GNN-PRP

Haoyang Ling

March 15, 2022

1 Momentum Contrast for Unsupervised Visual Representation Learning: MoCo

1.1 Notes

1. Contrastive learning has won over many supervised learning in ImageNet.
2. A main goal for the self-supervised learning is to achieve transfer learning.
3. Pretext/proxy task: find the similarity of the figures.
4. Instance discrimination: after transformation, the semantic meaning should reserve through positive pairs and negative pairs. Example: SimClr, CLIP.
5. Main contribution:
   Momentum(similar to Adam optimizer): $y_t = my_{t-1} + (1-m)x_t$
   Perspective: query & key problems (similar to attention mechanism)
6. use contrastive loss to calculate the similarity, and use the momentum to smoothly update the encoder.
7. popular self-supervised structure: contrastive learning and masked auto-encoding (MAE). **What is the difference between MAE and masked attribute in contrastive learning?**
8. The temperature in the InfoNCE: if large, too emphasis on the negative pairs.

1.2 Useful

1. momentum to smooth the update in the mini-batch
2. query & key problem (attention)
3. cross-channel, **pseudo labeling** (cluster assumption), patch ordering, and **clustering**.
4. Ablation experiment
5. Strange learning rate lr = 30?
6. shuffle-BN, group normalization, and weight standardization.
7. BYOL (no negative pair?)

2 Graph Self-Supervised Learning: A Survey

2.1 Notes

1. Four approaches: generation-based, auxiliary-property-based, contrast-based, and hybrid.
2. **Existing Problems:** over-fitting (not transferable), and proper pretext task (non-Euclidean data space). Hyperbolic space embedding/manifold learning (to learn the curvature)?
3. Pretext task in CV and NLP
   (a) image colorization: delete some attributes, use clustering to smooth the graph (or Personalized Page Rank). Use the embedded node attributes to predict edge attribute, and edge connection?

(b) image contrastive learning: rotation (task-dependent), color jittering, crop(ring to embed the context).
Think: different nodes in different positions contribute differently. Is proper to use gloabl_add_pool? (Different levels of attention Mechanism by query and key: graph attributes (with super-nodes/motifs?) $\longrightarrow$ sub-graph with neighborhoods (by clustering?) $\longrightarrow$ nodes attributes combined with neighbors and edge attributes by MessagePassing?)

(c) masked language modeling: masked adj_t/edge_index

(d) next sentence prediction: temporal/evolution (graph evolution)

4. GNN property: permutation in node index affects edge_index, node attributes matrix, edge attribute matrix, and further nodes embedding, but not graph embedding. Truncated random walk (?) & contextual nodes.

Granularity	**Data**
node	features, *positions (or generated embedding)
edge	connection, attributes, *direction, *type
motif	quantity, relative position, *(learned) features

Table 1: Different Levels of Granularity in Graph

Principals	**GNN**	**CV**
Restoration	masked feature regression	image in-painting
Compression	Attribute Mask	PCA
Robustness	MGAE(?)	de-noising auto-encoders

Table 2: Comparison of Methods in GNN and CV

Methods	Formula	Notes
Node feature masking	$t(X) = M \circ X$	adaptive: centrality/Page Rank
Node feature shuffling	$t(X) = M \cdot X$	M is not identity Matrix
Edge Modification	$t(A) = M_1 \circ A + M_2 \circ (1 - A)$	
Graph Diffusion	$T = AD^{-1}/D^{-1/2}AD^{-1/2}$	Fourier Series: eigen? (complexity is high)
Subgraph Sampling	hybrid	similar to image cropping

Table 3: Graph Augmentation

2.2 Results

1. Early random walkbased contrastive methods (e.g., DeepWalk and GraphSAGE) and autoencoder-based generative methods (e.g., GAE and SIG-VAE) perform worse than the majority of graph SSL methods.

2. Methods in CV is not applicable.

3. Some representative contrast-based methods perform better than the generalization-based and auxiliary property-based methods,

4. integrate multiple pretext tasks can provide supervision signals from diverse perspectives

2.3 Questions

1. Auxiliary-property-based (topological): how to do property extraction in an efficient way?
Clustering based: structure-based clustering(?)/short distance/centrality(or betweenness)
Pair-relation based: cut edge between sub-graphs? or edge between nodes (in structure generation)?

2. Graph diffusion

3. Joint learning and unsupervised representation learning
   mutual information estimation, JS divergence, and masked Wasserstein distance.
4. The difference between L and p_φ

$$\theta^*, \varphi^* = \arg\min_{\theta^*, \varphi^*} L_{avg*}(p_\varphi(f_\theta(\widetilde{G}), g(G, f_\theta))) \tag{1}$$

