**به نام خدا**


دانشگاه صنعتی امیرکبیر

Amirkabir University
of Technology

# پروژه درس تحلیل شبکه های پیچیده

# Graph Neural Network

**استاد درس: دکتر چهرقانی**

**نام: زهرا اخلاقی**

**شماره دانشجویی: ۴۰۱۱۳۱۰۶۴**

**زمستان۱۴۰۲**

# فهرست مطالب

# سوال اول: پیاده‌سازی شبکه‌های عصبی گرافی پایه

## الف)

با استفاده از تابع زیر داده ها به نسبت گفته شده به آموزش، ارزیابی و تست تقسیم شده‌اند. دیتاست گرفته میشود و تعداد گره های آن در N ذخیره میشود و براساس تعداد گره‌ها مقدار تست و ارزیابی و آموزش محاسبه می‌شود و براساس مقدار به دست آمده دوباره بارگیری دستاست انجام می‌شود.

```
def load(name):
   if name == 'citeseer':
       dataset = Planetoid("/tmp/CitationFull", name="CiteSeer")
   elif name == 'CoraFull':
       dataset = Planetoid("/tmp/Cora", name="Cora")

   graph = dataset[0]
   N = graph.num_nodes

   train_ratio = 0.7
   val_ratio = 0.1
   test_ratio = 0.2

   num_train = int(N * train_ratio)
   num_val = int(N * val_ratio)
   num_test = N - (num_val + num_train)

   num_class = dataset.num_classes

   if name == 'citeseer':
                                       dataset         =          Planetoid("/tmp/CitationFull",
name="CiteSeer",split="random",num_train_per_class=         int(num_train/num_class)         ,
num_val=num_val, num_test=num_test)
   elif name == 'CoraFull':
           dataset  =  Planetoid("/tmp/Cora",  name="Cora",  split="random",num_train_per_class=
int(num_train/num_class), num_val=num_val, num_test=num_test)

   num_features =  dataset.num_features
   data = dataset[0]

   return dataset, data, num_class, num_features
```

## ب)

پیاده سازی مدل MLP به صورت زیر میباشد:

```
[ ]  class MLP(torch.nn.Module):
        def __init__(self, input_size, hidden_sizes, output_size):
            super(MLP, self).__init__()
            layers = []
            sizes = [input_size] + hidden_sizes + [output_size]
            for i in range(len(sizes) - 1):
                layers.append(torch.nn.Linear(sizes[i], sizes[i + 1]))
                if i < len(sizes) - 2:
                    layers.append(torch.nn.ReLU())
            self.model = torch.nn.Sequential(*layers)

        def forward(self, x):
            return self.model(x)
```

در تابع زیر مدل mlp اجرا می‌شود و برای آنکه از اطلاعات ساختاری گراف استفاده نشود از data.x برای آموزش مدل استفاده شده است. معماری‌های مختلف mlp با تعداد متفاوت نورون در لایه مخفی که در hidden_size ذکر شده، امتحان شده است ([1024,128],[512,128,32],[1024,32],[1024,512,16],[32 ,1024,256]).

```
def mlp_model(dataset, data, num_class, num_features ):

 input_size = num_features
  hidden_sizes=[[1024,256, 32],[1024,512,16],[1024,32],[512,128,32],[1024,128]]

 output_size = num_class

 best_acc = 0
 best_arch = None

 for hidden_size in hidden_sizes:

    model = MLP(input_size, hidden_size, output_size)
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
    num_epochs = 20
    for epoch in range(num_epochs):
           model.train()
           optimizer.zero_grad()
           out = model(data.x)
           loss = criterion(out[data.train_mask], data.y[data.train_mask])
           loss.backward()
           optimizer.step()

    model.eval()
    with torch.no_grad():
     out = model(data.x)
     _,pred = out.max(dim=1)
     correct = (pred[data.val_mask] == data.y[data.val_mask]).sum().item()
     val_acc = correct / data.val_mask.sum().item()

    if val_acc > best_acc :
       best_acc = val_acc
       model_scripted = torch.jit.script(model)
```

```
        model_scripted.save('best_mlp_model.pt')
        best_arch = hidden_size

model = torch.jit.load('best_mlp_model.pt')
model.eval()
with torch.no_grad():
    out = model(data.x)
    _,pred = out.max(dim=1)
    correct = (pred[data.test_mask] == data.y[data.test_mask]).sum().item()
    test_acc = correct / data.test_mask.sum().item()
```

بهترین دقت ارائه شده و بهترین معماری برای دیتاست citeseer به صورت زیر است:

**Num. nodes: 3327 (train=2204, val=332, test=667)**

**Num Edges: 9104**

**best hidden_size_layers: [1024, 128]**

**Accuracy of best model on the test data: 0.44**

**Accuracy of best model on the val data: 0.43**

بهترین دقت ارائه شده و بهترین معماری برای دیتاست corafull به صورت زیر است:

**Num. nodes: 2708 (train=1747, val=270, test=543)**

**Num Edges: 10556**

**best hidden_size_layers: [1024, 128]**

**Accuracy of best model on the test data: 0.52**

**Accuracy of best model on the val data: 0.59**

برای آموزش مدل MLP از اطلاعات ساختاری نظیر گره‌ها و ارتباط آنها استفاده نشده‌است و تنها از ویژگی های هر گره برای آموزش مدل استفاده می‌شود.

