Member-only story

Node Classification using Graph Convolutional Neural Network

Node Classification on Cora Dataset in PyTorch using GCN

Renu Khandelwal · Follow

5 min read · Jun 29, 2022

20

Prerequisites:

Graph Basics and Application of Graph

Graph Representational Learning

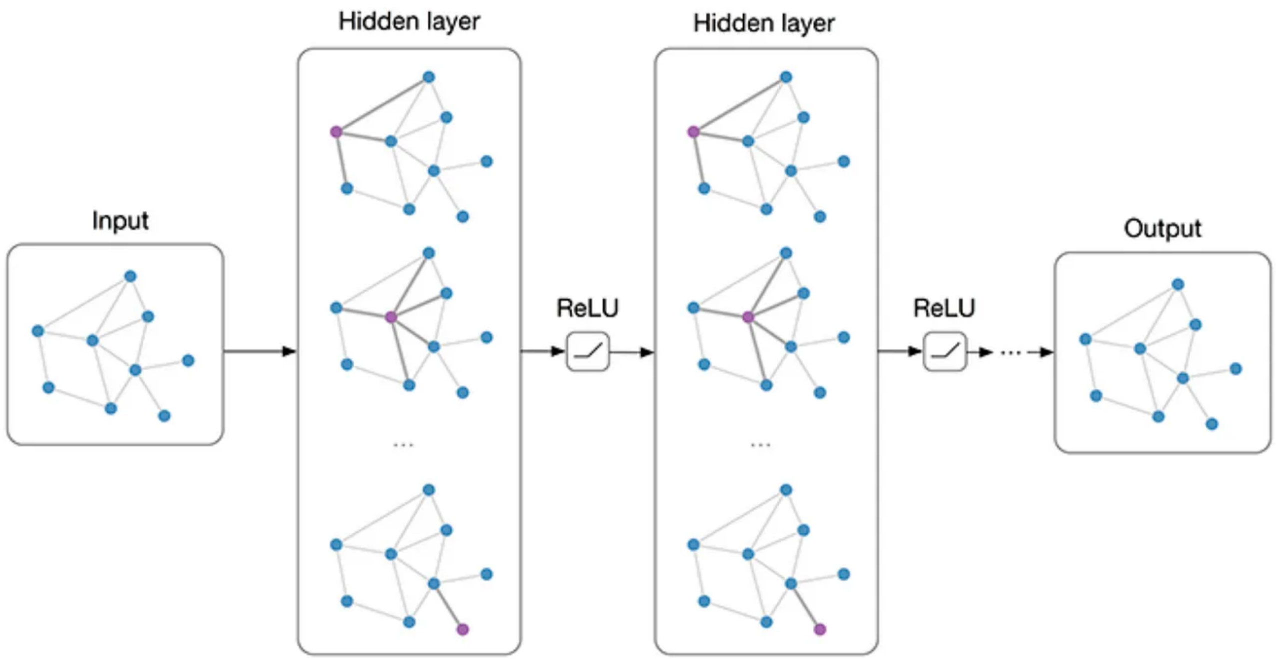
Graph Neural Networks: A Deep Neural Network for Graphs

Dataset: This article uses Cora Dataset, consisting of 2708 scientific publications classified into one of seven different classes. The citation network consists of 5429 links.

Objective: Node classification using GCN to accurately predict the subject of a paper given its words and citation network using PyTorch geometric

Graph Convolutional Neural Network(GCN) model is a framework of spectral graph convolutions applying a generalization of convolutions to non-Euclidean data.

