AI VIET NAM – 2024

Graph Neural Network - Exercise

Minh-Duc Bui và Vinh Dinh Nguyen

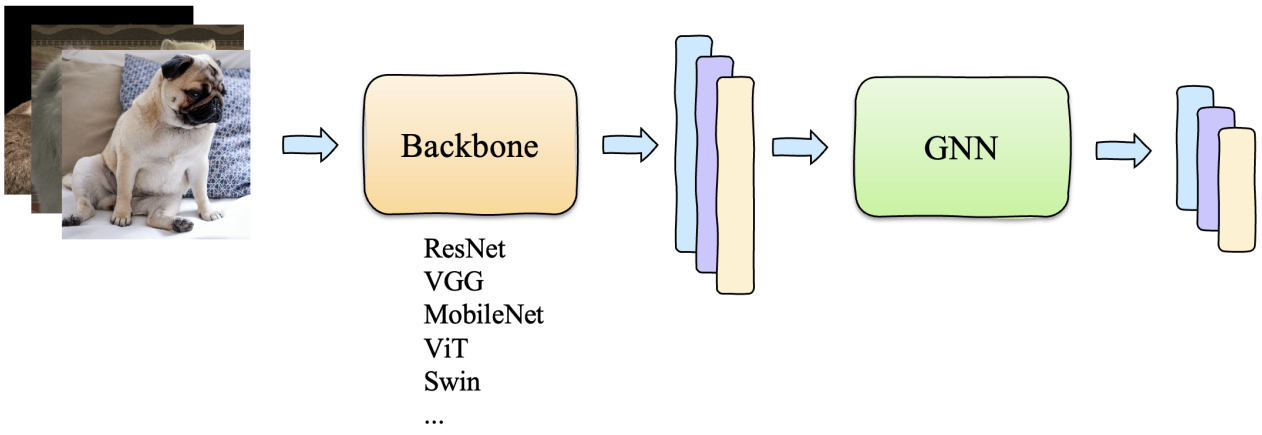
Ngày 28 tháng 4 năm 2024

Phần I: Giới thiệu

Graph Neural Network - GNN là một nerual network sử dụng để xử lý dữ liệu được biểu diễn dưới dạng graph. Trong graph, các đỉnh (node) biểu diễn các thực thể, và các cạnh (edge) thể hiện các mối quan hệ giữa các node. GNN được thiết kế để trích xuất thông tin từ cấu trúc graph để thực hiện các bài toán như classification, prediction, clustering, và labeling graph data.

Đặc điểm chính của GNN là khả năng tổng hợp thông tin từ các node lân cận trong graph. Mỗi node trong graph có thể được xem như một feature vector, và GNN sử dụng các cơ chế lan truyền thông tin qua các edge để cập nhật các vector này. Quá trình này thường được thực hiện qua nhiều layer của network, mỗi layer thực hiện một bước lan truyền và cập nhật thông tin.

GNN có hiệu suất mạnh mẽ trong nhiều ứng dụng thực tế, bao gồm social network analysis, predicting molecular interactions, graph image classification, và nhiều tác vụ khác.



Trong buổi hôm nay, ta sẽ sử dụng GNN vào bài toán classification nói chung và image classification nói riêng.

Phần II: Nội dung

1. **Ý tưởng chung**

   GNN hoạt động bằng cách tổng hợp thông tin từ các node có liên quan với nhau (có edge), trong các dataset dành riêng cho GNN (ví dụ Cora, Citeseer, Pubmed,...), các edge được đã được định nghĩa sẵn, điều này gây ra khó khăn khi sử dụng GNN vào các bài toán khác (ví dụ image classification, object detection,...).

   > Để có thể sử dụng GNN vào các bài toán bất kì, ta cần trả lời tuần tự 3 câu hỏi sau:
   >
   > - Node là gì?
   > - Edge là gì?
   > - Ta có thể thêm hàm loss phụ trợ không?

   Ba câu hỏi trên sẽ quyết định việc ta có thể sử dụng GNN vào một bài toán bất kì được hay không.

   > Tuy nhiên, để GNN có thể cải thiện performance của model hiện tại ta cần phải trả lời câu hỏi quan trọng sau:
   >
   > - GNN sẽ hoạt động như thế nào?

