Report on Causal Attention Learning for Graph Classification

May 22, 2024

1 Introduction

In the field of graph classification, traditional Graph Neural Networks (GNNs) often rely on statistical correlations between graph features and labels. This can lead to poor generalization performance, especially when the data distribution shifts. The paper "Causal Attention for Interpretable and Generalizable Graph Classification" proposes a novel method called Causal Attention Learning (CAL) to address these issues by focusing on causal relationships rather than mere correlations.

2 Methodology

2.1 Causal Attention Learning (CAL)

The CAL framework aims to enhance the interpretability and generalizability of GNNs by incorporating causal inference principles. The core idea is to separate causal features from shortcut (non-causal) features and ensure that the model predictions are based on the causal features.

3 Workflow and Key Components

3.1 1. Attention Mechanism

The model uses attention modules to estimate the causal and shortcut features from the input graph. Two Multi-Layer Perceptrons (MLPs) are employed: one for node-level attention and another for edge-level attention.

- **Node-level Attention**:

  $$\alpha_i = \text{softmax}(W_n \cdot h_i)$$

  where α_i is the attention score for node i, W_n is the weight matrix for node-level attention, and h_i is the feature representation of node i.

- **Edge-level Attention**:

  $$\beta_{ij} = \text{softmax}(W_e \cdot [h_i \| h_j])$$

  where β_{ij} is the attention score for edge (i, j), W_e is the weight matrix for edge-level attention, and $[h_i \| h_j]$ is the concatenated feature representation of nodes i and j.

3.2 2. Backdoor Adjustment

The model applies backdoor adjustment from causal theory to mitigate the confounding effect of shortcut features. By combining each causal feature with various shortcut features, the model ensures stable predictions regardless of changes in shortcut parts.

$$P(Y \mid do(X)) = \sum_z P(Y \mid X, Z = z) P(Z = z)$$

Here, Y is the target variable, X is the causal feature, and Z represents the shortcut features.

3.3 3. Disentanglement

The model disentangles the causal and trivial features using separate GNN layers for each type of feature. Supervised loss (L_{sup}) and uniform classification loss (L_{unif}) are used to guide the disentanglement process.

- **Supervised Loss**:

  $$L_{sup} = E_{(x,y)\sim D}[\text{CE}(f(x), y)]$$

  where CE denotes cross-entropy loss, $f(x)$ is the model prediction, and y is the ground truth label.

- **Uniform Classification Loss**:

  $$L_{unif} = \text{KL}(\hat{y} \parallel \text{Uniform})$$

  where KL is the Kullback-Leibler divergence between the predicted distribution $\hat{y}$ and a uniform distribution.

3.4 4. Causal Intervention

The causal intervention loss (L_{caus}) is introduced to enforce the model to make predictions invariant to the changes in shortcut features.

$$L_{caus} = \sum_i \text{Var}_Z[P(Y \mid X_i, Z)]$$

This loss ensures that the variance of the prediction given the causal feature X_i across different values of shortcut features Z is minimized.

3.5 Overall Objective Function

The overall objective function is a weighted sum of these losses:

$$L = L_{sup} + \lambda_1 L_{unif} + \lambda_2 L_{caus}$$

where λ_1 and λ_2 are hyperparameters that balance the contribution of each loss component.

4 Important Code Implementations

4.1 CausalGCN Class

The `CausalGCN` class is a key component that implements the CAL strategy using GCN layers. Below is a simplified version of the class highlighting its main features:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import global_add_pool, GCNConv

class CausalGCN(torch.nn.Module):
    def __init__(self, num_features, num_classes, args):
        super(CausalGCN, self).__init__()
        hidden = args.hidden
        self.global_pool = global_add_pool
        self.dropout = args.dropout
        self.conv_feat = GCNConv(num_features, hidden)
        self.bns_conv = torch.nn.ModuleList([nn.BatchNorm1d(hidden) for _ in range(args.laye
        self.convs = torch.nn.ModuleList([GCNConv(hidden, hidden) for _ in range(args.layers
        self.edge_att_mlp = nn.Linear(hidden * 2, 2)
        self.node_att_mlp = nn.Linear(hidden, 2)
        self.context_convs = GCNConv(hidden, hidden)
        self.objects_convs = GCNConv(hidden, hidden)
        self.fc1_c = nn.Linear(hidden, hidden)
        self.fc2_c = nn.Linear(hidden, num_classes)
        self.fc1_o = nn.Linear(hidden, hidden)
        self.fc2_o = nn.Linear(hidden, num_classes)

