GNN-PRP

Haoyang Ling

March 28, 2022

1 Ideas

Based on the code of graphcl and adgcl, I come up with the following idea.

1. use trainable graph augumentation inspired by self-attention and GraphCL: tanh/softmax, bernouli
2. use mutual information trained by the enhanced loss function: adversarial loss
3. try to resolve the loss up-and-down problems
4. strategies for updating different encoders: parallel/momentum

1.1 Codes

1.1.1 Edge and Node Weights

Hope the dropping edges and nodes leads to dropping/generating sub-graphs

```
def edge_weight(self, z, edge_index):
    src, dst = edge_index[0], edge_index[1]

    src_q = self.Q(src_embedding)
    dst_k = self.K(dst_embedding)
    src_v = self.V(src_embedding)

    # edge_weight = src_q @ dst_k.t()
    edge_weight = src_q * dst_k

    edge_weight = edge_weight / math.sqrt(edge_weight.shape[1])

    edge_weight = (F.softmax(edge_weight, dim=1) @ src_v).mean(dim=1)
    # edge_weight = self.V(torch.tanh(edge_weight)).mean(dim=1)

    node_feat_weight = (self.N(z) / math.sqrt(self.hidden_dims)).mean()

    return edge_weight, node_feat_weight
```

1.1.2 Encoders updating

```
def augment_encoder_update(self, inverse_momentum=1e-2):
    for param_q, param_k in zip(self.encoder.parameters(), self.augment_encoder.parameters()):
        param_k.data += (1 - inverse_momentum) * (param_q.data.detach() - param_k.data)
        # param_k.data = param_q.data.clone().detach()
```

1.1.3 Evaluator

```
class LogisticRegression(nn.Module):
    def __init__(self, num_features, num_classes):
        super(LogisticRegression, self).__init__()
        self.fc = nn.Linear(num_features, num_classes)
        torch.nn.init.xavier_uniform_(self.fc.weight.data)

    def forward(self, x):
        z = self.fc(x)
        return z
```

```
    def reg_loss(self, reg=1e-2):
        if reg == 0:
            return 0
        else:
            return reg * torch.mean(self.fc.weight.data**2)
```

1.1.4 Training

```
x, edge_index, batch = data.x, data.edge_index, data.batch

# if data.edge_attr is not None:
#     edge_attr = data.edge_attr
# else:
#     edge_attr = torch.zeros(edge_index.shape[1], 1, device=device)

z, _ = encoder_model(x, edge_index, batch)

edge_weights, node_feat_weights = encoder_model.edge_weight(z=z.clone().detach(), edge_index=
    edge_index, edge_attr=edge_attr)

gs = []
gs_ad = []
for drop_rate in torch.tensor([.3, .4], dtype=torch.float):
    edge = edge_index.clone()
    edge_prob = feature_drop_weights(edge_weights, tau, device=device)
    # feat_prob = feature_drop_weights(node_feat_weights, tau, device=device)

    # x_aug = drop_feature_weighted(x, feat_prob, 0.1)
    x_aug = x

    edge = drop_edge_weighted(edge, edge_prob, drop_rate)

    _, g_aug = encoder_model.encoder(x_aug, edge, batch)
    encoder_model.encoder_update()
    _, g_ad = encoder_model.augment_encoder(x_aug, edge, batch)
    gs.append(g_aug)
    gs_ad.append(g_ad)

g1, g2 = gs[0], gs[1]

t1 = encoder_model.mlp(g1)
t2 = encoder_model.mlp(g2)


# can we compare the residuals
# train it to the learn the noise instead if parallel training
t3 = encoder_model.aug_mlp(gs_ad[0])
t4 = encoder_model.aug_mlp(gs_ad[1])

loss = encoder_model.loss(t1, t2) - alpha * (encoder_model.loss(t3, t4)) + reg * encoder_model.
    reg_loss()
loss.backward()
optimizer.step()

epoch_loss += loss.item()
```

1.2 Results

Compare tanh-parallel with softmax-frozen.

1.2.1 Some Figures with Better Results

1. Up-and-down figures show the discontinuity of training. It shows that the dropping schemes may break the important structure of it **as we don't have the strategy to add some edges**.
2. It seems that better training loss leads to better performance. Initialization and learning rate may account for it.