## ج)

مدل GCN دو لایه پیاده سازی شده به صورت زیر میباشد:

```
class GCN2(torch.nn.Module):
   def __init__(self, hidden_size):
        super().__init__()
        self.conv1 = GCNConv(dataset.num_node_features, hidden_size)
        self.conv2 = GCNConv(hidden_size, dataset.num_classes)

   def forward(self, data):
        x, edge_index = data.x, data.edge_index
```

```
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

لایه‌های پنهان [16,8, 32, 64, 128] برای اجرا در نظر گرفته شده اند و بالاترین دقت به صورت زیر است:

**citeseer**:

test: 0.78, on validation data:0.78, depth :2 time: 2.75 hidden size : 8

**CoraFull**:

test: 0.89, on validation data:0.89, depth :2 time: 2.05 hidden size : 64

مدل های GCN یک لایه، سه لایه و چهار لایه نیز برای اجرا انتخاب شده‌اند (لایه‌های پنهان [16,8, 32, 64, 128] بررسی شده‌اند) و نتایج به صورت زیر است:

**citeseer**:

test: 0.76, on validation data:0.77, depth :1 time: 2.28

test: 0.78, on validation data:0.78, depth :2 time: 2.75 hidden size : 8

test: 0.78, on validation data:0.79, depth :3 time: 3.67 hidden size : 16

test: 0.75, on validation data:0.79, depth :4 time: 6.04 hidden size : 64

**CoraFull**:

test: 0.84, on validation data:0.82, depth :1 time: 1.10

test: 0.89, on validation data:0.89, depth :2 time: 2.05 hidden size : 64

test: 0.88, on validation data:0.89, depth :3 time: 2.80 hidden size : 64

test: 0.87, on validation data:0.88, depth :4 time: 2.39 hidden size : 32

برای ارائه نتایج بالا بهترین اندازه برای لایه پنهان براساس داده‌های اعتبار سنجی است و دقت در داده تست و اعتبار سنجی گزارش شده است.

- بهترین دقت برای داده citeseer در مدل سه لایه با ابعاد لایه پنهان ۱۶ میباشد.
- بهترین دقت برای داده corafull در مدل دو لایه با ابعاد لایه پنهان ۶۴ میباشد.

## د)

مدل GAT دو لایه پیاده سازی شده، به صورت زیر می‌باشد:

```
class GAT2(torch.nn.Module):
   def __init__(self,hidden_channels, heads):
       super().__init__()
       self.conv1 = GATConv(dataset.num_node_features, hidden_channels, heads, dropout=0.5)
       self.conv2 = GATConv(hidden_channels * heads, dataset.num_classes, heads)

   def forward(self, data):
       x, edge_index = data.x, data.edge_index
       x = self.conv1(x, edge_index)
       x = F.relu(x)
       x = self.conv2(x, edge_index)
       return F.log_softmax(x, dim=1)
```

تعداد لایه‌های پنهان [16,8] و سرهای توجه [128 ,64 ,32 ,8] برای اجرا بررسی شده‌اند. بهترین تعداد لایه پنهان و سر توجه برای GAT دو لایه به صورت زیر میباشد:

**citeseer**:

test: 0.78,  validation: 0.77, depth:2, time: 5.35, hidden size: 8, head: 5

**CoraFull**:

test: 0.87,  validation: 0.89, depth:2, time: 8.51, hidden size: 16, head: 10

مدل های GAT یک لایه، سه لایه و چهار لایه نیز برای اجرا انتخاب شده‌اند (لایه‌های پنهان [16,8] و سرهای توجه [128 ,64 ,32 ,8] و [1,2,3,5,10] بررسی شده‌اند) و نتایج به صورت زیر است:

**citeseer**:

test: 0.76, on validation data: 0.78, depth:1, time: 2.28, hidden size: 0, head: 1

test: 0.78, on validation data: 0.77, depth:2, time: 5.35, hidden size: 8, head: 5

test: 0.78, on validation data: 0.77, depth:3, time: 7.20, hidden size: 8, head: 5

test: 0.78, on validation data: 0.77, depth:4, time: 5.06, hidden size: 16, head: 1

**CoraFull**:

test 0.85, on validation data: 0.81, depth:1, time: 1.45, hidden size: 0, head: 2

test: 0.87, on validation data: 0.89, depth:2, time: 8.51, hidden size: 16, head: 10

test: 0.89, on validation data: 0.89, depth:3, time: 3.17, hidden size: 32, head: 1

test: 0.87, on validation data: 0.89, depth:4, time: 21.45, hidden size: 64, head: 5

در دیتاست citeseer تعداد لایه ۲ و ۳و ۴ دقت یکسان و بیشترین دقت را دارند.

در دیتاست corafull تعداد لایه ۳ با تعداد لایه پنهان ۳۲ و سر توجه ۱ دارای بیشترین دقت میباشد.

## ه)

مدل GATv2 دو لایه پیاده سازی شده، به صورت زیر می‌باشد:

