

96 lines - 31 Removals

```

1 import json
2 import torch
3 import torch.nn as nn
4 import numpy as np
5 import torch_geometric.transforms as T
6 from datasets.load_datasets import get_dataset, get_data
  loader
7
8 from models import GCN
9
10 def evaluate(dataloader, model, loss_fc):
11     acc = []
12     loss_list = []
13     model.eval()
14     with torch.no_grad():
15         for data in dataloader:
16             logit = model(data)
17             loss = loss_fc(logit, data.y)
18             prediction = torch.argmax(logit, -1)
19             loss_list.append(loss.item())
20             acc.append((prediction == data.y).numpy())
21     return np.concatenate(acc, axis=0).mean(), np.average(
  e(loss_list)
22
23 if __name__ == '__main__':
24     with open("configs.json") as config_file:
25         configs = json.load(config_file)
26         dataset_name = configs.get("dataset_name").get(
  ("graph")
27
28     epochs = 5000
29     pooling = {'mutagenicity': ['max', 'mean', 'sum'],
30               'ba_2motifs': ['max'],
31               'bbbp': ['max', 'mean', 'sum']}
32
33     early_stop = 100
34     loop = True
35
36     if dataset_name == 'ba_2motifs':
37         loop = False
38
39     normalize = T.NormalizeFeatures()
40     dataset = get_dataset(dataset_dir="./datasets", data
  set_name=dataset_name)
41     dataset.data.x = dataset.data.x.float()
42     dataset.data = normalize(dataset.data)
43
44     data_loader = get_dataloader(dataset, batch_size=32,
  random_split_flag=True,
45                               data_split_ratio=[0.8,
46                               0.1, 0.1], seed=2)
47
48     model = GCN(n_feat=dataset.num_node_features,
49               n_hidden=20,
50               n_class=dataset.num_classes,
51               pooling=pooling[dataset_name],
52               loop=loop)

```

93 lines + 28 Additions

```

1 import json
2 import torch
3 import torch.nn as nn
4 import numpy as np
5 import torch_geometric.transforms as T
6 from MalNet_Tiny import MalNetTiny
7
8 from torch_geometric.loader import DataLoader
9
10 from models import GCN
11 from tqdm import tqdm
12
13 def evaluate(dataloader, model, loss_fc):
14     acc = []
15     loss_list = []
16     model.eval()
17     with torch.no_grad():
18         for data in dataloader:
19             logit = model(data)
20             loss = loss_fc(logit, data.y)
21             prediction = torch.argmax(logit, -1)
22             loss_list.append(loss.item())
23             acc.append((prediction == data.y).numpy())
24     return np.concatenate(acc, axis=0).mean(), np.average(
  (loss_list)
25
26 if __name__ == '__main__':
27     with open("configs.json") as config_file:
28         configs = json.load(config_file)
29         dataset_name = configs.get("dataset_name").get("g
  raph")
30
31     epochs = 5000
32     pooling = {'malnet_tiny': ['max', 'mean', 'sum']}
33
34     early_stop = 100
35     transform = T.Compose([T.RemoveIsolatedNodes(), T.Add
  SelfLoops(), T.AddLaplacianEigenvectorPE(5, attr_name
  ='x'), T.AddRandomWalkPE(20, attr_name='x'), T.ToSparseTensor()])
36
37     normalize = T.NormalizeFeatures()
38     dataset = MalNetTiny(root='./datasets', transform=tra
  nsform)
39     for i, graph in enumerate(tqdm(dataset, total=3500)):
40         dataset[i].x = torch.cat([graph.x.to("cpu"), grap
  h.random_walk_pe.to("cpu")], dim=1)
41     data_loader = DataLoader(dataset, batch_size=32, shuf
  fle=True)
42
43     model = GCN(n_feat=dataset.num_node_features,
44               n_hidden=20,
45               n_class=dataset.num_classes,
46               pooling=pooling[dataset_name][0],
47               loop=True) # Adjust as needed

```

```

50 optimizer = torch.optim.Adam(model.parameters(), lr=
5e-3)
51 loss_fc = nn.CrossEntropyLoss()
52 model_file = './src/' + dataset_name + '.pt'
53
54 model.train()
55 early_stop_count = 0
56 best_acc = 0
57 best_loss = 100
58 for epoch in range(epochs):
59     acc = []
60     loss_list = []
61     model.train()
62     for i, data in enumerate(data_loader['train']):
63         logit = model(data)
64         loss = loss_fc(logit, data.y)
65         optimizer.zero_grad()
66         loss.backward()
67         nn.utils.clip_grad_norm_(model.parameters(),
max_norm=2.0)
68         optimizer.step()
69
70         prediction = torch.argmax(logit, -1)
71         loss_list.append(loss.item())
72         acc.append((prediction == data.y).numpy())
73
74         eval_acc, eval_loss = evaluate(dataloader=data_loader['eval'], model=model, loss_fc=loss_fc)
75
76         print(epoch, eval_acc, eval_loss)
77
78         is_best = (eval_acc > best_acc) or \
79                     (eval_loss < best_loss and eval_acc >=
best_acc)
80
81         if is_best:
82             early_stop_count = 0
83         else:
84             early_stop_count += 1
85         if early_stop_count > early_stop:
86             break
87         if is_best:
88             best_acc = eval_acc
89             best_loss = eval_loss
90             early_stop_count = 0
91             model.save(model_file)
92
93             model.load(model_file)
94
95             model.eval()
96             acc_test, acc_loss = evaluate(data_loader['test'], model, loss_fc)
97
98             print(acc_test)

```

```

47 optimizer = torch.optim.Adam(model.parameters(), lr=5
e-3)
48 loss_fc = nn.CrossEntropyLoss()
49
50 model_file = './src/' + dataset_name + '.pt'
51
52 model.train()
53 early_stop_count = 0
54 best_acc = 0
55 best_loss = 100
56 for epoch in range(epochs):
57     acc = []
58     loss_list = []
59     model.train()
60     for i, data in enumerate(data_loader):
61         print(data)
62         logit = model(data)
63         loss = loss_fc(logit, data.y)
64         optimizer.zero_grad()
65         loss.backward()
66         nn.utils.clip_grad_norm_(model.parameters(),
max_norm=2.0)
67         optimizer.step()
68
69         prediction = torch.argmax(logit, -1)
70         loss_list.append(loss.item())
71         acc.append((prediction == data.y).cpu().numpy
72 ())
73         eval_acc, eval_loss = evaluate(dataloader=data_loader, model=model, loss_fc=loss_fc)
74
75         print(epoch, eval_acc, eval_loss)
76
77         is_best = (eval_acc > best_acc) or \
78                     (eval_loss < best_loss and eval_acc >=
best_acc)
79
80         if is_best:
81             early_stop_count = 0
82         else:
83             early_stop_count += 1
84         if early_stop_count > early_stop:
85             break
86         if is_best:
87             best_acc = eval_acc
88             best_loss = eval_loss
89             early_stop_count = 0
90             torch.save(model.state_dict(), model_file)
91
92             model.load_state_dict(torch.load(model_file))
93
94             model.eval()
95             acc_test, acc_loss = evaluate(dataloader=data_loader, model=model, loss_fc=loss_fc)
96
97             print(acc_test)

```