Report

Description of The Datasets

I test the proposed framework's efficiency on FREE SOLVE regression tasks.

dataset	Category	#tasks	#compond	Task type	metric
FreeSolv	Physical chemistry	1	642	regression	RMSE

FreeSolv: is selected from the Free Solvation Database, which contains the hydration free energy of small molecules in water from experiments and alchemical free energy calculations.

2 Dataset Splitting

Since using the scaffold splitting for molecular property prediction tasks can better evaluate out-of-distribution generalization abilities of the models, we evaluate all models on the scaffold splittings of the datasets. Our scaffold splitting is similar to Wu et al.[1] and Yang et al.[17] In this scaffold splitting technique, molecules are partitioned into bins based on their Murcko scaffold calculated by RDKit.[18] Any bins larger than half of the desired test set size are placed into the training set to guarantee the scaffold diversity of the validation and test sets. All remaining bins are placed randomly into the training, validation, and test sets until each set has reached its desired size.

In this project, I use scaffold0 and run the code.

Details about molecular graphs, molecular quotient graphs, and functional groups in regression datasets

	Free solve
Dataset size	642
Total #nodes in MG	5600
Total #nodes in Q-non- FGs	2751
Total #nodes in MQG	2111
Abstraction ratio	0.38
Avg #nodes per mol in MG	8.72
Avg #nodes per mol in Q-non-FGs	4.28
Avg #nodes per mol in MQG	3.29
Total #FGs	1128
Total #unique FGs	61
Avg #FGs per mol	1.76
#Molecules without FGs	71

In this roject after considering different batchsize, I decided to choose 16. Number of Epoch is 100

The model is run in tree different architecture at the first we use GCN

```
class GNN(nn.Module):
    def __init__(self, config, global_size = 200, num_tasks = 1):
        super().__init__()
        self.config = config
        self.num_tasks = num_tasks

        # Node feature size
        self.node_feature_size = self.config.get('node_feature_size', 127)

        # Edge feature size
        self.edge_feature_size = self.config.get('edge_feature_size', 12)

        # Hidden size
        self.hidden_size = self.config.get('hidden_size', 100)

        self.conv1 = GraphConv(self.node_feature_size, self.hidden_size, allow_zero_in_degree = True)
        self.conv2 = GraphConv(self.hidden_size, self.num_tasks, allow_zero_in_degree = True)

    # def forward(self, g, in_feat):
    def forward(self, mol_dgl_graph, globals):
        mol_dgl_graph.ndata["v"]= mol_dgl_graph.ndata["v"][:,:self.node_feature_size]
        mol_dgl_graph.edata["e"] = mol_dgl_graph.edata["e"][:,:self.edge_feature_size]
        h = self.conv1(mol_dgl_graph, mol_dgl_graph.ndata["v"])
        h = F.relu(h)
        h = self.conv2(mol_dgl_graph, h)
        mol_dgl_graph.ndata["h"] = h
        return dgl.mean_nodes(mol_dgl_graph, "h")
```

Second we use GraphSage AS FOLLOW

```python
class GraphSAGE(nn.Module):
    def __init__(self, config, global_size = 200, num_tasks = 1):
        super().__init__()
        self.config = config
        self.num_tasks = num_tasks

        # Node feature size
        self.node_feature_size = self.config.get('node_feature_size', 127)

        # Edge feature size
        self.edge_feature_size = self.config.get('edge_feature_size', 12)

        # Hidden size
        self.hidden_size = self.config.get('hidden_size', 100)

        self.conv1 = SAGEConv(self.node_feature_size, self.hidden_size, aggregator_type='mean')
        self.conv2 = SAGEConv(self.hidden_size, self.hidden_size , aggregator_type='mean')

    def forward(self, mol_dgl_graph, globals):
        mol_dgl_graph.ndata["v"] = mol_dgl_graph.ndata["v"][:,:self.node_feature_size]
        mol_dgl_graph.edata["e"] = mol_dgl_graph.edata["e"][:,:self.edge_feature_size]
        h = self.conv1(mol_dgl_graph, mol_dgl_graph.ndata["v"])
        h = F.relu(h)
        h = self.conv2(mol_dgl_graph, h)
        mol_dgl_graph.ndata["h"] = h
        return dgl.mean_nodes(mol_dgl_graph, "h")
```

At last I use GAT

```python
import torch.nn as nn
import torch.nn.functional as F
import dgl

class GAT(nn.Module):
    def __init__(self, in_feats, hidden_feats, out_feats):
        super(GAT, self).__init__()
        self.conv1 = dgl.nn.pytorch.conv.GATConv(in_feats, hidden_feats, num_heads=8)
        self.conv2 = dgl.nn.pytorch.conv.GATConv(hidden_feats * 8, out_feats, num_heads=1)

    def forward(self, g, x):
        h = self.conv1(g, x).flatten(1)
        h = F.elu(h)
        h = F.dropout(h, p=0.5, training=self.training)
        h = self.conv2(g, h).mean(1)
        return F.log_softmax(h, dim=1)
```

Now we want to consider the result :

THE RESULT OF GCN

epoch	Training loss	Valid score	Best valid score until now
1	22.653	6.584	6.584
2	21.591	6.443	6.443
3	20.305	6.302	6.302
4	19.228	6.156	6.156
5	19.506	6.003	6.003
6	17.179	5.836	5.836
7	16.175	5.683	5.683
8	15.553	5.526	5.526
9	14.444	5.373	5.373
10	14.319	5.224	5.224
11	13.196	5.074	5.074
12	12.647	4.943	4.943
13	17.703	5.036	5.036
14	12.657	4.911	4.911
92	7.476	3.732	3.732
93	7.292	3.721	3.721
94	7.244	3.733	3.721
95	7.213	3.734	3.721
96	7.163	3.731	3.721
97	7.744	3.728	3.721
98	7.115	3.72	3.72
99	7.023	3.722	3.72
100	7.084	3.72	3.72

THE RESULT OF GRAPHSAGE

epoch	Training loss	Valid score	Best valid score until now
1	22.653	6.584	6.584
2	21.591	6.443	6.443
3	20.305	6.302	6.302
4	19.228	6.156	6.156
5	19.506	6.003	6.003
92	7.262	3.758	3.758
93	7.201	3.752	3.749
94	7.148	3.736	3.736

95	7.186	3.735	3.735
96	7.256	3.737	3.735
97	7.179	3.731	3.731
98	7.437	3.724	3.724
99	10.685	3.724	3.724
100	6.942	3.722	3.722

THE RESULT OF GAT

epoch	Training loss	Valid score	Best valid score until now
1	24.548	6.799	6.799
2	23.064	6.634	6.634
3	21.647	6.456	6.456
4	20.328	6.298	6.298
5	19.083	6.124	6.123
.92	7.034	3.753	3.746
93	6.948	3.752	3.746
94	6.918	3.757	3.746
95	6.893	3.756	3.746
96	6.925	3.758	3.746
97	6.820	3.762	3.746
98	6.832	3.753	3.746
99	6.799	3.757	3.746

TEST SCORE		
GCN	GRAPHSAGE	GAT
3.720	3.378	3.382

Refrence:

Hajiabolhassan .hossein, Taheri.Zahra and et.al “Supporting Information for FunQG: Molecular Representation Learning via Quotient Graphs