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.batch
        x = F.relu(self.conv_feat(x, edge_index))
        for i, conv in enumerate(self.convs):
            x = self.bns_conv[i](x)
            x = F.relu(conv(x, edge_index))
        edge_rep = torch.cat([x[edge_index[0]], x[edge_index[1]]], dim=-1)
        edge_att = F.softmax(self.edge_att_mlp(edge_rep), dim=-1)
        edge_weight_c, edge_weight_o = edge_att[:, 0], edge_att[:, 1]
        node_att = F.softmax(self.node_att_mlp(x), dim=-1)
        xc, xo = node_att[:, 0].view(-1, 1) * x, node_att[:, 1].view(-1, 1) * x
        xc = F.relu(self.context_convs(xc, edge_index, edge_weight_c))
        xo = F.relu(self.objects_convs(xo, edge_index, edge_weight_o))
        xc = self.global_pool(xc, batch)
        xo = self.global_pool(xo, batch)
        xc_logis = F.log_softmax(self.fc2_c(F.relu(self.fc1_c(xc))), dim=-1)
        xo_logis = F.log_softmax(self.fc2_o(F.relu(self.fc1_o(xo))), dim=-1)
        return xc_logis, xo_logis
```

5 Training and Evaluation

The training process involves iterating over epochs, computing losses, and updating model parameters. The evaluation is conducted on validation and test sets to monitor model performance.

5.1 Training Function

```
def train_causal_syn(train_set, val_set, test_set, model_func, args):
    train_loader = DataLoader(train_set, args.batch_size, shuffle=True)
    val_loader = DataLoader(val_set, args.batch_size, shuffle=False)
    test_loader = DataLoader(test_set, args.batch_size, shuffle=False)
    model = model_func(args.feature_dim, args.num_classes).to(device)
    optimizer = Adam(model.parameters(), lr=args.lr)
    lr_scheduler = CosineAnnealingLR(optimizer, T_max=args.epochs, eta_min=args.min_lr)
    best_val_acc = 0

    for epoch in range(1, args.epochs + 1):
        model.train()
        for data in train_loader:
            optimizer.zero_grad()
            data = data.to(device)
            xc_logis, xo_logis = model(data)
            loss = F.nll_loss(xc_logis, data.y) + F.nll_loss(xo_logis, data.y)
            loss.backward()
            optimizer.step()

        model.eval()
        val_acc = evaluate(model, val_loader)
        if val_acc > best_val_acc:
            best_val_acc = val_acc
            test_acc = evaluate(model, test_loader)
        lr_scheduler.step()
        print(f"Epoch: {epoch}, Loss: {loss.item()}, Val Acc: {val_acc}, Test Acc: {test_acc

