HW3

Machine learning with Graphs (AIM5056_41)

2022.11.13

2022711835 Junhee Kwon

1. Over-smoothing problem of GCN (20 pts)

To find out that the number of GCN layers matters to the over-smoothing problem, I used a pygcn code to visualise the over-smoothing problem of GCN. For that, I modified the models.py code to make more GCN layers. As you can see, the classification method of pygcn works quiet well on case 1 and 2. Case 1 is the original code for the pygcn and case 2 is the modified code with 3 gcn layers.

However, pygcn model cannot classify the nodes well from the case 3 to 5. The representations of the nodes are lumped together like a graph of linear function. And even for the case 6, which has 7 GCN layers, there are only 5 points in the graph. It means that all the different nodes are represented in only 5 points.

As we can see in the visualized graph, the number of GCN layers matter to the classification task. If the number of the layer gets more, each node is represented by the nodes located a lot of steps away. Then, there are possibilities that all nodes are represented based on all the nodes.

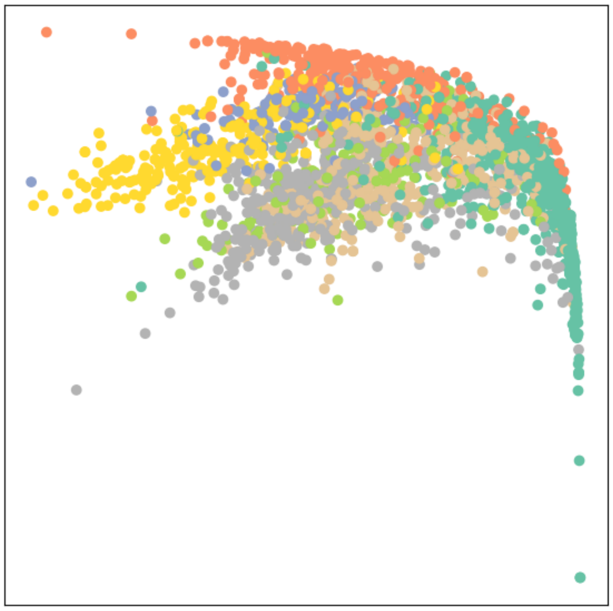
Case 1.

```
#models.py

class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc2(x, adj)
        return F.log_softmax(x, dim=1)
```



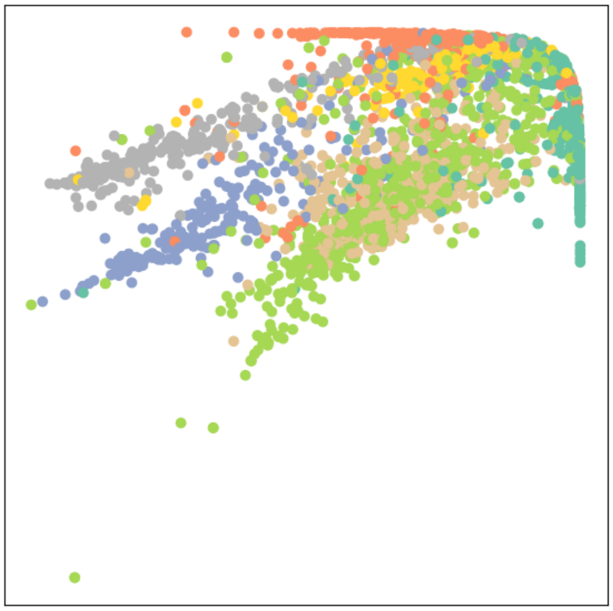
```
Epoch: 0200 loss_train: 0.4517 acc_train: 0.9071 loss_val: 0.7096 acc_val: 0.8067 time: 0.0053s
Optimization Finished!
Total time elapsed: 1.1092s
```

