based

Genetic\_Algorithms

Neural\_Networks

Probabilistic\_Methods

Reinforcement\_Learning

Rule\_Learning

Theory

The papers were selected in a way such that in the final corpus every paper cites or is cited by atleast one other paper. There are 2708 papers in the whole corpus.

After stemming and removing stopwords we were left with a vocabulary of size 1433 unique words. All words with document frequency less than 10 were removed.

THE DIRECTORY CONTAINS TWO FILES:

The **.content** file contains descriptions of the papers in the following format:

**<paper\_id> <word\_attributes>+ <class\_label>**

The first entry in each line contains the unique string ID of the paper followed by binary values indicating whether each word in the vocabulary is present (indicated by 1) or absent (indicated by 0) in the paper. Finally, the last entry in the line contains the class label of the paper.

The **.cites** file contains the citation graph of the corpus. Each line describes a link in the following format:

**<ID of cited paper> <ID of citing paper>**

Each line contains two paper IDs. The first entry is the ID of the paper being cited and the second ID stands for the paper which contains the citation. The direction of the link is from right to left. If a line is represented by "paper1 paper2" then the link is "paper2->paper1".

*Nodes (Papers):* Each node in the graph represents an academic paper. Each paper is characterized by a bag-of-words representation of its text content.

*Edges (Citations):* The edges in the graph represent citations between papers. If paper A cites paper B, there will be a directed edge from paper A to paper B in the graph.

*Node Features:* Each paper is associated with a feature vector that represents its content. In the Cora dataset, these features are typically the presence or absence of certain words in the paper's abstract. Each feature indicates whether a specific word appears in the abstract of the paper.

*Node Labels:* Each paper is assigned to one of several pre-defined categories based on its topic. These categories include Computer Science subfields such as Machine Learning, Neural Networks, Databases, etc. In the semi-supervised learning setting, only a small subset of papers have known labels, while the majority of papers are unlabeled.