   Ta cần phải giải thích rõ cách GNN sẽ hoạt động trước khi tiến hành code thay vì hoàn thành 3 câu hỏi đầu tiên và cầu nguyện model sẽ chạy tốt.

2. **Ý tưởng GNN cho bài toán image classification**

   Để có thể sử dụng GNN cho bài toán image classification, ta cần trả phải trả lời 3 câu hỏi ở phần trên.

   - **Node**: Ta có thể định nghĩa mỗi sample (image) là một node.
   - **Edge**: Khi mỗi sample là một node thì edge chính là liên kết giữa 2 sample khi 2 sample đó có cùng class. Nếu 2 sample không cùng class thì sẽ không có edge.
   - **Hàm loss phụ trợ**: Trong các bài toán mà các edge được định nghĩa rõ ràng (ví dụ trong bài này là cùng class với nhau) thì thêm hàm loss phụ trợ chắc chắn sẽ tăng performance so với không thêm.

   Sau khi đã trả lời được 3 câu hỏi trên thì ta đã chắc chắn rằng đã có thể triển khai code. Tuy nhiên, ta cần phải giải thích vai trò của GNN là gì trong model mới này.

   Ta thấy, trong 1 batch data sẽ có nhiều sample có cùng class, và trong các sample cùng class sẽ có các sample khó học và các sample dễ học. Ví dụ class về ghế sofa, các sample dễ học là các sample mà trong đó chỉ có mỗi ghế sofa, các sample khó học sẽ bao gồm ghế sofa và bàn gần vị trí đó. Việc này là hoàn toàn bình thường vì ghế sofa và bàn thường nằm cùng nhau trong nhà, nên khi chụp ảnh ghế thì sẽ dính phải bàn. Khi sử dụng GNN để tổng hợp thông tin từ các sample cùng class trong 1 batch thì ta sẽ dùng các sample dễ để "cung cấp" thông tin cho các sample khó, từ

đó giúp model học tốt các sample khó và performance tổng quát của model sẽ được cải thiện.

Ta sẽ tăng độ khó của dataset trong bài toán này để thấy sự hiệu quả của GNN.

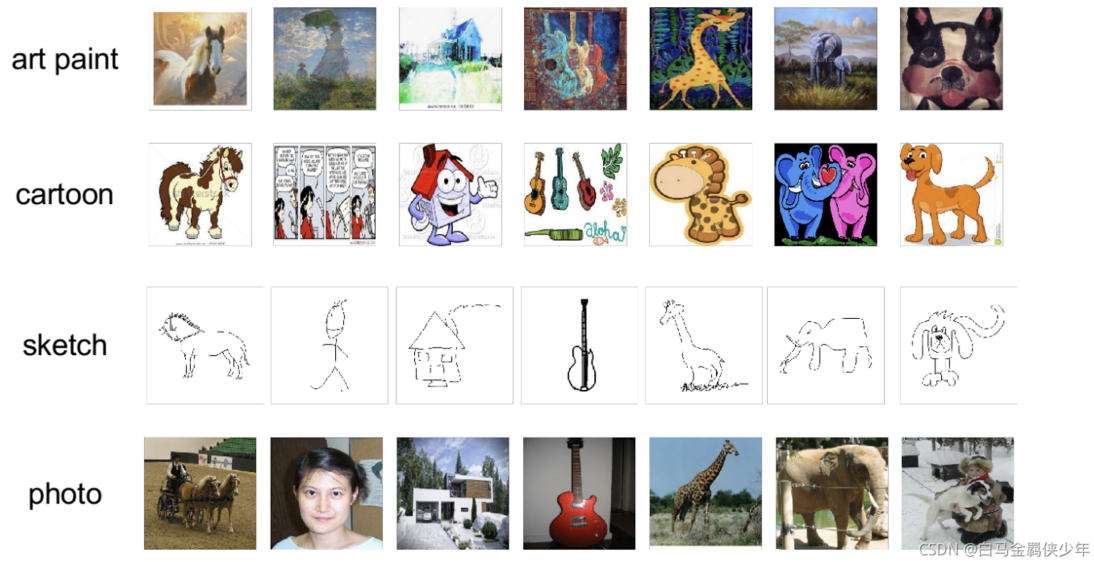
3. **Dataset PACS**

PACS là một tập dữ liệu hình ảnh về domain generalization. PACS bao gồm 4 domain: photo (1.670 ảnh), art painting (2.048 ảnh), cartoon (2.344 ảnh) và sketch (3.929 ảnh). Mỗi domain bao gồm 7 class.

