Assignment 16

# Results from DL30A.py

## Results using the random graph generator nx.gnm\_random\_graph(30, 50) [ORIGINAL]

Diagram

Description automatically generated

A picture containing sky

Description automatically generated

Chart

Description automatically generated

A picture containing sky

Description automatically generated

A picture containing sky

Description automatically generated

## Chart Description automatically generated

## Results using Karate Club graph nx.karate\_club\_graph() [MODIFIED]

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

A picture containing sky

Description automatically generated

A picture containing sky, accessory

Description automatically generated

## Modified Code

# DL30A.py CS5173/6073 cheng 2023  
# graph convolutional network  
# (7.39) of Hamilton  
# initial random embedding is improved with Adam optimizer  
# Usage: python DL30A.py  
  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torch.optim as optim  
import numpy as np  
import networkx as nx  
import matplotlib.pyplot as plt  
  
G = nx.karate\_club\_graph()  
N = G.number\_of\_nodes()  
A = torch.tensor(nx.to\_numpy\_array(G) + np.identity(N), dtype=torch.float)  
one = torch.ones(len(A))  
d = torch.matmul(one, A)  
d2 = 1.0 / torch.sqrt(d)  
D2 = torch.diag(d2)  
Atilda = torch.matmul(torch.matmul(D2, A), D2)  
  
class Model(nn.Module):  
 def \_\_init\_\_(self, inputsize, hiddensize=16, outputsize=2):  
 super(Model, self).\_\_init\_\_()  
 self.h = nn.Parameter(torch.rand(inputsize, hiddensize))  
 self.w = nn.Parameter(torch.randn(hiddensize, outputsize))  
 self.z = torch.zeros(inputsize, outputsize)  
  
 def forward(self):  
 self.z = torch.matmul(torch.matmul(Atilda, self.h), self.w)  
 return torch.matmul(self.z, self.z.T)  
  
model = Model(N)  
model()  
z2 = model.z.detach().numpy()  
pos = {}  
for i in G.nodes:  
 pos.update({i: z2[i]})  
nx.draw(G, pos)  
plt.show()  
  
optimizer = optim.Adam(model.parameters())  
loss\_fun = nn.CrossEntropyLoss()  
  
for i in range(5):  
 for j in range(100):  
 dec = model()  
 loss = loss\_fun(dec, A)  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
 z2 = model.z.detach().numpy()  
 pos = {}  
 for i in G.nodes:  
 pos.update({i: z2[i]})  
 nx.draw(G, pos)  
 plt.show()

# Results from DL30B.py

## Results using the random graph generator nx.gnm\_random\_graph(30, 50) [ORIGINAL]

A picture containing chart

Description automatically generated

Chart, scatter chart

Description automatically generated

A picture containing sky

Description automatically generated

Chart, radar chart

Description automatically generated

Chart, radar chart

Description automatically generated

A picture containing sky

Description automatically generated

## Results using Karate Club graph **nx.karate\_club\_graph()** [MODIFIED]

Diagram, schematic

Description automatically generated

A close-up of a syringe

Description automatically generated with medium confidence

A picture containing shape

Description automatically generated

A picture containing chart

Description automatically generated

A picture containing chart

Description automatically generated

A picture containing shape

Description automatically generated

## Modified Code

# DL30A.py CS5173/6073 cheng 2023  
# graph neural network  
# (7.44) of Hamilton  
# initial random embedding is improved with Adam optimizer  
# Usage: python DL30A.py  
  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torch.optim as optim  
import numpy as np  
import networkx as nx  
import matplotlib.pyplot as plt  
  
G = nx.karate\_club\_graph()  
N = G.number\_of\_nodes()  
A = torch.tensor(nx.to\_numpy\_array(G) + np.identity(N), dtype=torch.float)  
one = torch.ones(len(A))  
d = torch.matmul(one, A)  
d2 = 1.0 / torch.sqrt(d)  
D2 = torch.diag(d2)  
Atilda = torch.matmul(torch.matmul(D2, A), D2)  
  
class Model(nn.Module):  
 def \_\_init\_\_(self, inputsize, hiddensize=16, outputsize=2):  
 super(Model, self).\_\_init\_\_()  
 self.X = torch.randn(inputsize, hiddensize)  
 self.mlp10 = nn.Linear(hiddensize, hiddensize)  
 self.mlp11 = nn.Linear(hiddensize, hiddensize)  
 self.mlp20 = nn.Linear(hiddensize, hiddensize)  
 self.mlp21 = nn.Linear(hiddensize, outputsize)  
 self.z = torch.zeros(inputsize, outputsize)  
  
 def forward(self):  
 h1 = F.relu(self.mlp10(self.X))  
 h2 = self.mlp11(h1)  
 h3 = torch.matmul(Atilda, h2)  
 h4 = F.relu(self.mlp20(h3))  
 self.z = self.mlp21(h4)  
 return torch.matmul(self.z, self.z.T)  
  
model = Model(N)  
model()  
z2 = model.z.detach().numpy()  
pos = {}  
for i in G.nodes:  
 pos.update({i: z2[i]})  
nx.draw(G, pos)  
plt.show()  
  
optimizer = optim.Adam(model.parameters())  
loss\_fun = nn.CrossEntropyLoss()  
  
for i in range(5):  
 for j in range(100):  
 dec = model()  
 loss = loss\_fun(dec, A)  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
 z2 = model.z.detach().numpy()  
 pos = {}  
 for i in G.nodes:  
 pos.update({i: z2[i]})  
 nx.draw(G, pos)  
 plt.show()