Case 2.

```
#models.py

class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nhid)
        self.gc3 = GraphConvolution(nhid, nclass)
        self.dropout = dropout
    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc2(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc3(x, adj)
        return F.log_softmax(x, dim=1)
```



```
Epoch: 0200 loss_train: 0.2218 acc_train: 0.9357 loss_val: 0.6630 acc_val: 0.8133 time: 0.0066s
Optimization Finished!
Total time elapsed: 1.4836s
```

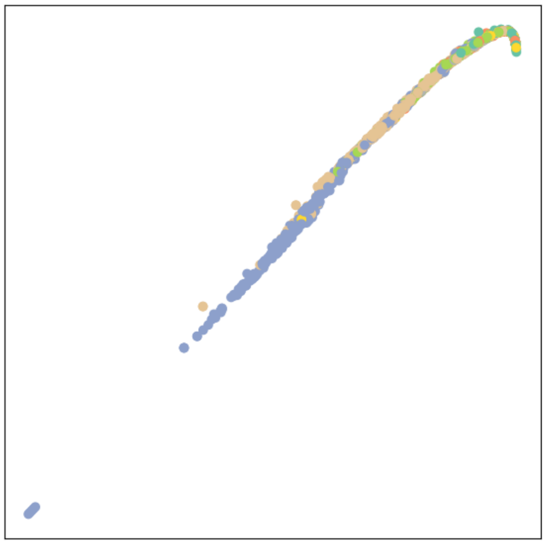
Case 3.

```
#models.py

class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nhid)
        self.gc3 = GraphConvolution(nhid, nhid)
        self.gc4 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc2(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc3(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc4(x, adj)
        return F.log_softmax(x, dim=1)
```



```
Epoch: 0200 loss_train: 0.6454 acc_train: 0.7286 loss_val: 1.1771 acc_val: 0.6033 time: 0.0074s
Optimization Finished!
Total time elapsed: 1.6478s
```

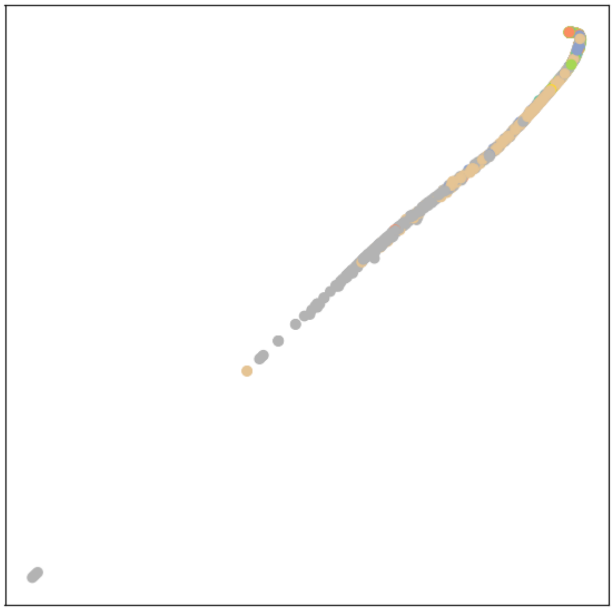
Case 4.

```
#models.py

class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nhid)
        self.gc3 = GraphConvolution(nhid, nhid)
        self.gc4 = GraphConvolution(nhid, nhid)
        self.gc5 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc2(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc3(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc4(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc5(x, adj)
        return F.log_softmax(x, dim=1)
```



```
Epoch: 0200 loss_train: 0.6945 acc_train: 0.7000 loss_val: 1.3271 acc_val: 0.5600 time: 0.0093s
Optimization Finished!
Total time elapsed: 1.8654s
```