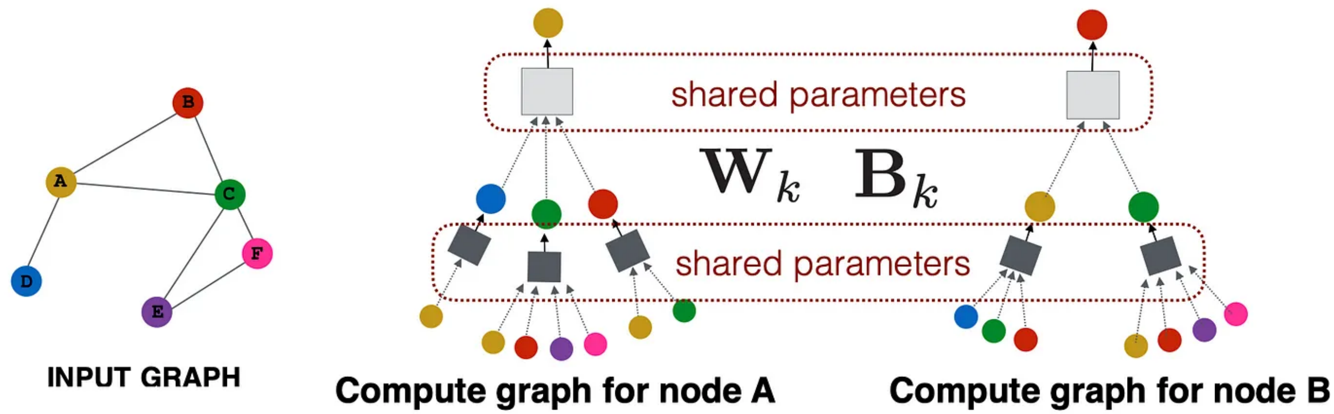
GCNs are similar to convolutions applied to images as they generalize the graph data's convolution operations. The filter parameters are shared over all locations in the graph. GCN is based on graph convolutions built by stacking multiple convolutional layers, and a point-wise non-linearity function follows each layer.



source: http://tkipf.github.io/graph-convolutional-networks/

In this example, you will classify the scientific papers in a citation graph where labels are only available for a small subset of nodes, and GCN must predict the correct label for the node.

The key idea of GCN is to generate node embeddings based on local network neighborhoods. Nodes aggregate information from their neighbors using neural networks. As a result, every node defines a computation graph based on its neighborhood by averaging neighbor messages and applying a neural network, as shown below.



source:http://snap.stanford.edu/class/cs224w-2019/slides/08-GNN.pdf

Exploring the Dataset

Open the tar files inside the .tgz files

```
import urllib.request
import tarfile

coraTarFile = 'https://linqs-data.soe.ucsc.edu/public/lbc/cora.tgz'
tarfiles = urllib.request.urlopen(coraTarFile)
zip_file = tarfile.open(fileobj=tarfiles, mode="r|gz")
for tarinfo in zip_file:
    print(tarinfo.name, "is", tarinfo.size, "bytes in size and is ",
end="")
    if tarinfo.isreg():
        print("a regular file.")

    elif tarinfo.isdir():
        print("a directory.")
    else:
```

```
        print("something else.")
zip_file.close()
```

The dataset has two files, **cora.cites** and **cora.content**

```
data_dir = os.path.join('.', "cora")

citations = pd.read_csv(
    os.path.join(data_dir, "cora.cites"),
    sep="\t",
    header=None,
    names=["target", "source"],
)

papers = pd.read_csv(
    os.path.join(data_dir, "cora.content"),
    sep="\t",
    header=None,
    names=["paper_id"] + [f"term_{idx}" for idx in range(1433)] +
["subject"],
)
```

cora.cities contain the citation records with two columns: `cited_paper_id` (target) and `citing_paper_id` (source).

The **cora.content** includes the paper content records as well as the subject

Finding unique values for the subject of the papers and their counts

```
print(papers.subject.value_counts())
```

```
Neural_Networks             818
Probabilistic_Methods       426
Genetic_Algorithms          418
Theory                      351
Case_Based                  298
Reinforcement_Learning      217
Rule_Learning               180
Name: subject, dtype: int64
```

This code uses **Planetoid**, **a novel Graph-based semi-supervised learning framework (Predicting Labels And Neighbors with Embeddings Transductively Or Inductively from Data).** Planetoid contains the citation network datasets Cora, CiteSeer and PubMed graph datasets(MIT license).