```
class GATv22(torch.nn.Module):
   def __init__(self,  hidden_channels, num_heads):
     super().__init__()
      self.conv1   =   GATv2Conv(dataset.num_node_features,   hidden_channels,   heads=num_heads,
dropout=0.5)
     self.conv2 = GATv2Conv(hidden_channels * num_heads, dataset.num_classes, heads=num_heads)

   def forward(self, data):
       x, edge_index = data.x, data.edge_index
       x = F.relu(self.conv1(x, edge_index))
       x = self.conv2(x, edge_index)
       return F.log_softmax(x, dim=1)
```

تعداد لایه‌های پنهان [16,8, 32 , 64 , 128] و سرهای توجه [1,2,3,5,10] برای اجرا بررسی شده‌اند. بهترین تعداد لایه پنهان و سر توجه برای GAT دو لایه به صورت زیر میباشد:

**citeseer:**

test: 0.79 , on validation data: 0.82, depth :2, time: 8.56, hidden size: 8, head: 2

**CoraFull:**

test: 0.87 , on validation data: 0.89, depth :2, time: 163.60, hidden size: 128, head: 10

مدل های GATv2 یک لایه، سه لایه و چهار لایه نیز برای اجرا انتخاب شده‌اند (لایه‌های پنهان [16,8, 32 , 64 , 128] و سرهای توجه [1,2,3,5,10] بررسی شده‌اند) و نتایج به صورت زیر است:

**citeseer:**

test: 0.78 , on validation data: 0.78, depth :1, time: 5.67, hidden size: 0, head: 2

test: 0.79 , on validation data: 0.82, depth :2, time: 8.56, hidden size: 8, head: 2

test: 0.79 , on validation data: 0.79, depth :3, time: 8.25, hidden size: 8, head: 3

test: 0.76 , on validation data: 0.79, depth :4, time: 7.29, hidden size: 16, head: 1