Điểm khó của PACS là việc sẽ có nhiều domain cho cùng 1 class, ví dụ trong class person sẽ có 4 domain (cột số 2 trong hình 1):

- Ảnh person được chụp từ thực tế (photo)
- Ảnh person được vẽ phác thảo (sketch)
- Ảnh person từ phim hoạt hình (cartoon)
- Ảnh vẽ person (art painting)

Việc có nhiều domain cho cùng một class như vậy sẽ tăng độ khó của dataset, đòi hỏi model phải có khả năng generalize tốt hơn.



Hình 1: Dataset PACS.

Sau đây là đoạn code download dataset PACS và tạo dataloader. Vì có 4 domain khác nhau nên ta sẽ có 4 dataset khác nhau.

```
# means and standard deviations ImageNet because the network is pretrained
means, stds = (0.485, 0.456, 0.406), (0.229, 0.224, 0.225)

# Define transforms to apply to each image
transf = transforms.Compose(
    [
```

```
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(means, stds),
    ]
)

# Clone github repository with data
if not os.path.isdir("./Homework3-PACS"):
    !git clone https://github.com/MachineLearning2020/Homework3-PACS

# Define datasets root
DIR_PHOTO = "Homework3-PACS/PACS/photo"
DIR_ART = "Homework3-PACS/PACS/art_painting"
DIR_CARTOON = "Homework3-PACS/PACS/cartoon"
DIR_SKETCH = "Homework3-PACS/PACS/sketch"

# Prepare Pytorch train/test Datasets
photo_dataset = torchvision.datasets.ImageFolder(DIR_PHOTO, transform=transf
    )
art_dataset = torchvision.datasets.ImageFolder(DIR_ART, transform=transf)
cartoon_dataset = torchvision.datasets.ImageFolder(DIR_CARTOON, transform=
    transf)
sketch_dataset = torchvision.datasets.ImageFolder(DIR_SKETCH, transform=
    transf)
```

Trong bài toán này, ta sẽ sử dụng domain photo làm tập train, và sử dụng tập val của photo và toàn bộ các domain còn lại để test. Tiếp theo ta sẽ tạo train và test data loader. Ta sẽ dùng ConcatDataset để concat các dataset với nhau.

```
# Split dataset into train and test
train_size = int(0.8 * len(photo_dataset))
test_size = len(photo_dataset) - train_size

train_dataset, test_dataset = torch.utils.data.random_split(
    photo_dataset, [train_size, test_size]
)

# Concatenate all datasets
test_datasets = torch.utils.data.ConcatDataset([test_dataset, art_dataset])

# Create Dataloaders
trainloader = DataLoader(
    train_dataset, batch_size=128, shuffle=True, num_workers=4, drop_last=
    True
)
testloader = DataLoader(test_datasets, batch_size=128, shuffle=False,
    num_workers=4)
```

4. Áp dụng GNN cho bài toán image classification

Ta sẽ định nghĩa model GNN trước khi tích hợp GNN vào model chính. GNN sẽ bao gồm 2 phần chính là Edge và Node network. Edge network sẽ làm nhiệm vụ edge prediction, cụ thể, ta sẽ dùng edge network để predict giữa 2 sample bất kì trong 1 batch có liên kết với nhau không. Ta cần dùng 1 model để predict edge giữa 2 sample bất kì và dùng hàm loss để giúp edge network predict tốt hơn. Nói cách khác, vì khi inference ta không có label nên không thể biết 2 sample bất kì có