5. Contrastive learning: cross-scale may be more powerful? Path-global and Context-global?
6. cold start and zero-shot?

3 Strategies for Pre-Training Graph Neural Networks

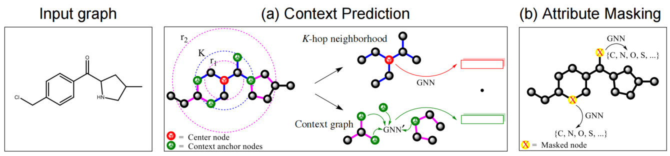

Figure 1: GNN

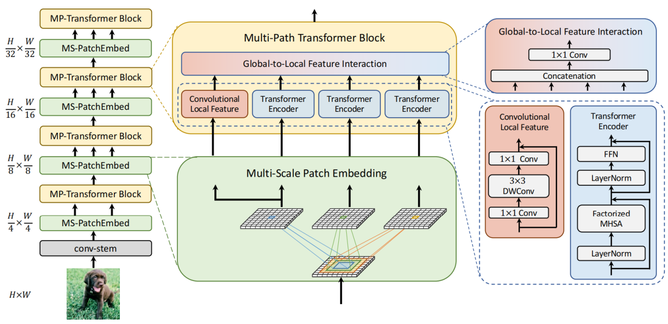

Figure 2: MPViT

1. **Existing Problems:**
   a. graph data from real-world applications often contain out-of-distribution samples
   b. node embeddings are not composable, and thus resulting graph embeddings (denoted by their classes, + and) that are created by pooling node-level embeddings are not separable.
   c. Instead, it requires substantial domain expertise to carefully select examples and target labels that are correlated with the downstream task of interest. (Learn the characteristics of different kinds of graph datasets?)
2. The key idea is to use easily accessible node-level information and encourage GNNs to capture domain-specific knowledge about nodes and edges, in addition to graph-level knowledge.
3. READOUT is a permutation-invariant function, such as averaging or a more sophisticated graph-level pooling function
4. Context Prediction and Attribute Masking
5. apply the context GNN to obtain node embeddings in the context graph, and then average embeddings of context anchor nodes to obtain a **fixed-length context embedding**.
6. from node-level regularization to graph level
7. scaffold split
8. Context Prediction + Graph-level multi-task supervised pre-training strategy gives the most promising performance
9. GIN and GINE

4 GraphCL

4.1 GConv

a. According to the official codes, I try to reproduce the results.
b. I find that it propagates all the outputs from hidden layers to the last.
c. kaiming_uniform seems better than others.

```
class GConv(nn.Module):
    def __init__(self, in_dims, hidden_dims, num_layers):
        super(GConv, self).__init__()
        self.convs = nn.ModuleList()
        self.bns = nn.ModuleList()
        self.in_dims = in_dims
        self.hidden_dims = hidden_dims
        self.num_layers = num_layers

        self.convs.extend([self.make_gin_conv(in_dims, hidden_dims)])
        self.convs.extend([self.make_gin_conv(hidden_dims, hidden_dims) for x in range(num_layers-1)
            ])
        self.bns.extend([nn.BatchNorm1d(hidden_dims) for x in range(num_layers)])

        # very interesting, combine different levels of hidden outputs
        project_dims = hidden_dims * num_layers
        self.project_heads = nn.Sequential(
            nn.Linear(project_dims, project_dims),
            nn.ReLU(inplace=True),
            nn.Linear(project_dims, project_dims),
        )

        self.reset_parameters()

    @staticmethod
    def make_gin_conv(in_dims, out_dims, eps=0, train_eps=False):
        return GINConv(
            nn = nn.Sequential(nn.Linear(in_dims, out_dims), nn.ReLU(inplace=True), nn.Linear(
                out_dims, out_dims),),
            eps=eps,
            train_eps=train_eps,
        )

    @staticmethod
    def linear_reset(layer):
        if isinstance(layer, nn.Linear):
            nn.init.kaiming_uniform_(layer.weight.data)
            nn.init.zeros_(layer.bias.data)

    def reset_parameters(self):
        for layer in self.project_heads:
            self.linear_reset(layer)
        for conv in self.convs:
            for layer in conv.nn:
                self.linear_reset(layer)

    # multiple layer outputs is used here
    def forward(self, x, edge_index, batch):
        out = x
        outputs = []
        for conv, bn in zip(self.convs, self.bns):
            out = bn(F.relu(conv(out, edge_index)))
            outputs.append(out)
        gs = [global_mean_pool(out, batch) for out in outputs]
        out, g = [torch.cat(x, dim=1) for x in [outputs, gs]]
        return out, g
```

4.2 Encoder

a. Minimize g1 and g.
b. Use g as embedding to feed into SVMEvaluator.

```
class Encoder(torch.nn.Module):
```

```
    def __init__(self, encoder, augmentor):
        super(Encoder, self).__init__()
        self.encoder = encoder
        self.augmentor = augmentor

    def forward(self, x, edge_index, batch):
        aug1, aug2 = self.augmentor
        x1, edge_index1, _ = aug1(x, edge_index)
        x2, edge_index2, _ = aug2(x, edge_index)
        z, g = self.encoder(x, edge_index, batch)
        z1, g1 = self.encoder(x1, edge_index1, batch)
        z2, g2 = self.encoder(x2, edge_index2, batch)
        return z, g, z1, z2, g1, g
```

4.3 Results

Intentionally set the random seed here as 1234 and random walk as 10.

4.3.1 MUTAG

Theoretical: $86.80 \pm 1.34\%$
Experiment: 85%
Without initalization: 70%

4.4 NCI1

Theoretical: $77.87 \pm 0.41\%$
Experiment: 65%
Without initialization: 64%