**CoraFull:**

test: 0.83 , on validation data: 0.82, depth :1, time: 6.34, hidden size: 0, head: 5

test: 0.87 , on validation data: 0.89, depth :2, time: 163.60, hidden size: 128, head: 10

test: 0.88 , on validation data: 0.88, depth :3, time: 4.74, hidden size: 16, head: 1

test: 0.87 , on validation data: 0.90, depth :4, time: 8.83, hidden size: 16, head: 2

در دیتاست citeseer تعداد لایه ۲ و ۳و ۴ دقت یکسان و بیشترین دقت را دارند.

در دیتاست CoraFull لایه ۳ با تعداد لایه مخفی ۱۶ و اندازه سر توجه ۳ دارای بیشترین دقت میباشد.

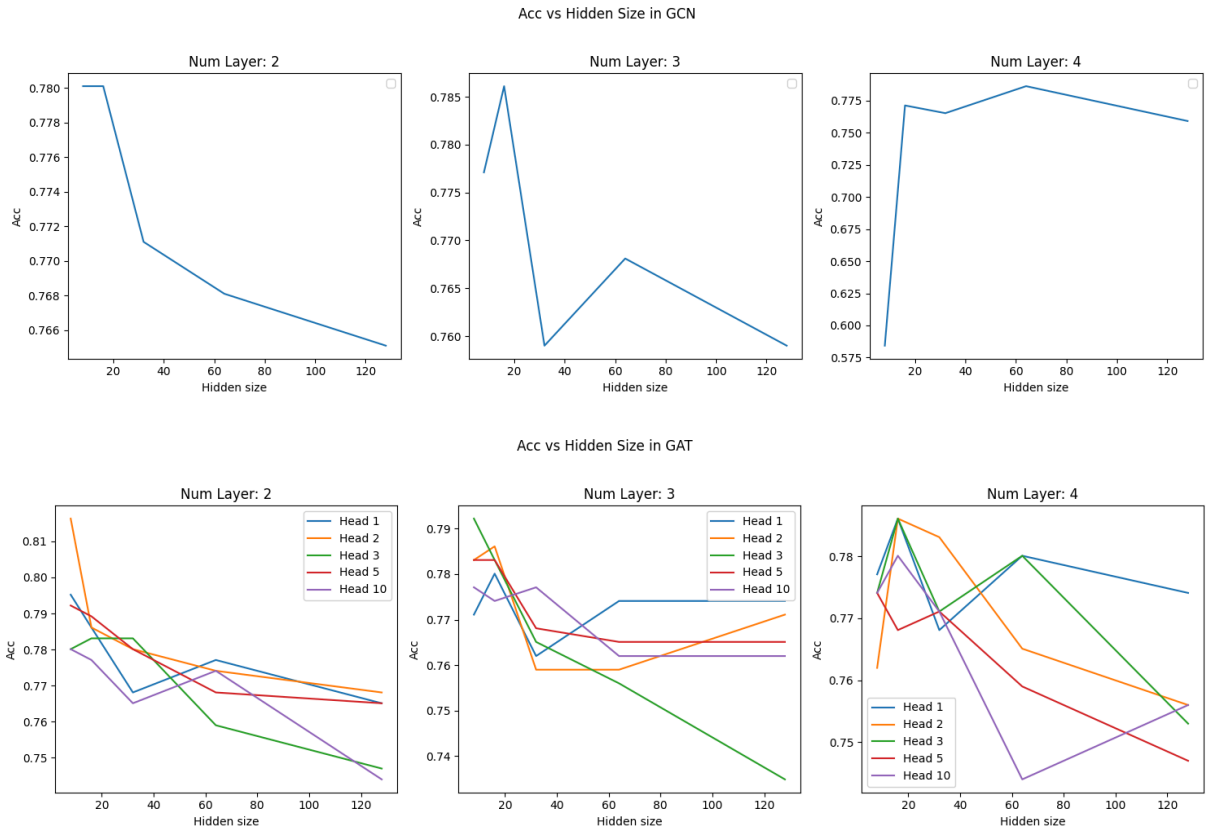
**میان مدلهای اجرا شده برای دیتاست CoraFull بیشترین دقت GAT سه لایه با تعداد لایه پنهان ۳۲ و سر توجه ۱ دارای میباشد و در دیتاست citeseer مدل GATv2 دارای بیشترین دقت میباشد.**

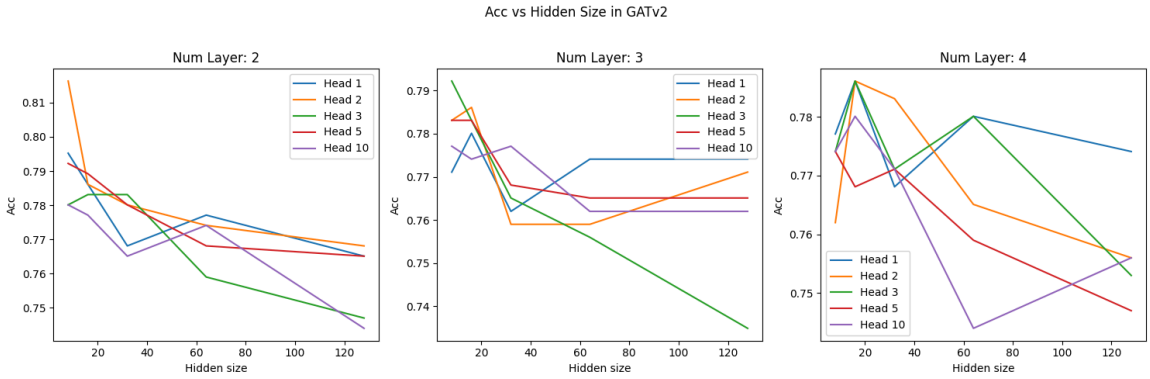