edge với nhau hay không, nên ta mới cần edge network làm nhiệm vụ edge prediction.

```
class GCN(nn.Module):
    def __init__(self, in_features, edge_features, out_feature, device, ratio=(1,)):
        super(GCN, self).__init__()

        self.edge_net = EdgeNet(
            in_features=in_features,
            num_features=edge_features,
            device=device,
            ratio=ratio,
        )
        # set edge to node
        self.node_net = NodeNet(
            in_features=in_features,
            num_features=out_feature,
            device=device,
            ratio=ratio,
        )
        # mask value for no-gradient edges
        self.mask_val = -1

    def label2edge(self, targets):
        """convert node labels to affinity mask for backprop"""
        num_sample = targets.size()[1]
        label_i = targets.unsqueeze(-1).repeat(1, 1, num_sample)
        label_j = label_i.transpose(1, 2)
        edge = torch.eq(label_i, label_j).float()
        target_edge_mask = (
            torch.eq(label_i, self.mask_val) + torch.eq(label_j, self.mask_val)
        ).type(torch.bool)
        source_edge_mask = ~target_edge_mask
        init_edge = edge * source_edge_mask.float()
        return init_edge[0], source_edge_mask

    def forward(self, init_node_feat):
        #  compute normalized and not normalized affinity matrix
        edge_feat, edge_sim = self.edge_net(init_node_feat)
        # compute node features and class logits
        logits_gnn = self.node_net(init_node_feat, edge_feat)
        return logits_gnn, edge_sim
```

Sau đây là đoạn code về edge network. Ta thấy output của edge network là edge_feat có shape là (batch_size, batch_size), đây chính là correlation matrix. Mỗi vị trí ij trong matrix này là mối liên hệ giữa sample i và sample j, giá trị càng gần 1 thì 2 sample càng có mối liên hệ cao, và ngược lại, giá trị càng thấp thì mối liên hệ càng thấp. Nói cách khác, mối liên hệ cao là các sample có cùng class với nhau, mối quan hệ thấp là các sample khác class với nhau.

```
class EdgeNet(nn.Module):
    def __init__(self, in_features, num_features, device, ratio=(1,)):
        super(EdgeNet, self).__init__()
        num_features_list = [num_features * r for r in ratio]
```

```
        self.device = device
        # define layers
        layer_list = OrderedDict()
        for l in range(len(num_features_list)):
            layer_list["conv{}".format(l)] = nn.Conv2d(
                in_channels=num_features_list[l - 1] if l > 0 else in_features,
                out_channels=num_features_list[l],
                kernel_size=1,
                bias=False,
            )
            layer_list["norm{}".format(l)] = nn.BatchNorm2d(
                num_features=num_features_list[l]
            )
            layer_list["relu{}".format(l)] = nn.LeakyReLU()
        # add final similarity kernel
        layer_list["conv_out"] = nn.Conv2d(
            in_channels=num_features_list[-1], out_channels=1, kernel_size=1
        )
        self.sim_network = nn.Sequential(layer_list).to(device)

    def forward(self, node_feat):
        node_feat = node_feat.unsqueeze(dim=0)  # (1, bs, dim)
        num_tasks = node_feat.size(0)  # 1
        num_data = node_feat.size(1)  # bs
        x_i = node_feat.unsqueeze(2)  # (1, bs, 1, dim)
        x_j = torch.transpose(x_i, 1, 2)  # (1, 1, bs, dim)
        x_ij = torch.abs(x_i - x_j)  # (1, bs, bs, dim)
        x_ij = torch.transpose(x_ij, 1, 3)  # (1, dim, bs, bs)
        # compute similarity/dissimilarity (batch_size x feat_size x num_samples x num_samples)
        sim_val = (
            torch.sigmoid(self.sim_network(x_ij)).squeeze(1).squeeze(0).to(self.device)
        )  # (bs, bs)
        # normalize affinity matrix
        force_edge_feat = (
            torch.eye(num_data).unsqueeze(0).repeat(num_tasks, 1, 1).to(self.device)
        )  # (1, bs, bs)
        edge_feat = sim_val + force_edge_feat  # (bs, bs)
        edge_feat = edge_feat + 1e-6  # add small value to avoid nan
        edge_feat = edge_feat / torch.sum(edge_feat, dim=1).unsqueeze(1)  # normalize
        return edge_feat, sim_val  # (bs, bs), (bs, bs)
```

Lưu ý: edge_feat là tensor sau khi đã normalize sim_val, ta sẽ dùng sim_val để cập nhật hàm loss cho model nên cần return về.