The Planetoid dataset **is a single graph that** holds

- **x the node features,** its dimension is the number of nodes times feature dimension

- **edge_index,** which contains the edge list
- **y the ground truth to the class labels** for each node; in the case of the Cora dataset, it is the **classification of the papers.**
- **Three masks: train_mask, val_mask, and test_mask,** which denote nodes for training, validation, and test, respectively.

Open the tarfile to view the contents and normalize the tensor image with a mean and standard deviation.

```
dataset = Planetoid(coraTarFile, ‘cora’,
transform=T.NormalizeFeatures())
data = dataset[0]
data
```

```
Data(x=[2708, 1433], edge_index=[2, 10556], y=[2708], train_mask=[2708], val_mask=[2708], test_mask=[2708])
```

Printing the unique classes, nodes, edge features, and the shape of the dataset

```
print(dataset)
print("number of graphs:\t\t",len(dataset))
```

```
print("number of classes:\t\t",dataset.num_classes)
print("number of classes:\t\t",np.unique(data.y))
print("number of node features:\t",data.num_node_features)
print("number of edge features:\t",data.num_edge_features)
print("X shape: ", data.x.shape)
print("Edge shape: ", data.edge_index.shape)
print("Y shape: ", data.y.shape)
```

```
cora()
number of graphs:                 1
number of classes:                7
number of classes:                [0 1 2 3 4 5 6]
number of node features:          1433
number of edge features:          0
X shape:  torch.Size([2708, 1433])
Edge shape:  torch.Size([2, 10556])
Y shape:  torch.Size([2708])
```

Set the device dynamically

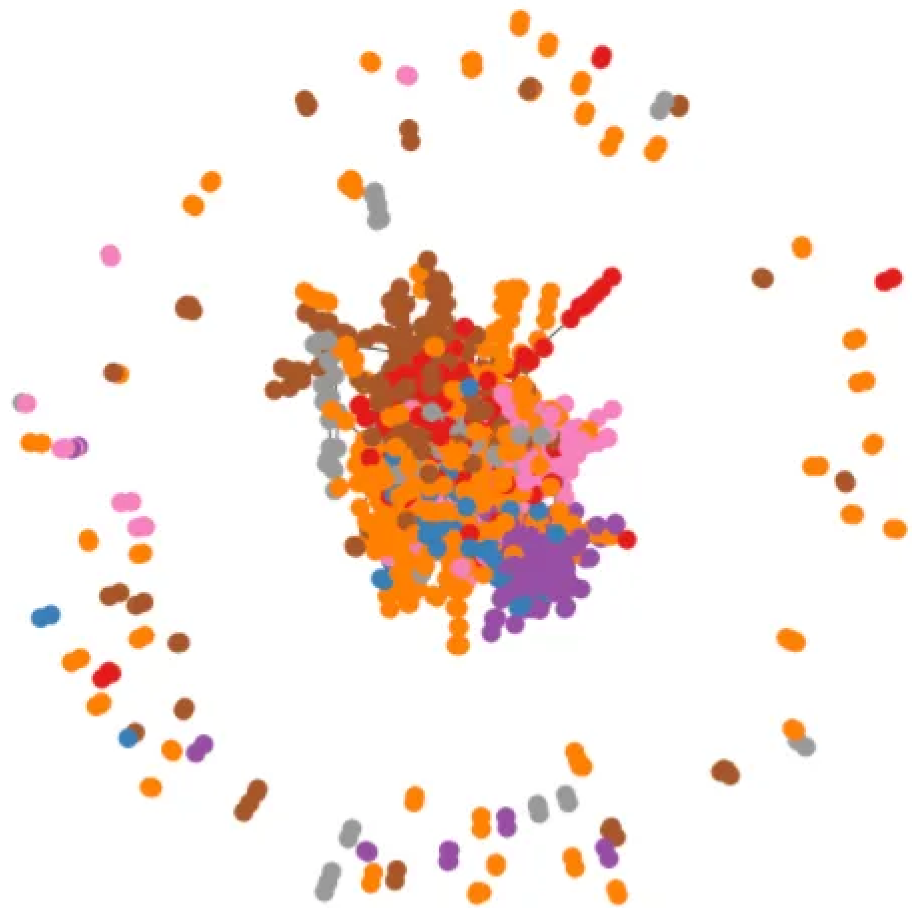
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