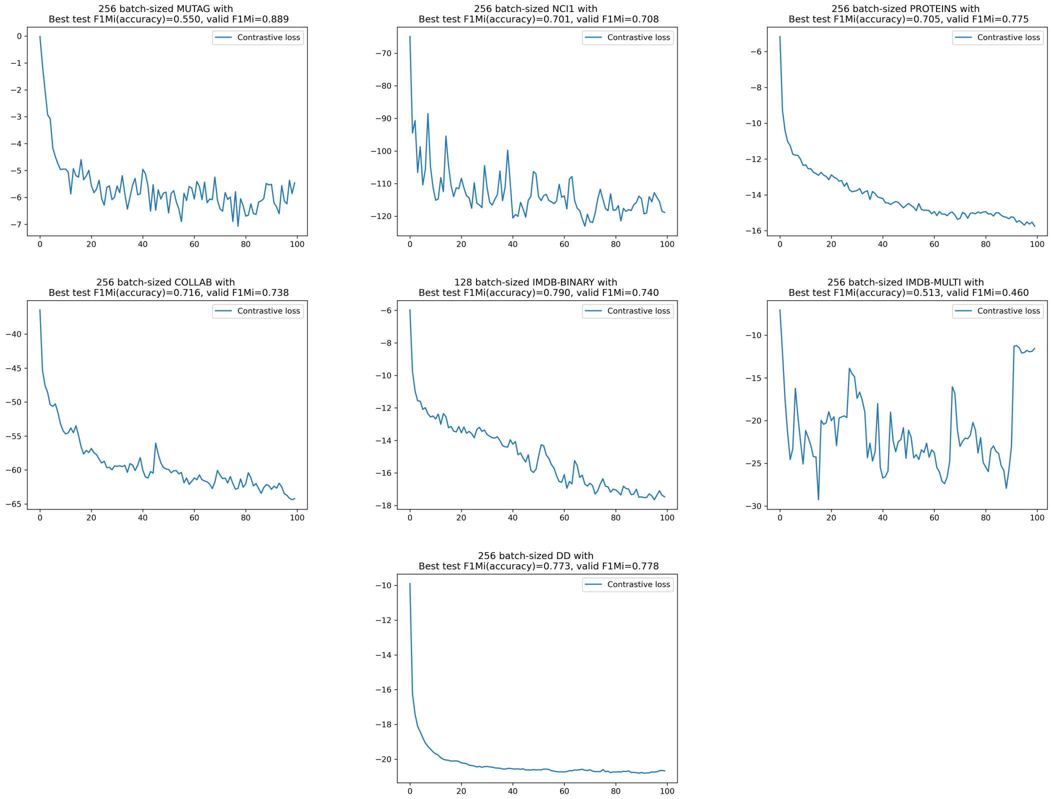

Figure 1: Contrastive Loss with Softmax-frozen

1.3 Reflection

1. Frozen may focus the adversarial loss to train the transformation only.
2. using edge_attr in edge weights not improving the graphs as some don't have the edge_attr
3. Some similarities from self-attention mechanism in transformer. How to calculate the attention? How to define the key and value pairs?
4. Find a proper way to embedding evaluation.

Dataset	PROTEINS				PROTEINS			
Parameter	36297							
Time	Before		After		Before		After	
Mode	Valid	Test	Valid	Test	Valid	Test	Valid	Test
1	0.739	0.741	0.820	0.750	0.775	0.759	0.757	0.732
2	0.775	0.705	0.694	0.750	0.793	0.741	0.829	0.812
3	0.784	0.741	0.757	0.768	0.703	0.741	0.802	0.741
4	0.820	0.643	0.748	0.795	0.757	0.786	0.760	0.759
5	0.757	0.741	0.775	0.705	0.766	0.750	0.757	0.759
Mean	0.775	**0.714**	0.759	**0.754**	0.759	**0.755**	0.781	**0.761**
Std	0.027	**0.038**	0.041	**0.029**	0.030	**0.017**	0.029	**0.028**
Max	0.820	**0.741**	0.820	**0.795**	0.793	**0.786**	0.829	**0.812**

Dataset	IMDB-BINARY				IMDB-BINARY			
Parameter	36039							
Time	Before		After		Before		After	
Mode	Valid	Test	Valid	Test	Valid	Test	Valid	Test
1	0.690	0.620	0.670	0.730	0.660	0.590	0.690	0.680
2	0.710	0.770	0.700	0.690	0.750	0.760	0.770	0.690
3	0.740	0.670	0.770	0.750	0.750	0.710	0.680	0.680
4	0.700	0.740	0.660	0.670	0.760	0.690	0.750	0.710
5	0.660	0.710	0.700	0.770	0.690	0.670	0.730	0.740
Mean	0.700	**0.702**	0.700	**0.722**	0.722	**0.684**	0.724	**0.700**
Std	0.026	**0.053**	0.038	**0.037**	0.040	**0.056**	0.034	**0.023**
Max	0.740	**0.770**	0.770	**0.770**	0.760	**0.760**	0.770	**0.740**

