Homework 3

Deepak Akhare

November 15, 2022

Problem 1:

B. A python class that performs GCN propagation rule has been implemented using the following rule -

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right). \tag{1}$$

code -

```python
class GCN:
    """
        Graph convolutional layer
    """
    def __init__(self, in_features, out_features):
        # -- initialize weight
        self.W = torch.nn.Parameter(torch.rand((in_features, out_features))*0.01,
                                    requires_grad=True).type(torch.
                                    DoubleTensor).to(device)

        # -- non-linearity

    def __call__(self, A, H):
        # -- GCN propagation rule
        I = torch.eye(A.shape[1]).to(device)
        Hn = []
        for i in range(len(A)):
            AI      = A[i]+I
            D       = I*torch.sum(AI,0)
            Dinv_hf = torch.sqrt(torch.inverse(D))
            DAD     = torch.mm(torch.mm(Dinv_hf,AI),Dinv_hf)
            HW      = torch.mm(H[i], self.W)
            Hn.append(torch.unsqueeze(torch.mm(DAD, HW), 0))
        return torch.cat((Hn))
```

C. A python class for graph pooling layer that uses sum as the pooling function has been implemented - code -

```python
class GraphPooling:
```

```
    """
        Graph pooling layer
    """
    def __init__(self):
        pass

    def __call__(self, H):
        # -- multi-set pooling operator
        return torch.sum(H, 1).type(torch.DoubleTensor).to(device)
```

D. A neural network model to predict the Highest Occupied Molecular Orbital (HOMO) energy of the molecule -

code -

```
class MyModel(nn.Module):
    """
        Regression  model
    """
    def __init__(self):
        super(MyModel, self).__init__()
        # -- initialize layers
        hidd_features = 3
        self.gcn_layer = GCN(in_features = sigs[0].shape[1], out_features =
                                                    hidd_features)
        self.Gpool = GraphPooling()
        # self.fc = nn.Sequential(nn.Linear(6,1),nn.LeakyReLU())
        self.fc = nn.Linear(hidd_features,1)
        self.relu = nn.ReLU()
    def forward(self, A, h0):
        h1 = self.gcn_layer(A,h0)
        h1 = self.relu(h1)
        h2 = self.Gpool(h1)
        h3 = self.fc(h2.type(torch.float))
        return h3
```