'Case\_Based',

'Genetic\_Algorithms',

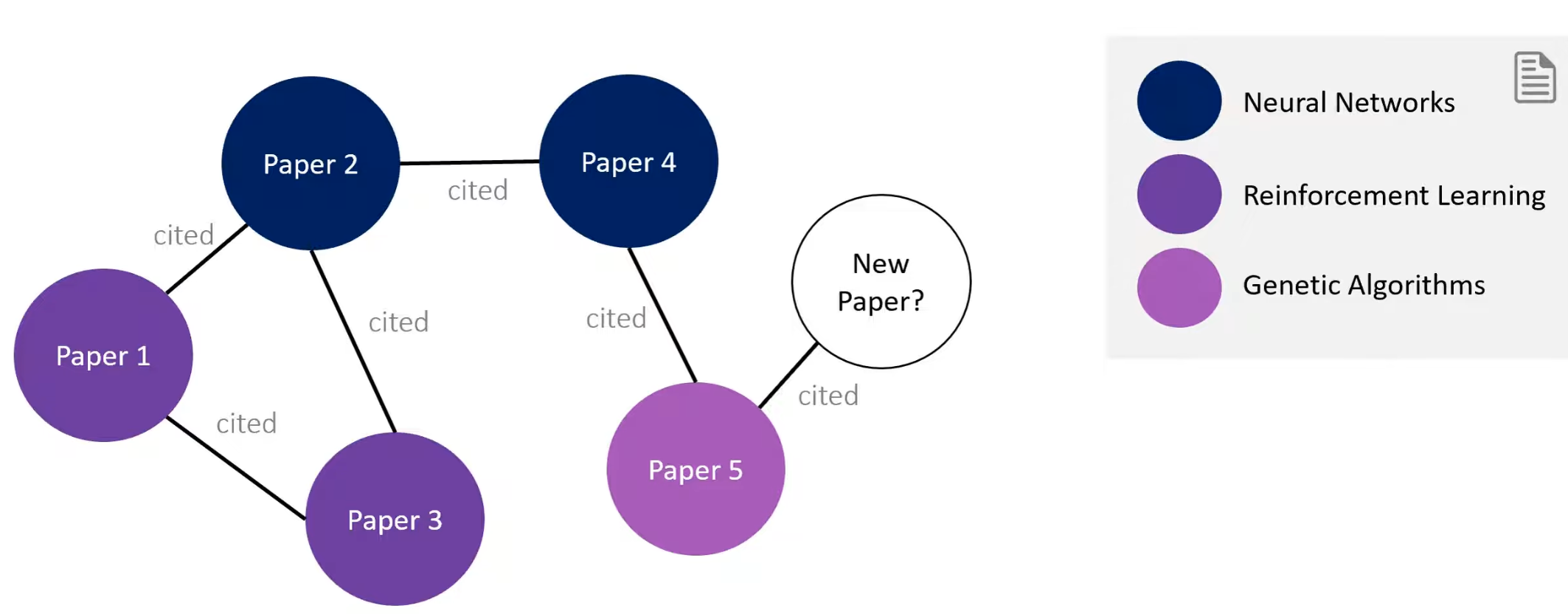
'Neural\_Networks',

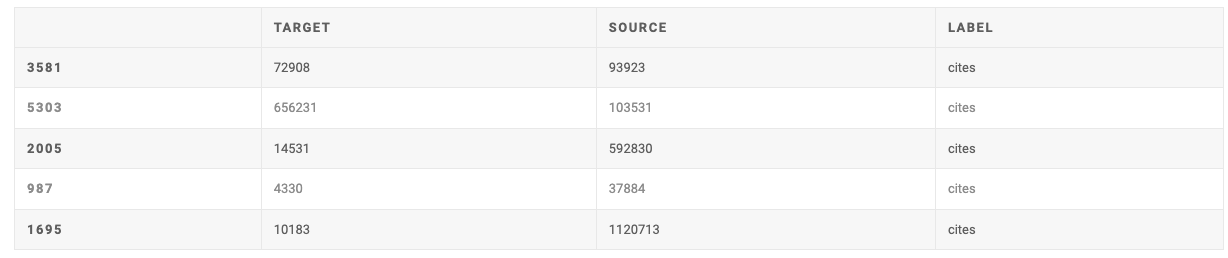
'Probabilistic\_Methods',

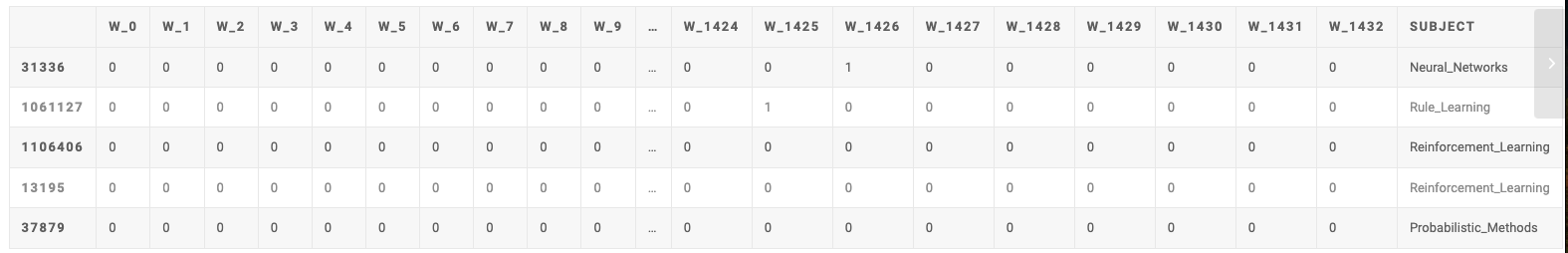
'Reinforcement\_Learning',

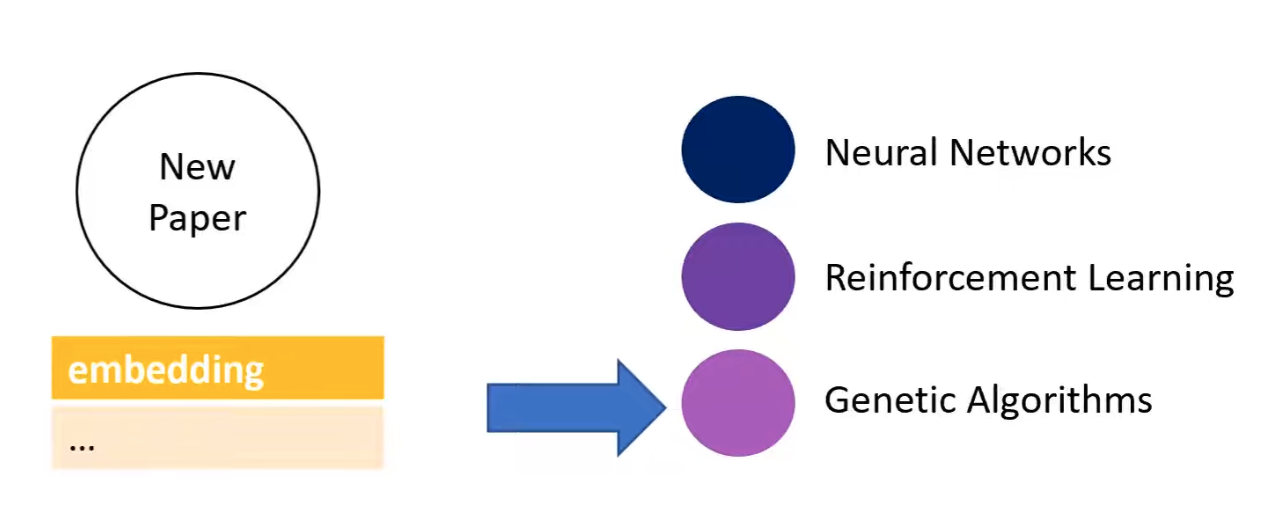
'Rule\_Learning',