Visualize the Citation graph

Convert the Cora graph dataset to the NetworkX graph

```
from torch_geometric.utils.convert import to_networkx
import networkx as nx

plt.figure(figsize=(10, 10))
cora = torch_geometric.data.Data(x=data.x[:500],
edge_index=data.edge_index[:500])
g = torch_geometric.utils.to_networkx(cora, to_undirected=True)
coragraph = to_networkx(cora)
node_labels = data.y[list(coragraph.nodes)].numpy()
nx.draw(g, cmap=plt.get_cmap('Set1'),node_color =
node_labels,node_size=75,linewidths=6)
```



GCN Model

The model uses two GCNConv layers. The first GCNConv layer is followed by a non-linearity ReLU activation function and a Dropout. The result of the first GCNConv layer is fed to the second GCNConv layer. A Softmax is finally applied to get distribution over the number of classes.

The GCN model uses an Adam optimizer with a learning rate of 0.01

```
class GCN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = GCNConv(dataset.num_node_features, 16)
        self.conv2 = GCNConv(16, dataset.num_classes)

    def forward(self, data):
        # x: Node feature matrix
        # edge_index: Graph connectivity matrix

        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)

model = GCN().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01,
weight_decay=5e-4)

print("Graph Convolutional Network (GCN):")
GCN()
```

```
Graph Convolutional Network (GCN):

GCN(
  (conv1): GCNConv(1433, 16)
  (conv2): GCNConv(16, 7)
)
```

Training the GCN Model

Function for computing accuracy

```
# useful function for computing accuracy
def compute_accuracy(pred_y, y):
    return (pred_y == y).sum()
```

Training the model on the training dataset for 200 epochs

```
# train the model
model.train()
losses = []
accuracies = []
for epoch in range(200):
    optimizer.zero_grad()
    out = model(data)

loss = F.nll_loss(out[data.train_mask], data.y[data.train_mask])
    correct = compute_accuracy(out.argmax(dim=1)[data.train_mask],
data.y[data.train_mask])
    acc = int(correct) / int(data.train_mask.sum())
```