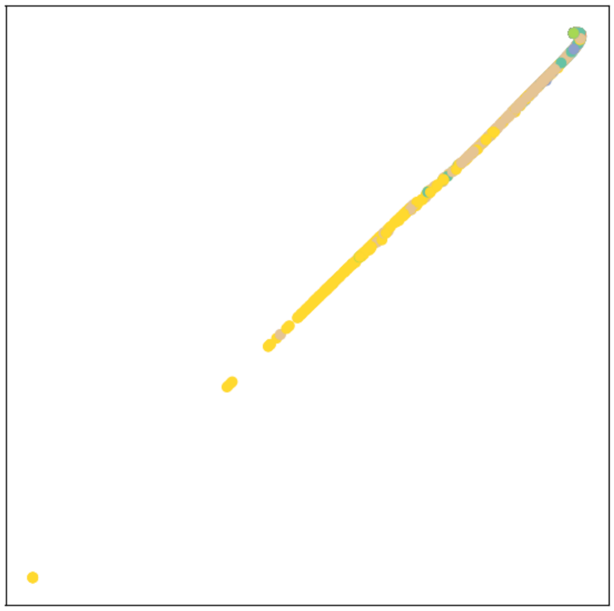
Case 5.

```
#models.py

class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nhid)
        self.gc3 = GraphConvolution(nhid, nhid)
        self.gc4 = GraphConvolution(nhid, nhid)
        self.gc5 = GraphConvolution(nhid, nhid)
        self.gc6 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc2(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc3(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc4(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc5(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc6(x, adj)
        return F.log_softmax(x, dim=1)
```

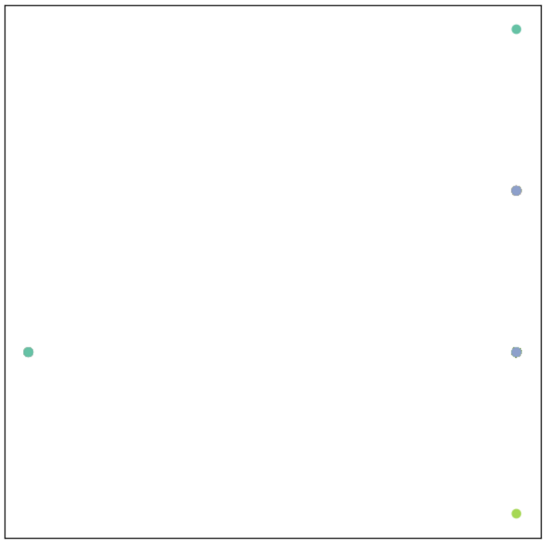


```
Epoch: 0200 loss_train: 0.8481 acc_train: 0.6429 loss_val: 1.2974 acc_val: 0.5400 time: 0.0104s
Optimization Finished!
Total time elapsed: 1.9390s
```

Case 6.

```
class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()
        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nhid)
        self.gc3 = GraphConvolution(nhid, nhid)
        self.gc4 = GraphConvolution(nhid, nhid)
        self.gc5 = GraphConvolution(nhid, nhid)
        self.gc6 = GraphConvolution(nhid, nhid)
        self.gc7 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc2(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc3(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc4(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc5(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc6(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc7(x, adj)
        return F.log_softmax(x, dim=1)
```



```
Epoch: 0200 loss_train: 1.8096 acc_train: 0.2929 loss_val: 1.8018 acc_val: 0.3500 time: 0.0113s
Optimization Finished!
Total time elapsed: 2.2699s
```

2. Skip-connection (20 pts)

Skip-connection is the method to give the more weight on the original representation while in the representation process. I applied the skip connection method to the case 6 of the problem 1, because I thought it would be interesting to make the worst result better.

Based on the “Deep Residual Learning for Image Recognition”, I implemented the formula of the paper:

$y = F(x, Wi) + x$

As you can see in the forward function of the GCN class, x is added to the input for the layers like below:

1. Case 6 with 2 skip connection
2. Case 6 with 3 skip connection
3. Case 6 with 4 skip connection
4. Case 6 with 5 skip connection

As the skip connection method is added, the classification work of the node representaions result better. The result of Case 6 of the problem 1 was really bad. There was no difference between the node representations. The differences were shown when I applied the skip connection method.