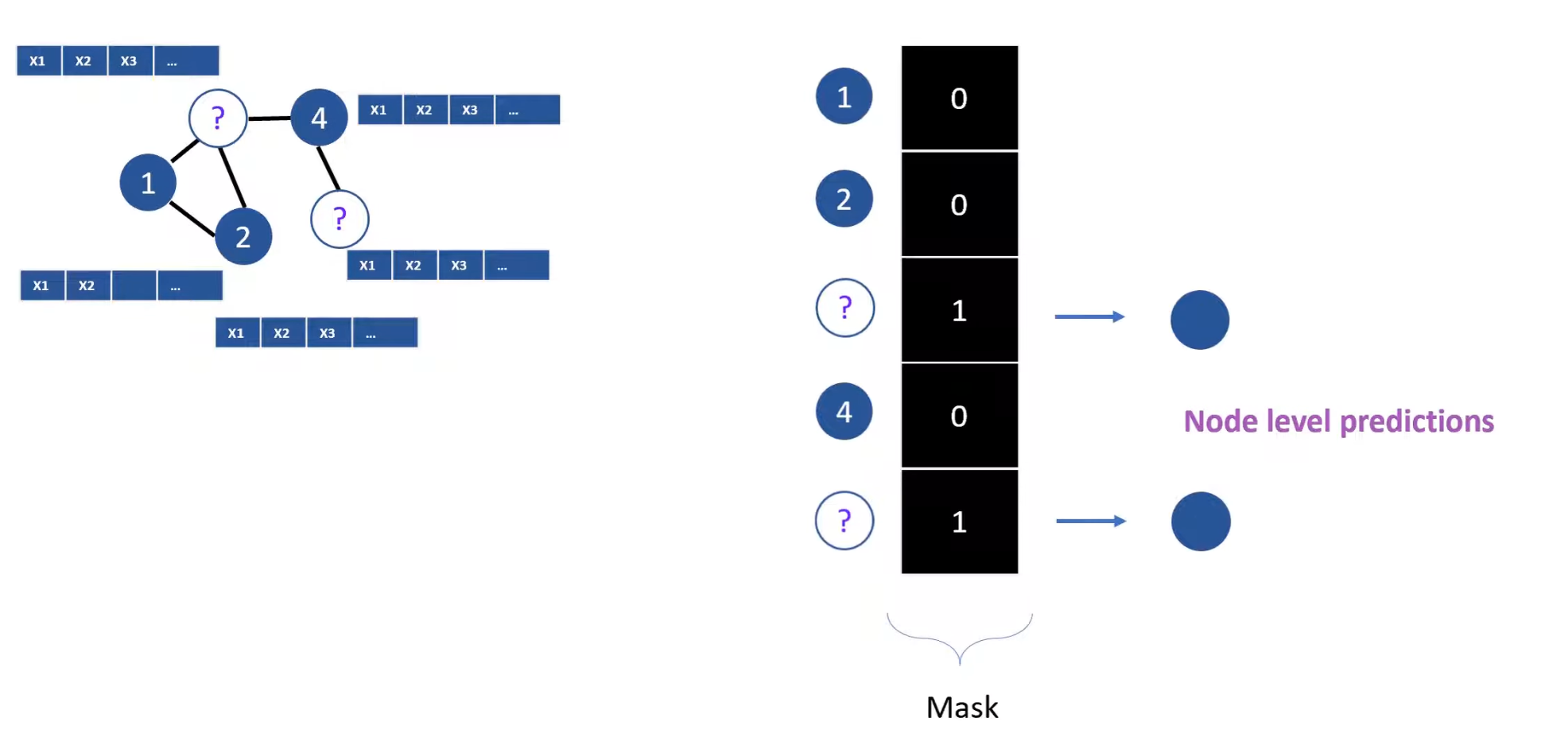
'Theory'











A typical ML challenges with this dataset in mind:

* **label prediction**: predict the subject of a paper (node) on the basis of the surrounding node data and the structure of the graph
* **edge prediction**: given node data, can one predict the papers that should be cited?