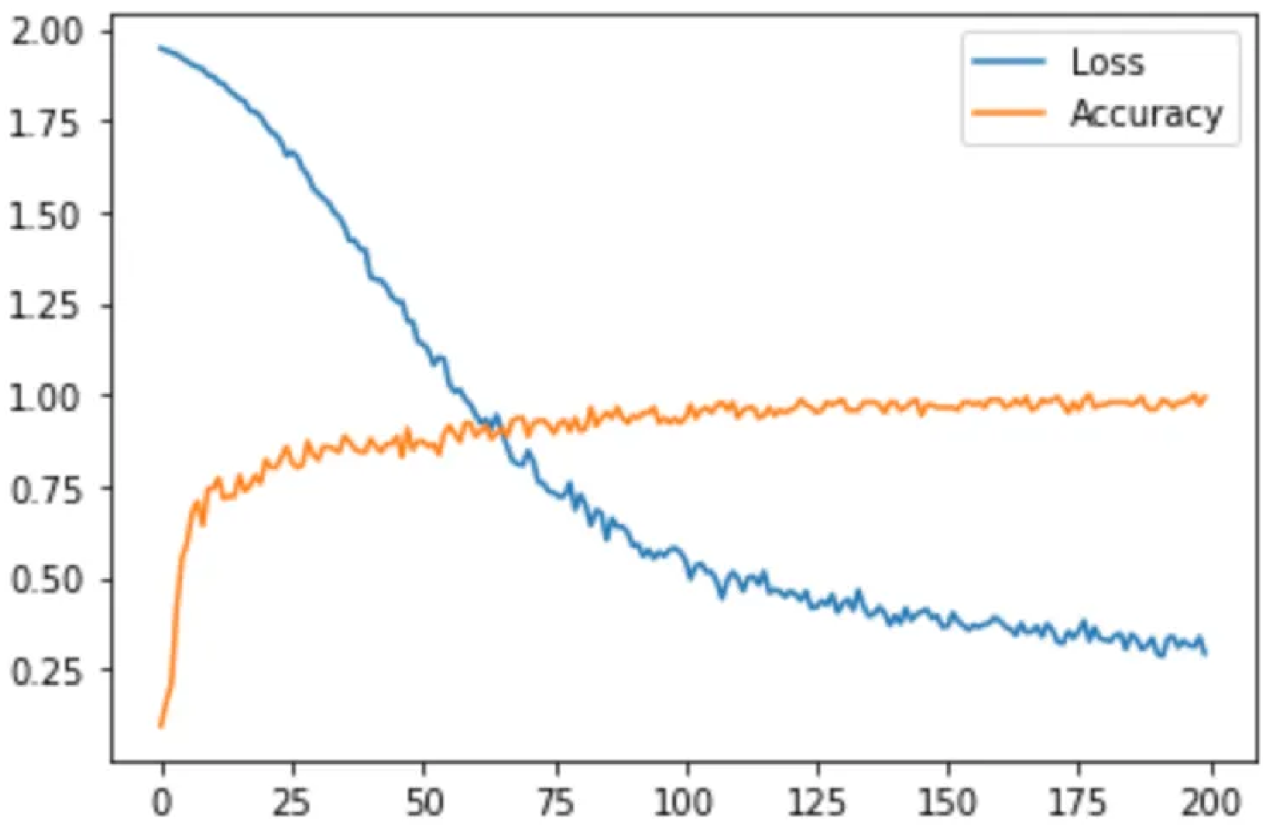
Search Medium

Write

```
loss.backward()
    optimizer.step()
    if (epoch+1) % 10 == 0:
        print('Epoch: {}, Loss: {:.4f}, Training Acc:
{:.4f}'.format(epoch+1, loss.item(), acc))
```

plotting the accuracy and loss during the model training

```
# plot the loss and accuracy
import matplotlib.pyplot as plt
plt.plot(losses)
plt.plot(accuracies)
plt.legend(['Loss', 'Accuracy'])
plt.show()
```



Evaluation of the GCN Model

Evaluate the model on the test dataset

```
# evaluate the model on test set
model.eval()
pred = model(data).argmax(dim=1)
correct = compute_accuracy(pred[data.test_mask],
data.y[data.test_mask])
acc = int(correct) / int(data.test_mask.sum())
print(f'Accuracy: {acc:.4f}')
```

```
Accuracy: 0.8140
```

Full code available on Github

References:

Revisiting Semi-Supervised Learning with Graph Embeddings. Zhilin Yang, William W. Cohen, Ruslan Salakhutdinov. ICML 2016.

How powerful are Graph Convolutional Networks?