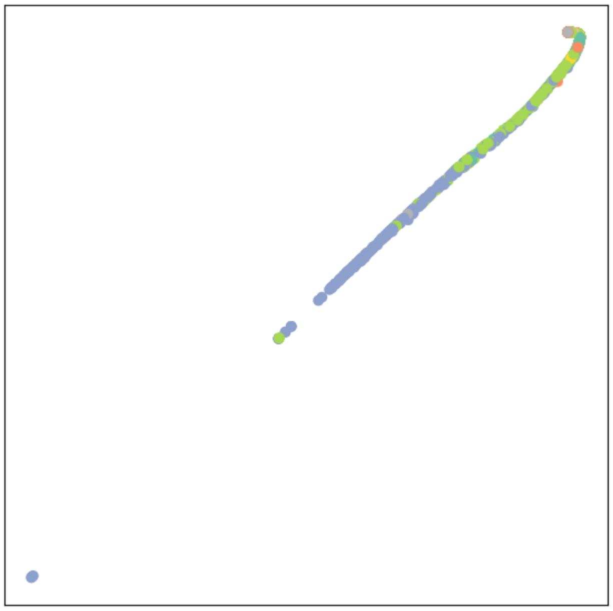
When I added 2 skip connection, I could find out that the result was similar to case 4 and 5. As I increase the number of skip connection, the result got better. In the figure of 3, 4, 5 skip connection, the results are similar to Case 2 of the problem 1.

Case 7 with 2 skip connection

```
class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nhid)
        self.gc3 = GraphConvolution(nhid, nhid)
        self.gc4 = GraphConvolution(nhid, nhid)
        self.gc5 = GraphConvolution(nhid, nhid)
        self.gc6 = GraphConvolution(nhid, nhid)
        self.gc7 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc2(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc3(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc4(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc5(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc6(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc7(x, adj)
        return F.log_softmax(x, dim=1)
```



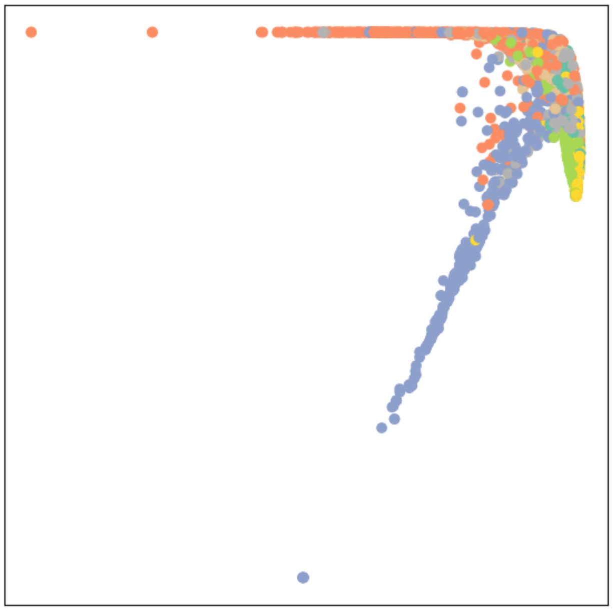
```
Epoch: 0200 loss_train: 0.6873 acc_train: 0.6929 loss_val: 1.4135 acc_val: 0.5633 time: 0.0109s
Optimization Finished!
Total time elapsed: 2.2261s
```

Case 7 with 3 skip connection

```
class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nhid)
        self.gc3 = GraphConvolution(nhid, nhid)
        self.gc4 = GraphConvolution(nhid, nhid)
        self.gc5 = GraphConvolution(nhid, nhid)
        self.gc6 = GraphConvolution(nhid, nhid)
        self.gc7 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc2(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc3(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc4(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc5(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc6(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc7(x, adj)
        return F.log_softmax(x, dim=1)
```