E. errors for 200 epochs -

F.tested model on 1000 points -

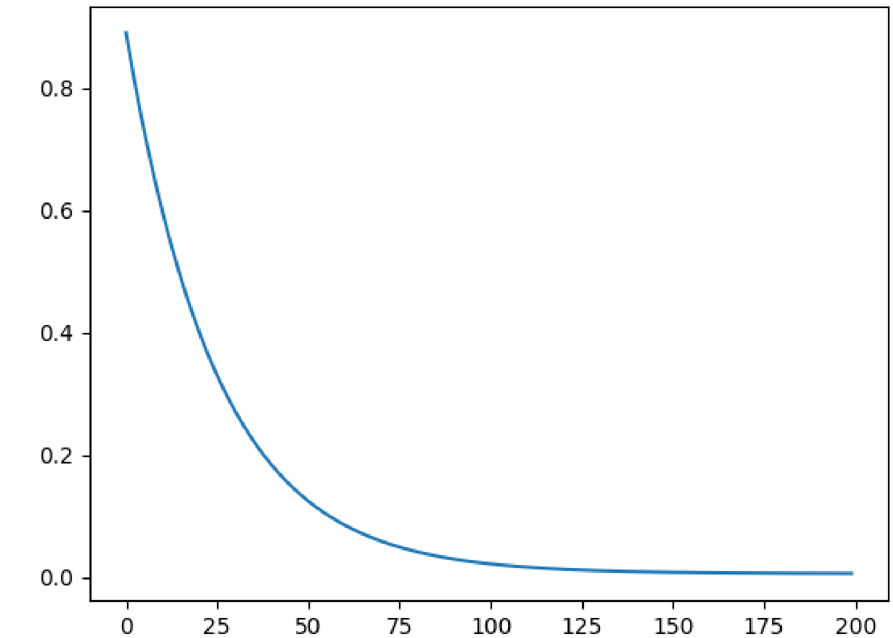


Figure 1: Training error for 200 epoch

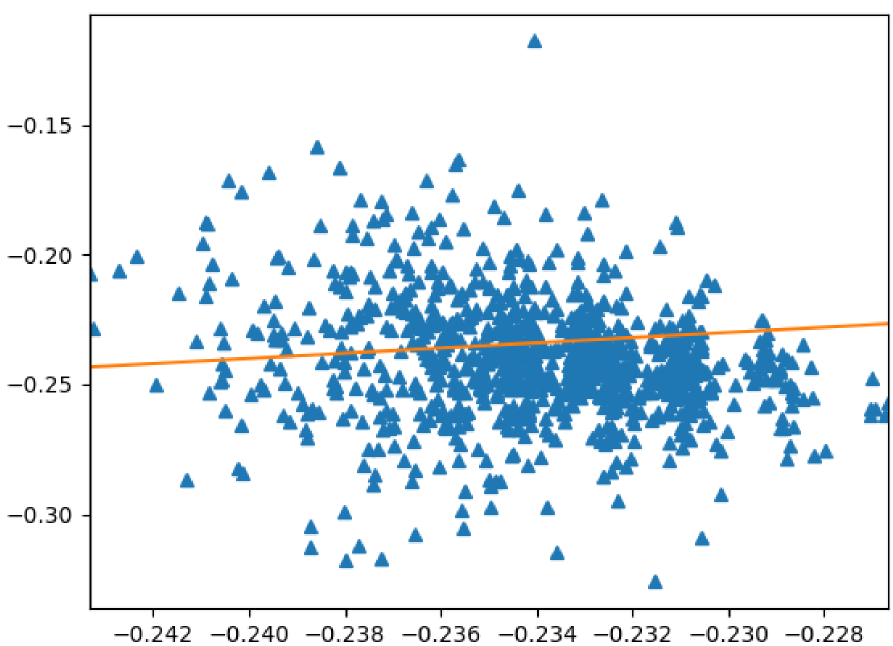
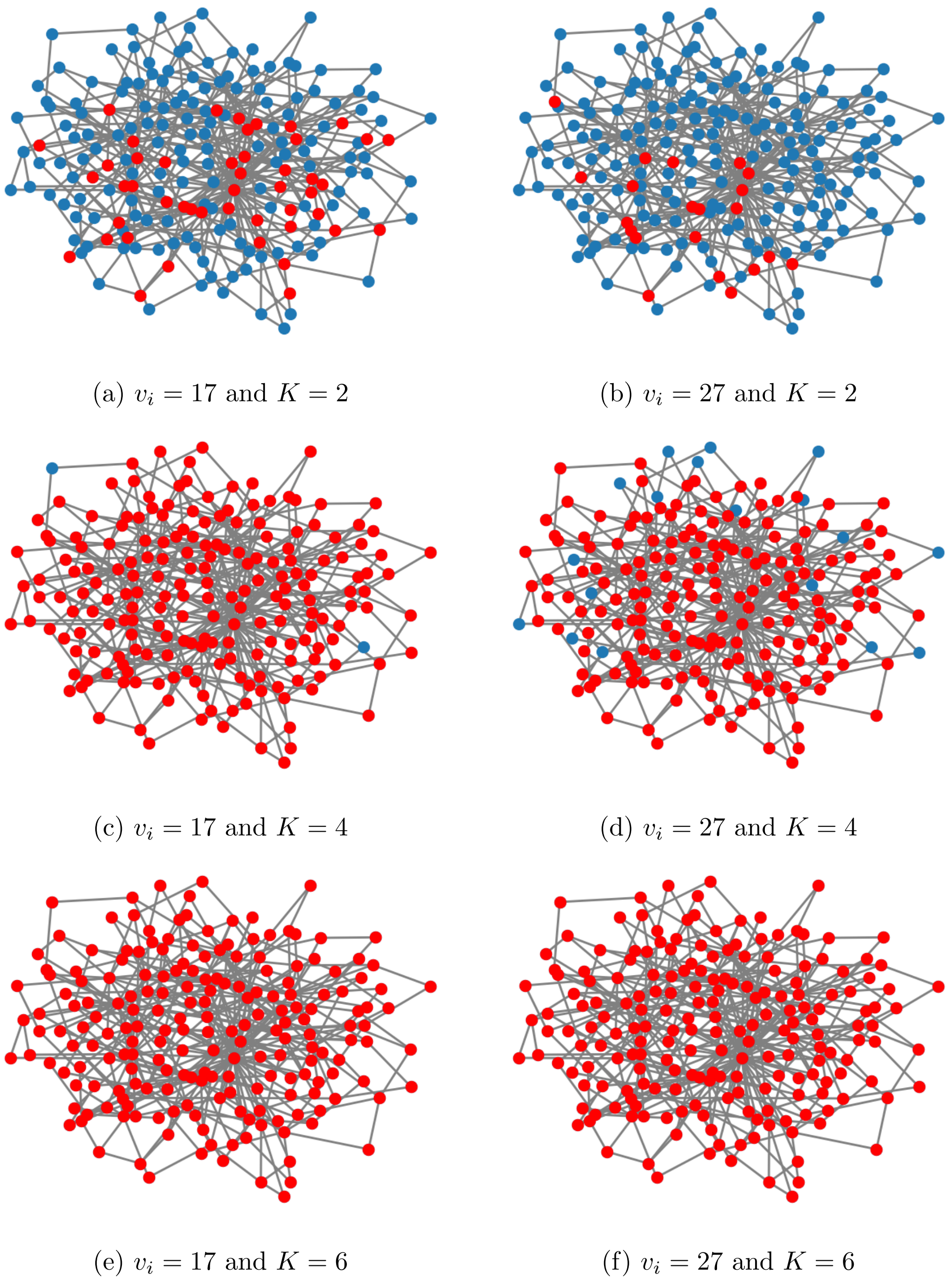