Many important real-world datasets come in the form of graphs or networks: social networks, knowledge graphs…

tkipf.github.io

Keras documentation: Node Classification with Graph Neural Networks

Author: Khalid Salama Date created: 2021/05/30 Last modified: 2021/05/30 Description: Implementing a graph neural…

keras.io

Colab Notebooks and Video Tutorials - pytorch_geometric documentation

The Stanford CS224W course has collected a set of graph machine learning tutorial blog posts, fully realized with PyG…

PyTorch-geometric.readthedocs.io

pytorch_geometric/gcn.py at master · pyg-team/pytorch_geometric

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or…

github.com

https://pytorch-geometric.readthedocs.io/en/latest/notes/introduction.html

Pytorch Pytorch Geometric Graph Convolution Network Artificial Intelligence

Cora

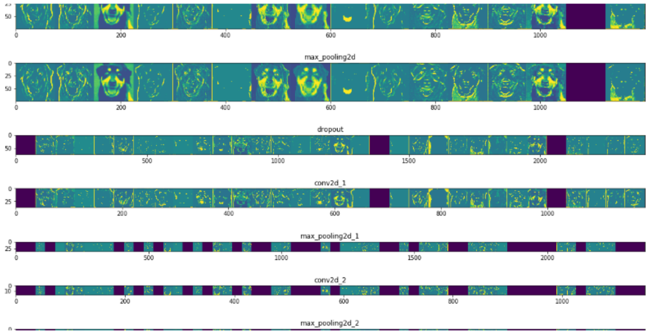
Written by Renu Khandelwal

Follow

6K Followers

A Technology Enthusiast who constantly seeks out new challenges by exploring cutting-edge technologies to make the world a better place!

More from Renu Khandelwal



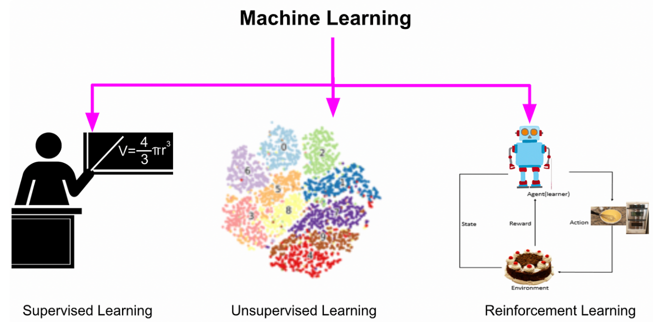
Renu Khandelwal in Towards Data Science

Convolutional Neural Network: Feature Map and Filter…

Learn how Convolutional Neural Networks understand images.

· 8 min read · May 18, 2020

434 6



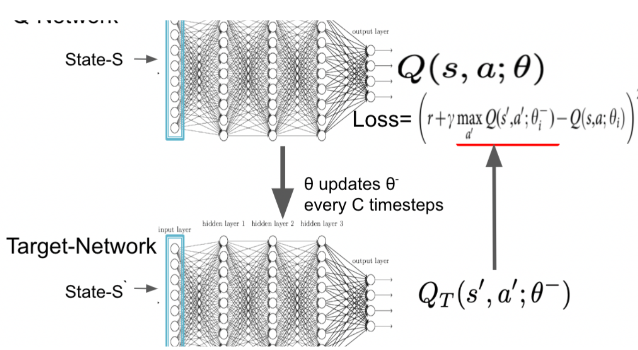
Renu Khandelwal

Supervised, Unsupervised, and Reinforcement Learning

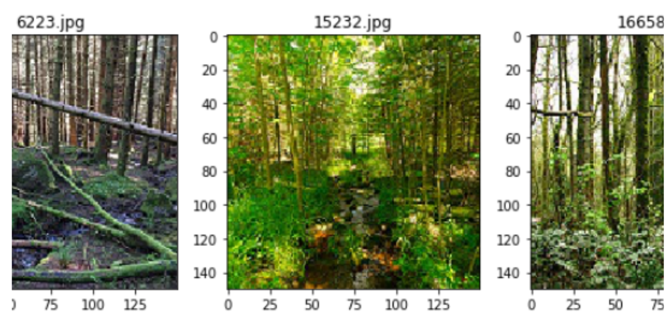
An Intuitive explanation of Supervised, Unsupervised, and Reinforcement learning…

· 5 min read · Jul 20, 2022

168



Renu Khandelwal



Renu Khandelwal in Towards Data Science

Deep Q Learning: A Deep Reinforcement Learning Algorithm

An easy-to-understand explanation of Deep Q-Learning with PyTorch code...