**و)**

**دقت:**

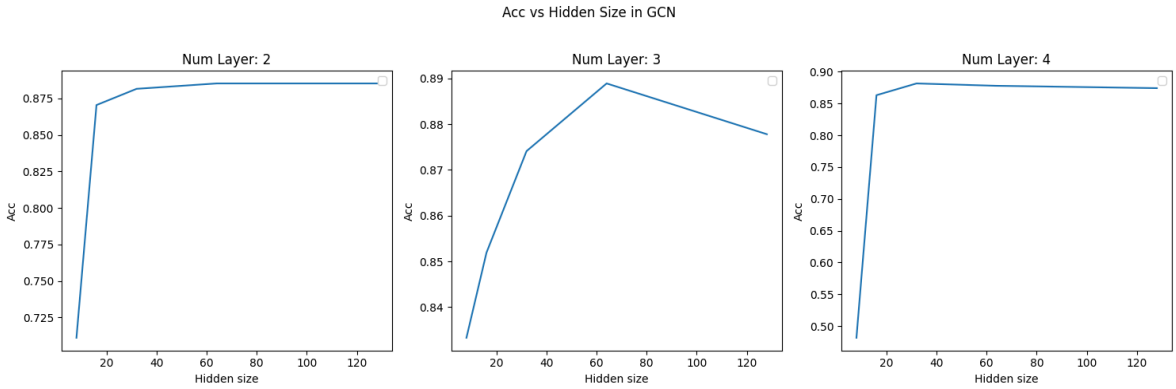
**CiteSeer:**

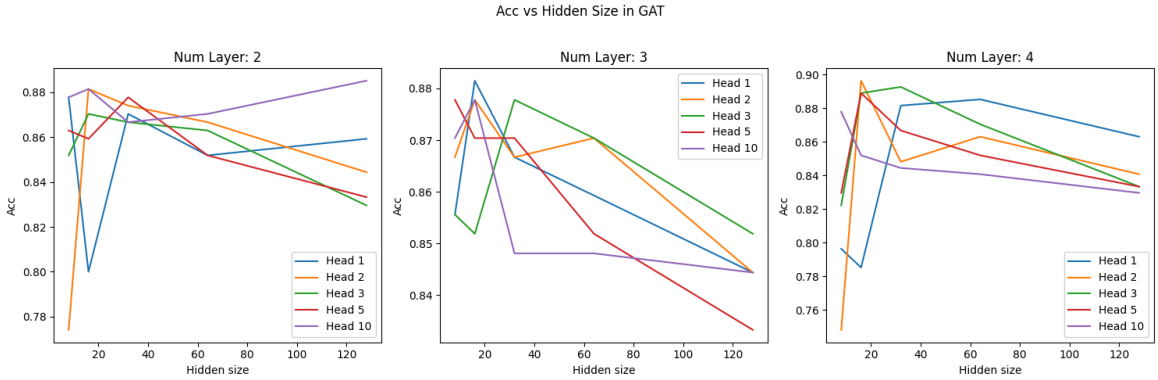
- نمودارهای زیر تاثیر لایه پنهان را بر دقت نشان میدهند:

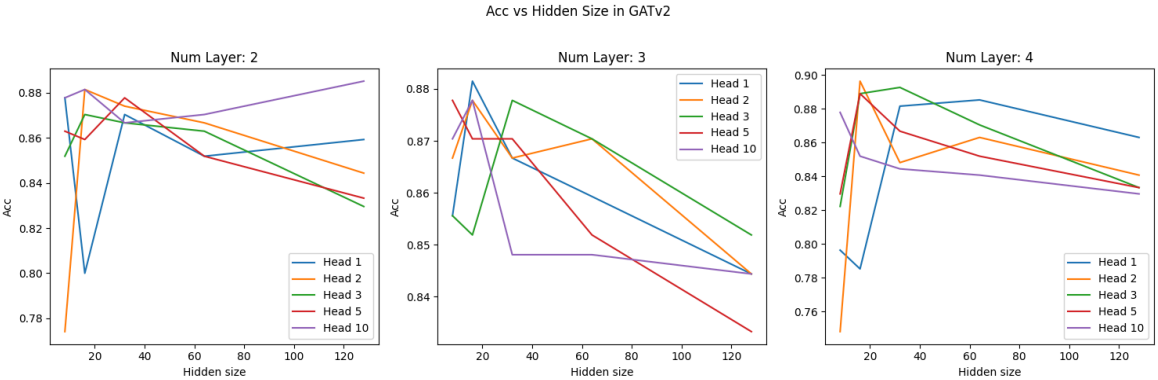



Acc vs Hidden Size in GATv2
Num Layer: 2
Num Layer: 3
Num Layer: 4
Head 1
Head 2
Head 3
Head 5
Head 10
Acc
Hidden size

**:CoraFull**


Acc vs Hidden Size in GCN
Num Layer: 2
Num Layer: 3
Num Layer: 4
Acc
Hidden size

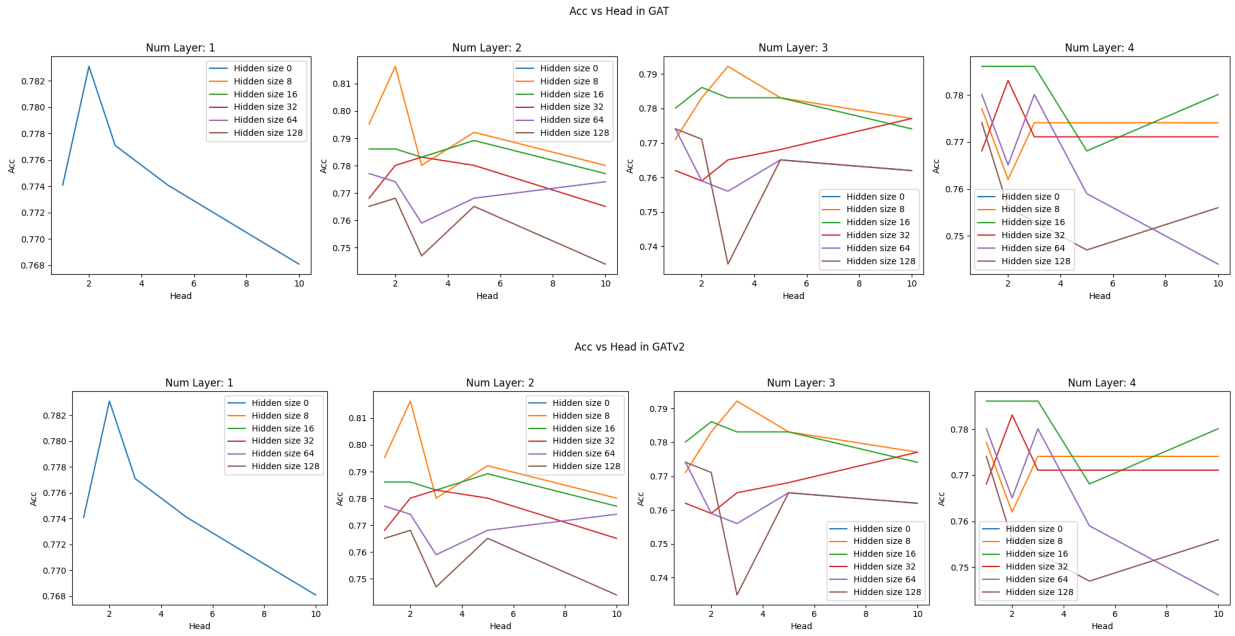

Acc vs Hidden Size in GAT
Num Layer: 2
Num Layer: 3
Num Layer: 4
Head 1
Head 2
Head 3
Head 5
Head 10
Acc
Hidden size


Acc vs Hidden Size in GATv2
Num Layer: 2
Num Layer: 3
Num Layer: 4
Head 1
Head 2
Head 3
Head 5
Head 10
Acc
Hidden size