def evaluate(model, loader):
    correct = 0
    model.eval()
    with torch.no_grad():
        for data in loader:
            data = data.to(device)
            xc_logis, xo_logis = model(data)
            pred = xc_logis.max(1)[1]
            correct += pred.eq(data.y).sum().item()
    return correct / len(loader.dataset)
```

6 Experimental Setup and Results

6.1 Experimental Setup

To verify the effectiveness of the proposed Causal Attention Learning (CAL) strategy, the authors conducted extensive experiments on both synthetic and real-world datasets. The experimental setup includes the following components:

- **Datasets:** The experiments were conducted on synthetic datasets and real-world datasets. The synthetic datasets consist of graphs with controlled biases to simulate various levels of out-of-distribution (OOD) scenarios. Real-world datasets include biological datasets (MUTAG, NCI1, PROTEINS), social datasets (COLLAB, IMDB-B, IMDB-M), and superpixel datasets (MNIST, CIFAR-10).

- **Baselines:** Various graph classification models were used as baselines, including attention-based methods (GAT, GATv2, SuperGAT), pooling-based methods (SortPool, DiffPool, Top-k Pool, SAGPool), kernel-based methods (GK, WL, DGK), and GNN-based methods (GCN, GIN). Additionally, state-of-the-art algorithms designed for OOD issues (IRM, DRO) were also included for comparison.

- **Metrics:** The primary evaluation metric was classification accuracy. The performance on OOD scenarios was particularly emphasized by comparing results across different bias levels in synthetic datasets.

- **Implementation Details:** Hyperparameters such as learning rate, number of epochs, batch size, and the coefficients for loss components (λ_1 and λ_2) were tuned for optimal performance. The authors also provided the code repository for reproducibility.

6.2 Results

6.2.1 Synthetic Datasets

The experiments on synthetic datasets aimed to evaluate the robustness of CAL under various bias levels. The results are summarized in Table 1 and Figure 4 of the paper.

Method	SYN-0.1	SYN-0.3	Unbiased	SYN-0.7	SYN-0.9
GATv2	87.25 (↓7.37%)	92.19 (↓2.12%)	94.19	93.31 (↓0.93%)	90.62 (↓3.79%)
SuperGAT	83.81 (↓12.75%)	91.94 (↓4.29%)	96.06	88.50 (↓7.89%)	82.81 (↓13.79%)
GCN + CAL	89.38 (↓6.03%)	93.50 (↓1.70%)	95.12	95.06 (↓0.06%)	93.31 (↓1.90%)
GIN + CAL	93.19 (↓3.87%)	96.31 (↓0.65%)	96.94	96.56 (↓0.39%)	95.25 (↓1.74%)

Table 1: Test Accuracy (%) on synthetic datasets with different biases.

The key observations from the synthetic dataset experiments are:

- **Effectiveness of CAL:** The proposed CAL significantly improves the OOD performance across different bias levels. Models with CAL consistently outperform their vanilla counterparts, demonstrating the effectiveness of mitigating the confounding effect of shortcut features.

- **Stability:** CAL-equipped models show more stable performance under extreme biases compared to baseline methods, indicating better generalization capabilities.

6.2.2 Real-World Datasets

The results on real-world datasets further validate the practicality of CAL in diverse graph classification tasks. The results are summarized in Table 2 of the paper.

Dataset	MUTAG	NCI1	PROTEINS	COLLAB	IMDB-B	MNIST
GCN	88.20	82.97	75.65	81.72	73.89	90.49
GCN + CAL	**89.24**	**83.48**	**76.28**	**82.08**	**74.40**	**94.58**
GIN	89.42	82.71	76.21	82.08	73.40	96.51
GIN + CAL	**89.91**	**83.89**	**76.92**	**82.68**	**74.13**	**96.93**

Table 2: Test Accuracy (%) on real-world datasets.

The key observations from the real-world dataset experiments are:

- **Generalization:** CAL provides consistent performance improvements on real-world datasets, indicating its robustness in practical applications.

- **Interpretability:** The causal attended-graphs captured by CAL provide insightful interpretations, helping to understand what knowledge the GNN uses for predictions.

7 Conclusion

The experiments demonstrate that the proposed Causal Attention Learning (CAL) strategy effectively enhances the interpretability and generalizability of graph classification models. By focusing on causal relationships and mitigating the confounding effect of shortcut features, CAL achieves superior performance in both synthetic and real-world scenarios.

8 Conclusion

The CAL framework enhances GNNs by addressing the limitations of traditional attention mechanisms. By focusing on causal features and mitigating the influence of shortcut features, CAL improves both interpretability and generalization of graph classification models. The provided code snippets demonstrate the implementation of CAL in a GCN-based model and the associated training and evaluation procedures.

9 References

- Yongduo Sui, Xiang Wang, Jiancan Wu, Min Lin, Xiangnan He, and Tat-Seng Chua. "Causal Attention for Interpretable and Generalizable Graph Classification." In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), 2022. Link to the paper