· 11 min read · Jan 12

50

Loading Custom Image Dataset for Deep Learning Models: Part 1

A simple guide to different techniques for loading a custom image dataset into deep...

· 4 min read · Aug 20, 2020

133 8

See all from Renu Khandelwal

Recommended from Medium

AI TutorMas... in Artificial Intelligence in Plain Eng...

Graph Neural Networks—Introduction for Beginners

A Graph Neural Network (GNN) is a type of neural network that is designed to work with...

· 10 min read · Jan 14

424 1

Gareth Paul Jones

PyTorch Debugging 101: Essential Tips for Success

As someone that loves to create machine learning models, debugging is an essential...

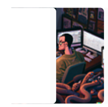
3 min read · Jan 8

27

Lists

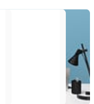
AI Regulation

6 stories · 1 save



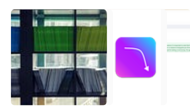
ChatGPT

17 stories · 2 saves



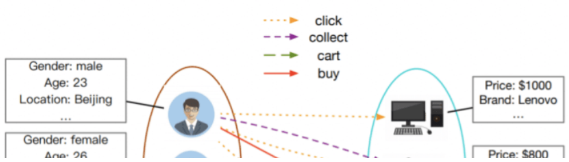
ChatGPT prompts

17 stories · 5 saves



Generative AI Recommended Reading

51 stories · 5 saves



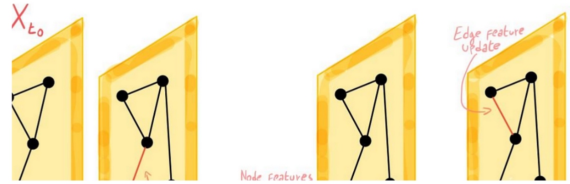
PyTorch Geometric

Link Prediction on Heterogeneous Graphs with PyG

By Jan Eric Lenssen and Matthias Fey

10 min read · Dec 22, 2022

85 3



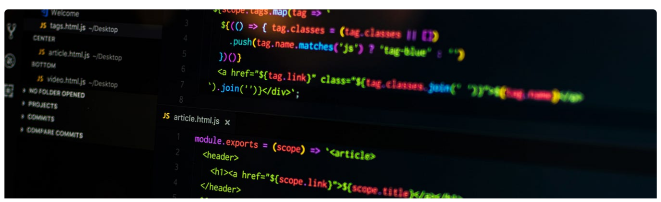
Shenyang(Andy) Huang in Towards Data Science

Temporal Graph Learning in 2023

The story so far

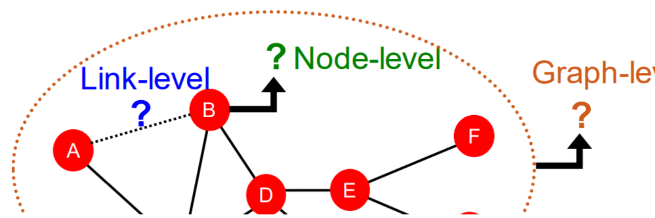
15 min read · Jan 17

482 4



Ifeanyi Nneji in Python in Plain English

Real-Time Image Processing using WebSockets and Flask in Python...



Shweta Gargade

Graph Networks: Traditional Methods to extract features from...

Building a Real-Time Image Processing App with Python and JavaScript

8 min read · Feb 23

47

To get the basics of What is Graph, please read my previous article:...

16 min read · Feb 1

100 1

See more recommendations