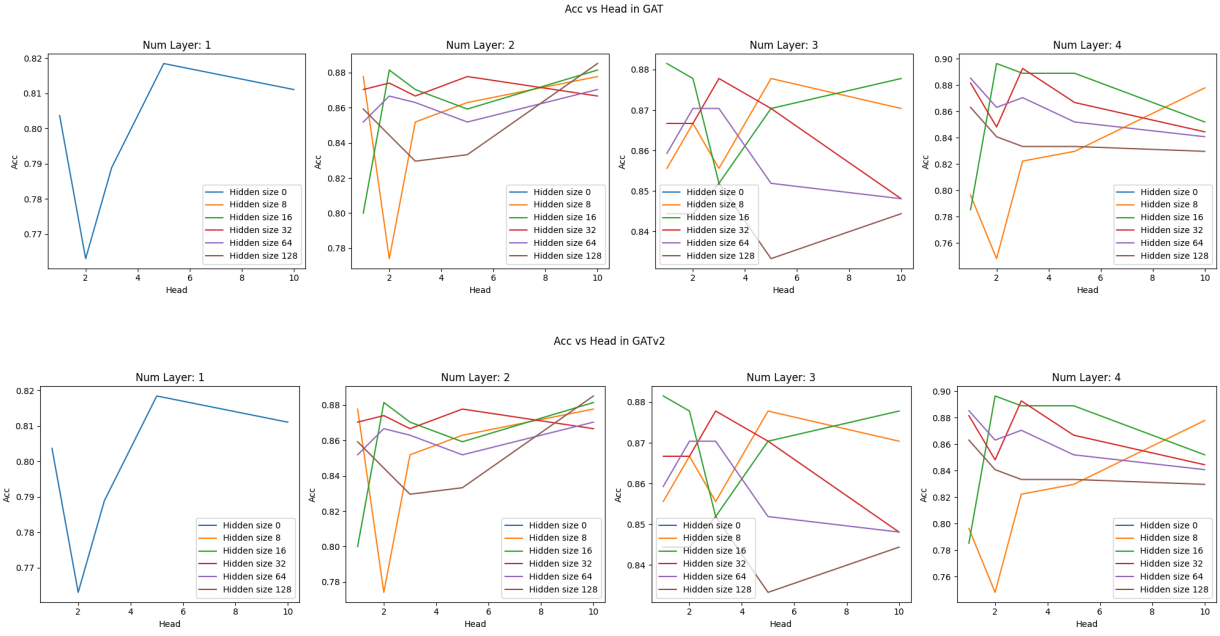
با افزایش تعداد لایه پنهان، دقت مدل افزایش میابد ولی اگر این افزایش از حدی بیشتر باشد منجر به overfit میشود و برای مدلهایی با تعداد لایه کمتر مقدار حد آستانه کوچکتر است، زیرا با مشاهده نمودارهای بالا مدل ها با لایه کمتر با تعداد لایه پنهان کوچکتری مشکل overfit دارند، نسبت به مدلهای با لایه بزرگتر.

- نمودارهای زیر تاثیر سرهای توجه را بر دقت نشان میدهند:

**:CiteSeer**



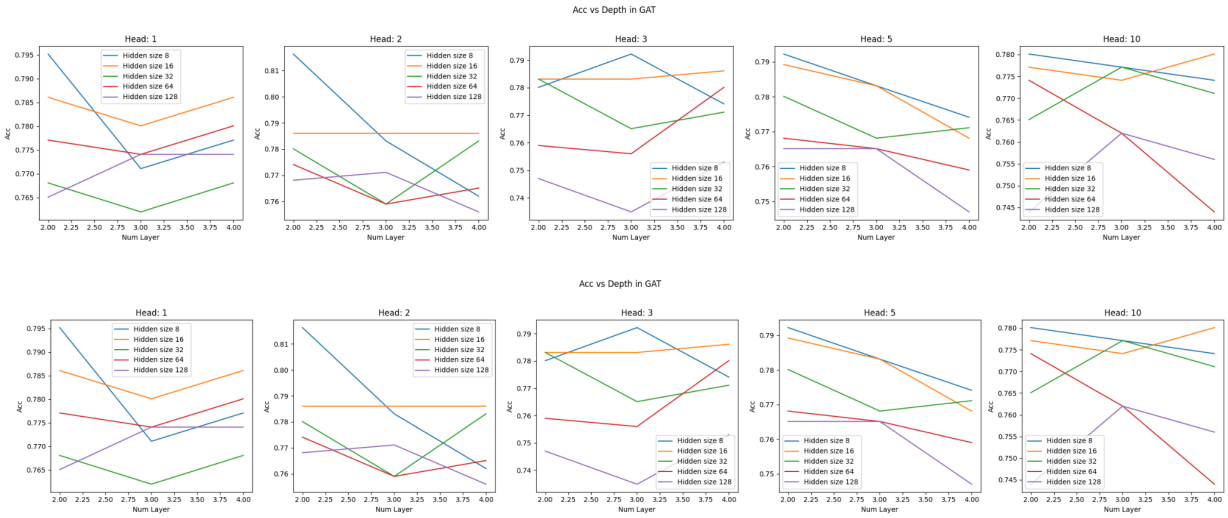

**:CoraFull**



تعداد سرهای توجه می تواند تاثیر قابل توجهی بر دقت داشته باشد. به طور کلی، با افزایش تعداد سرها، مدل قادر به یادگیری وابستگی های پیچیده تر بین گره ها در گراف می شود. با این حال، افزایش بیش از حد تعداد سرها می تواند منجر به overfitting شود، جایی که مدل به جای یادگیری الگوهای کلی، به