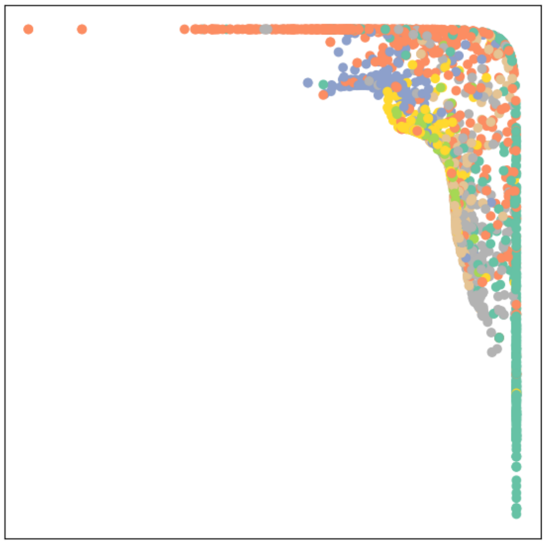
```
Epoch: 0200 loss_train: 0.3836 acc_train: 0.8786 loss_val: 1.3024 acc_val: 0.7500 time: 0.0112s
Optimization Finished!
Total time elapsed: 2.2284s
```

Case 7 with 4 skip connection

```
class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nhid)
        self.gc3 = GraphConvolution(nhid, nhid)
        self.gc4 = GraphConvolution(nhid, nhid)
        self.gc5 = GraphConvolution(nhid, nhid)
        self.gc6 = GraphConvolution(nhid, nhid)
        self.gc7 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc2(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc3(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc4(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc5(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc6(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc7(x, adj)
        return F.log_softmax(x, dim=1)
```



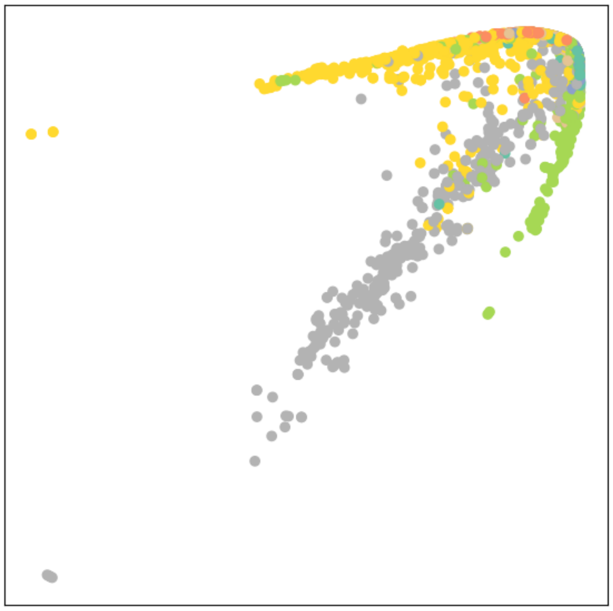
```
Epoch: 0200 loss_train: 0.5537 acc_train: 0.7857 loss_val: 1.2914 acc_val: 0.7067 time: 0.0107s
Optimization Finished!
Total time elapsed: 2.2401s
```

Case 7 with 5 skip connection

```
class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nhid)
        self.gc3 = GraphConvolution(nhid, nhid)
        self.gc4 = GraphConvolution(nhid, nhid)
        self.gc5 = GraphConvolution(nhid, nhid)
        self.gc6 = GraphConvolution(nhid, nhid)
        self.gc7 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc2(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc3(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc4(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc5(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = F.relu(self.gc6(x, adj))+x
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc7(x, adj)
        return F.log_softmax(x, dim=1)
```



```
Epoch: 0200 loss_train: 0.6001 acc_train: 0.7286 loss_val: 1.2995 acc_val: 0.6833 time: 0.0113s
Optimization Finished!
Total time elapsed: 2.2312s
```