Figure 2: Model evaluation on 1000 test data points

Problem 2:

B. The effective range of nodes $v_i = 17$ and $v_i = 27$ for K = 2, 4, and 6 message-passing layers can be seen below.



(a) $v_i = 17$ and $K = 2$

(b) $v_i = 27$ and $K = 2$

(c) $v_i = 17$ and $K = 4$

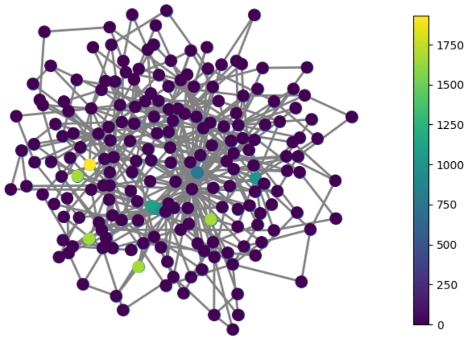
(d) $v_i = 27$ and $K = 4$

(e) $v_i = 17$ and $K = 6$

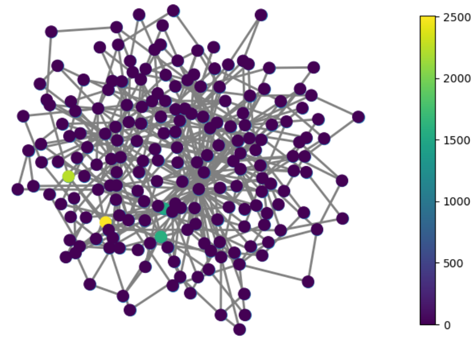
(f) $v_i = 27$ and $K = 6$

Figure 3: Effective range

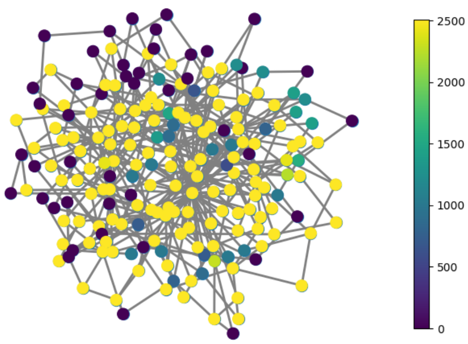
C. and D.



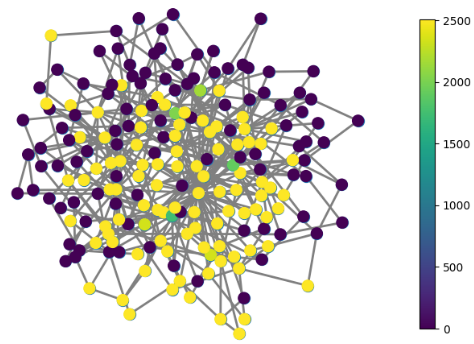
(a) Model 1 and $v_i = 17$



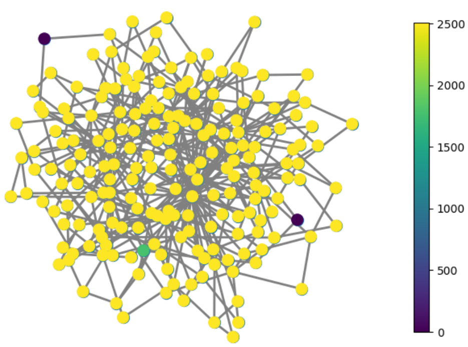
(b) Model 1 and $v_i = 27$



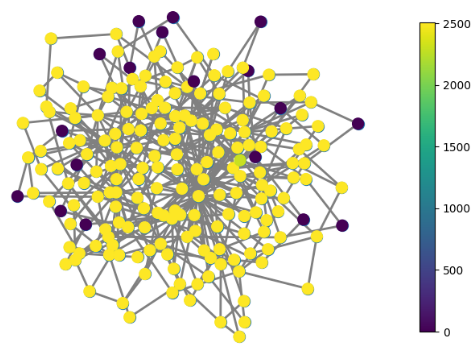
(c) Model 2 and $v_i = 17$



(d) Model 2 and $v_i = 27$



(e) Model 3 and $v_i = 17$



(f) Model 3 and $v_i = 27$

Figure 4: Influence score

Conclusion:

Both plots show that as the number of GCN layers increases, the effective range and its influence on neighboring nodes increase. For two GCN layers, the information coming to a node is the only function of its local neighbor nodes. However, as the number of GCN layers increases, the information coming to a node will now be a function of almost every node. Finally, for 6 layers, every node of the given graph is influenced by all the nodes in the entire graph. As the GCN aggregates information coming from nodes, GNN with more than 3 GCN layers will lead to over-smoothing, and, therefore, the feature vectors for every node in the graph will be similar. This will hinder the learning capability of the GNN. Therefore, it is recommended a GNN should only have 2-3 GCN layers.