Tiếp theo, node network sẽ dùng edge_feat để tổng hợp thông tin. Node network sẽ dùng các giá trị trong edge_feat để tổng hợp thông tin cho các sample, giá trị càng cao sẽ được tổng hợp càng nhiều, và ngược lại.

```
class NodeNet(nn.Module):
    def __init__(self, in_features, num_features, device, ratio=(1,)):
```

```
        super(NodeNet, self).__init__()
        num_features_list = [num_features * r for r in ratio]
        self.device = device
        # define layers
        layer_list = OrderedDict()
        for l in range(len(num_features_list)):
            layer_list["conv{}".format(l)] = nn.Conv2d(
                in_channels=num_features_list[l - 1] if l > 0 else
    in_features * 2,
                out_channels=num_features_list[l],
                kernel_size=1,
                bias=False,
            )
            layer_list["norm{}".format(l)] = nn.BatchNorm2d(
                num_features=num_features_list[l]
            )
            if l < (len(num_features_list) - 1):
                layer_list["relu{}".format(l)] = nn.LeakyReLU()
        self.network = nn.Sequential(layer_list).to(device)

    def forward(self, node_feat, edge_feat):
        """node_feat: (bs, dim), edge_feat: (bs, bs)"""
        node_feat = node_feat.unsqueeze(dim=0)  # (1, bs, dim)
        num_tasks = node_feat.size(0)  # 1
        num_data = node_feat.size(1)  # bs
        # get eye matrix (batch_size x node_size x node_size) only use inter
     dist.
        diag_mask = 1.0 - torch.eye(num_data).unsqueeze(0).repeat(num_tasks,
     1, 1).to(
            self.device
        )  # (1, bs, bs)
        # set diagonal as zero and normalize
        edge_feat = F.normalize(edge_feat * diag_mask, p=1, dim=-1)  # (bs,
    bs)
        # compute attention and aggregate
        aggr_feat = torch.bmm(edge_feat.squeeze(1), node_feat)  # (bs, dim)
        node_feat = torch.cat([node_feat, aggr_feat], -1).transpose(
            1, 2
        )  # (1, 2*dim, bs)
        # non-linear transform
        node_feat = self.network(node_feat.unsqueeze(-1)).transpose(
            1, 2
        )  # (1, bs, dim)
        node_feat = node_feat.squeeze(-1).squeeze(0)  # (bs, dim)
        return node_feat
```

Tóm lại, input của GNN sẽ có shape [batch_size, dim_1] và output có shape [batch_size, dim_2], ta hoàn toàn có thể đặt dim_1 = dim_2 hoặc dim_2 = num_classes.

Bước cuối cùng là tích hợp GNN vào một backbone bất kì:

```
from torchvision.models import mobilenet_v3_small

class Model(nn.Module):
    def __init__(self, num_classes=7):
        super(Model, self).__init__()
```

```
        self.backbone = mobilenet_v3_small(pretrained=True)
        self.backbone.classifier = nn.Sequential()

        self.gcn = GCN(
            in_features=576,
            edge_features=576,
            out_feature=num_classes,
            device="cuda",
            ratio=(1,),
        )

    def forward(self, x):
        x = self.backbone(x)
        x, edge_sim = self.gcn(x)
        return x, edge_sim
```

Để dễ dàng hình dung hơn nữa thì ta hãy nhìn vào model bình thường không có GNN:

```
from torchvision.models import mobilenet_v3_small

class Model(nn.Module):
    def __init__(self, num_classes=2):
        super(Model, self).__init__()
        self.backbone = mobilenet_v3_small(pretrained=True)
        self.backbone.classifier = nn.Sequential()

        self.head = nn.Sequential(OrderedDict([
            ('fc1', nn.Linear(576, 256)),
            ('relu', nn.ReLU()),
            ('fc2', nn.Linear(256, num_classes))
        ]))

    def forward(self, x):
        x = self.backbone(x)
        x = self.head(x)
        return x
```

Qua 2 đoạn code trên ta thấy chỉ cần một thay đổi đơn giản là ta đã có thể tích hợp GNN vào backbone bất kì cho bài toán image classification.