#https://graphsandnetworks.com/the-cora-dataset/  
import networkx as nx  
import os  
import pandas as pd  
import matplotlib.pyplot as plt  
  
#visualize dataset  
def visualize():  
 data\_dir= os.path.expanduser("./dataset/")  
 edgelist = pd.read\_csv(os.path.join(data\_dir, "cora.cites"), sep='\t', header=None, names=["target", "source"])  
 edgelist["label"] = "cites"#adding new column named cites  
 edgelist.sample(frac=1).head(5)#select all rows, effectively shuffling the rows randomly  
 G=nx.Graph()  
 G = nx.from\_pandas\_edgelist(edgelist, edge\_attr="label")  
 nx.set\_node\_attributes(G, "paper", "label")  
  
 # in ra man hinh danh sach cac node  
 for node in G.nodes(data=True):  
 print(node)  
  
 # ve cac node ra man hinh  
 plt.figure(figsize=(10, 8))  
 pos = nx.spring\_layout(G) # Layout algorithm for positioning nodes  
 #nx.draw(G, pos, with\_labels=True, node\_color='skyblue', node\_size=1500, edge\_color='black', linewidths=1, font\_size=10)  
 nx.draw(G, pos, node\_color='skyblue', node\_size=10, edge\_color='black', linewidths=1,  
 font\_size=10)  
 plt.title("Cora Citation Network")  
 plt.show()  
  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 visualize()



**Install package**

https://files.pythonhosted.org/packages/2b/52/e6d298d328858aaebf91ca78d81195e3ccaa99ba3b33b0ffc0af5ec0c86d/torch\_geometric-2.5.2.tar.gz

py -m pip install torch\_geometric-2.5.2.tar.gz

**Setup in Terminal/Conda first**

**Build model:**

import torch  
from torch.nn import Linear  
import torch.nn.functional as F  
from torch\_geometric.nn import GCNConv #GATConv  
  
class GCN(torch.nn.Module):  
 def \_\_init\_\_(self, hidden\_channels):  
 super(GCN, self).\_\_init\_\_()  
 torch.manual\_seed(42)  
  
 # Initialize the layers  
 self.conv1 = GCNConv(dataset.num\_features, hidden\_channels)  
 self.conv2 = GCNConv(hidden\_channels, hidden\_channels)  
 self.out = Linear(hidden\_channels, dataset.num\_classes)  
  
 def forward(self, x, edge\_index):  
 # First Message Passing Layer (Transformation)  
 x = self.conv1(x, edge\_index)  
 x = x.relu()  
 x = F.dropout(x, p=0.5, training=self.training)  
  
 # Second Message Passing Layer  
 x = self.conv2(x, edge\_index)  
 x = x.relu()  
 x = F.dropout(x, p=0.5, training=self.training)  
  
 # Output layer  
 x = F.softmax(self.out(x), dim=1)  
 return x

**from** torch\_geometric.data **import** Data

data = Data(x=x, edge\_index=edge\_index, ...)

*# Add additional arguments to `data`:*

data.train\_idx = torch.tensor([...], dtype=torch.long)

data.test\_mask = torch.tensor([...], dtype=torch.bool)

*# Analyzing the graph structure:*

data.num\_nodes

>>> 23

data.is\_directed()

>>> False

*# PyTorch tensor functionality:*

data = data.pin\_memory()

data = data.to('cuda:0', non\_blocking=True)

1. https://docs.dgl.ai/en/0.8.x/guide/graph-basic.html [↑](#footnote-ref-1)
2. ,3,4 https://www.youtube.com/watch?v=vSMUv5YAZ00&list=PLB1nTQo4\_y6sfLtCrGAKG\_l7xOHjtYqBk&index=3&ab\_channel=LLMsExplained-AggregateIntellect-AI.SCIENCE [↑](#footnote-ref-2)
3. [↑](#footnote-ref-3)
4. [↑](#footnote-ref-4)
5. [Node Classification on Knowledge Graphs using PyTorch Geometric (youtube.com)](https://www.youtube.com/watch?v=ex2qllcVneY&list=PLV8yxwGOxvvoNkzPfCx2i8an--Tkt7O8Z&index=4&ab_channel=DeepFindr) [↑](#footnote-ref-5)