Dataset	IMDB-MULTI				IMDB-MULTI			
Parameter	36039							
Time	Before		After		Before		After	
Mode	Valid	Test	Valid	Test	Valid	Test	Valid	Test
1	0.433	0.513	0.473	0.413	0.500	0.453	0.520	0.447
2	0.487	0.460	0.527	0.460	0.547	0.467	0.533	0.467
3	0.480	0.480	0.520	0.433	0.513	0.440	0.487	0.413
4	0.453	0.400	0.480	0.487	0.580	0.393	0.553	0.480
5	0.487	0.380	0.460	0.513	0.540	0.433	0.507	0.420
Mean	0.468	**0.447**	0.492	**0.461**	0.536	**0.437**	0.520	**0.445**
Std	0.022	**0.050**	0.027	**0.036**	0.028	**0.025**	0.022	**0.026**
Max	0.487	**0.513**	0.527	**0.513**	0.580	**0.467**	0.553	**0.480**

Dataset	NCI1				NCI1			
Parameter	40683							
Time	Before		After		Before		After	
Mode	Valid	Test	Valid	Test	Valid	Test	Valid	Test
1	0.725	0.713	0.713	0.706	0.723	0.684	0.633	0.606
2	0.700	0.681	0.691	0.727	0.689	0.715	0.672	0.640
3	0.720	0.691	0.754	0.703	0.713	0.706	0.657	0.635
4	0.720	0.708	0.701	0.725	0.732	0.706	0.640	0.625
5	0.715	0.689	0.708	0.701	0.766	0.706	0.620	0.642
Mean	0.716	**0.696**	0.713	**0.712**	0.725	**0.703**	0.644	**0.630**
Std	0.009	**0.012**	0.022	**0.011**	0.025	**0.010**	0.018	**0.013**
Max	0.725	**0.713**	0.754	**0.727**	0.766	**0.715**	0.672	**0.642**

Dataset	DD				DD			
Parameter	47391							
Time	Before		After		Before		After	
Mode	Valid	Test	Valid	Test	Valid	Test	Valid	Test
1	0.735	0.714	0.761	0.798	0.740	0.689	0.829	0.748
2	0.709	0.681	0.812	0.739	0.863	0.672	0.812	0.756

3	0.795	0.714	0.821	0.689				
4	0.744	0.697	0.821	0.765				
5	0.803	0.672	0.778	0.773				
Mean	0.757	**0.696**	0.799	**0.753**	0.802	**0.681**	0.821	**0.752**
Std	0.036	**0.017**	0.025	**0.037**	0.062	**0.008**	0.008	**0.004**
Max	0.803	**0.714**	0.821	**0.798**	0.863	**0.689**	0.829	**0.756**

Dataset	**COLLAB**				**COLLAB**			
Parameter	**36039**							
Time	Before		After		Before		After	
Mode	Valid	Test	Valid	Test	Valid	Test	Valid	Test
1	0.644	0.672	0.746	0.698	0.660	0.644	0.708	0.672
2	0.698	0.670	0.722	0.690				
3	0.696	0.692	0.722	0.664				
4	0.686	0.676	0.692	0.700				
5	0.664	0.662	0.738	0.716				
Mean	0.678	**0.674**	0.724	**0.694**				
Std	0.021	**0.010**	0.019	**0.017**				
Max	0.698	**0.692**	0.746	**0.716**				

Dataset	**MUTAG**				**MUTAG**			
Parameter	**36039**							
Time	Before		After		Before		After	
Mode	Valid	Test	Valid	Test	Valid	Test	Valid	Test
1	0.889	0.650	0.944	0.700	1.000	0.900	0.944	0.900
2	0.833	0.700	0.944	0.850	0.944	0.800	1.000	0.700
3	0.889	0.650	0.833	0.700	1.000	0.950	0.944	0.950
4	1.000	0.950	0.889	0.850	0.899	0.950	1.000	0.800
5	0.889	0.800	0.944	0.900	0.944	0.900	0.944	0.900
Mean	0.900	**0.750**	0.911	**0.800**	0.957	**0.900**	0.966	**0.850**
Std	0.055	**0.114**	0.044	**0.084**	0.038	**0.055**	0.027	**0.089**
Max	1.000	**0.950**	0.944	**0.900**	1.000	**0.950**	1.000	**0.950**