Tiếp theo ta cần định nghĩa thêm hàm loss để giúp model predict edge giữa các sample chính xác hơn. Hàm edge loss này sẽ giúp model predict 1 cho 2 sample cùng class và 0 khi 2 sample khác class, do đó ta cần dùng Binary Cross Entroy (dòng 2). Hàm edege loss sẽ được tính toán ở dòng 18-21. Biến edge_sim mà model return về chính là dạng chưa normalize của correlation matrix có shape [bs, bs]. Các label ta dùng để train model không thể trực tiếp sử dụng cho hàm edge loss, các label này cần phải chuyển sang dạng mới có shape [bs, bs] và mang các giá trị 0 hoặc 1 (được tính toán thông qua hàm label2edge).

```
criterion = nn.CrossEntropyLoss()
criterion_edge = nn.BCELoss()
...
for i, (inputs, labels) in enumerate(trainloader, 0):
```

```
    # Move inputs and labels to the device
    inputs, labels = inputs.to(device), labels.to(device)

    # Zero the parameter gradients
    optimizer.zero_grad()

    # Forward pass
    outputs, edge_sim = model(inputs)

    # Cls loss
    loss_cls = criterion(outputs, labels)

    # Edge loss
    edge_gt, edge_mask = model.gcn.label2edge(labels.unsqueeze(dim=0))
    loss_edge = criterion_edge(
        edge_sim.masked_select(edge_mask), edge_gt.masked_select(edge_mask)
    )

    # Total loss
    loss = 0.3 * loss_cls + loss_edge

    running_loss += loss.item()

    # Backward pass and optimization
    loss.backward()
    optimizer.step()
...
```

Lưu ý: file pdf này chỉ cung cấp một số đoạn code chính để tránh người đọc phân tâm. Tất cả các phần code còn lại như một bài toán image classification bình thường, full code sẽ nằm ở file notebook.

Phần III: Câu hỏi trắc nghiệm

1. Đâu là lưu ý khi áp dụng GNN vào các bài toán bất kì?
   (*a*) Label là gì?
   (*b*) Độ lớn dataset
   (*c*) Độ phức tạp của model
   (*d*) Edge là gì?

2. Đâu là lưu ý khi áp dụng GNN vào các bài toán bất kì?
   (*a*) Label là gì?
   (*b*) Node là gì?
   (*c*) Số lượng param
   (*d*) Loss là gì?

3. Đâu là lưu ý khi áp dụng GNN vào các bài toán bất kì?
   (*a*) Có cần hàm loss phụ trợ hay không?
   (*b*) Loại hàm loss của model hiện tại.
   (*c*) Model có tính toán song song hay không?
   (*d*) Loss là gì?

4. Sau khi chắc chắn GNN có thể sử dụng, điều gì ta cần cân nhắc tiếp theo là phù hợp nhất?
   (*a*) Không cần gì
   (*b*) Độ lớn của model sau khi thêm GNN
   (*c*) Hiểu rõ cách GNN tổng hợp thông tin
   (*d*) Tốc độ của model sau khi thêm GNN

5. Nếu input của GNN có shape [batch_size, dim_1] và output có shape [batch_size, dim_2] thì nhận định nào sau đây là **SAI**:
   (*a*) dim_2 có thể bằng dim_1 * 2
   (*b*) dim_2 có thể bằng số lượng class
   (*c*) dim_1 và dim_2 không bắt buộc bằng nhau
   (*d*) dim_1 và dim_2 phải bằng nhau

6. Khi muốn kết hợp nhiều dataset với nhau ta có thể dùng:
   (*a*) torch.utils.data.MergeDataset
   (*b*) torch.utils.data.ConcatDataset
   (*c*) torch.utils.data.MergeData
   (*d*) torch.utils.data.ConcatData

7. Trong GNN, cơ chế nào được sử dụng để lan truyền thông tin qua các node và edge?
   (*a*) Feedforward
   (*b*) Backpropagation
   (*c*) Message passing

(*d*) Gradient descent

8. Đâu là điểm yếu của GNN?
   (*a*) Tốc độ train model chậm
   (*b*) Tốc độ inference nhanh
   (*c*) Số lượng param cực lớn
   (*d*) Không có điểm yếu nào
9. Đâu là điểm yếu của GNN?
   (*a*) Cần train model theo 2 stage
   (*b*) Cần inference model theo 2 stage
   (*c*) Tốc độ inference chậm
   (*d*) Tốc độ train model nhanh
10. Đâu là những yếu cầu của GNN?
    (*a*) Batch size bất kì, data bất kì
    (*b*) Batch size = 1, data clean
    (*c*) Batch size > 1, data bất kì
    (*d*) Batch size > 1, data clean

- Hết -