طور خاص بر روی داده های آموزشی تمرکز می کند. در مدل هایی با تعداد لایه بیشتر هر چه تعداد لایه پنهان بیشتر باشد، با head کمتر باید برای جلوگیری از overfit باشد. در مدل یک لایه برای corafull انازه مناسب برابر ۵ و برای citeseer برابر ۲ می باشد.

- نمودارهای زیر تاثیر عمق شبکه را بر دقت نشان میدهند:

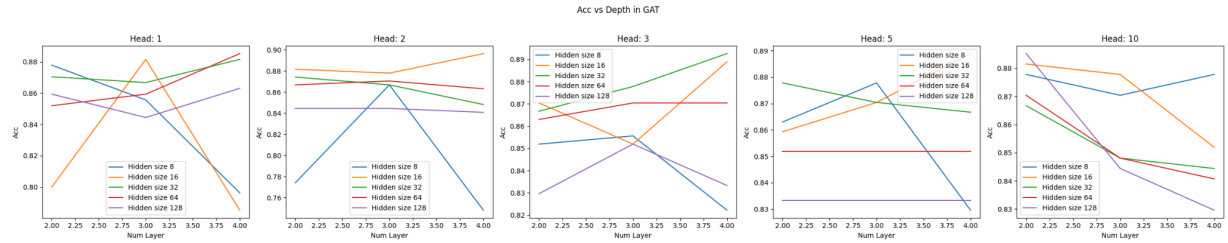
**:CiteSeer**

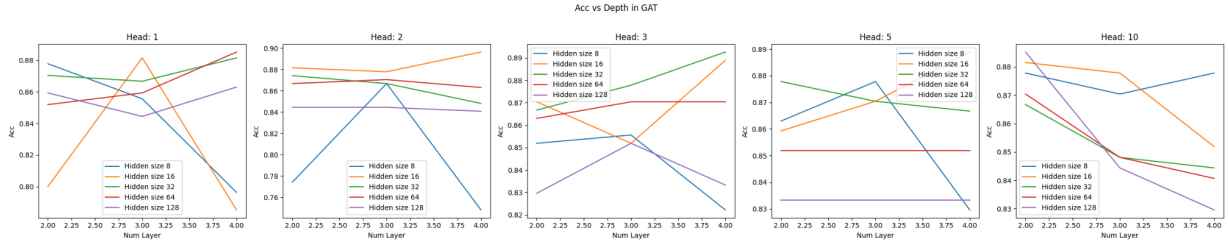






**:CoraFull**



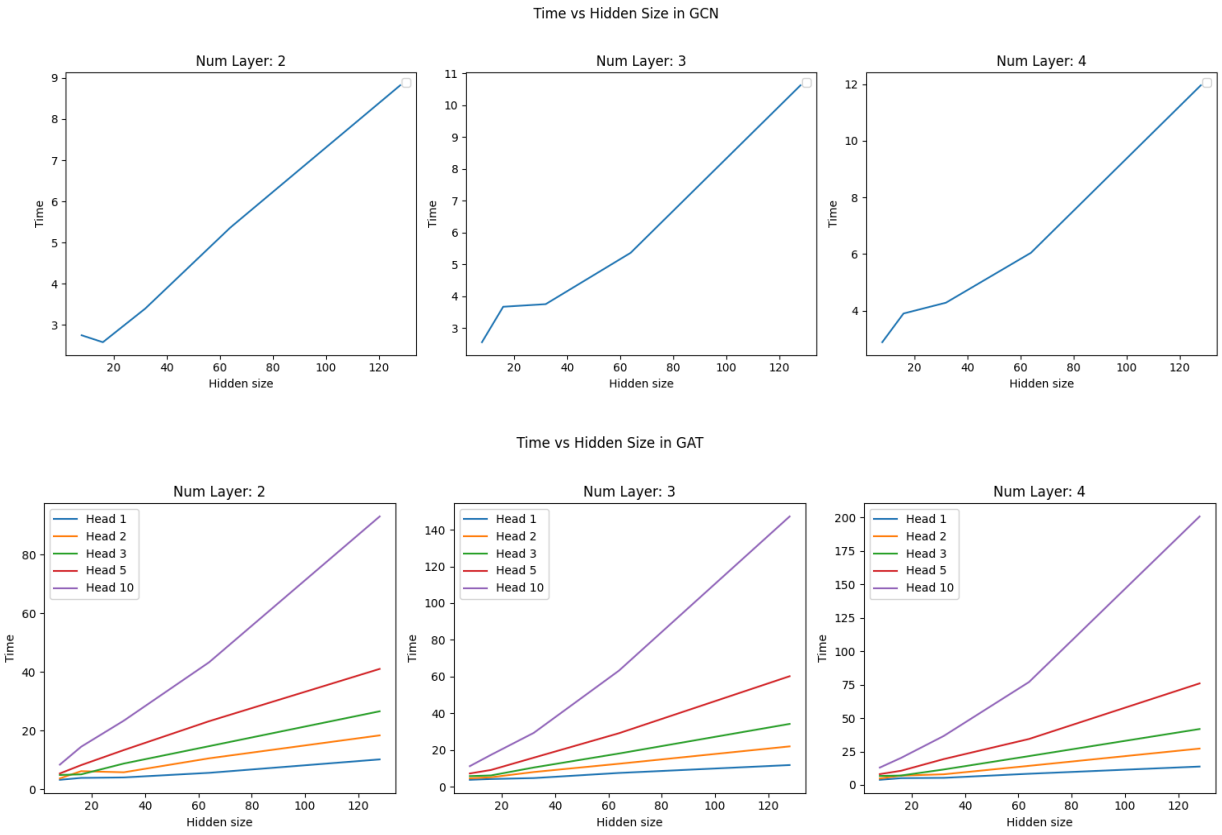
افزایش تعداد لایه‌ها منجر به افزایش دقت مدل می شود، همانطور که در نمودار های بالا مشخص است. با این حال، افزایش بیش از حد تعداد سرها می تواند منجر به oversmoothing شود ولی با توجه به اینکه در مدلهای بالا تنها تا ۴ لایه مدلها پیاده سازی شده اند، over smoothing مشاهده نمیشود.

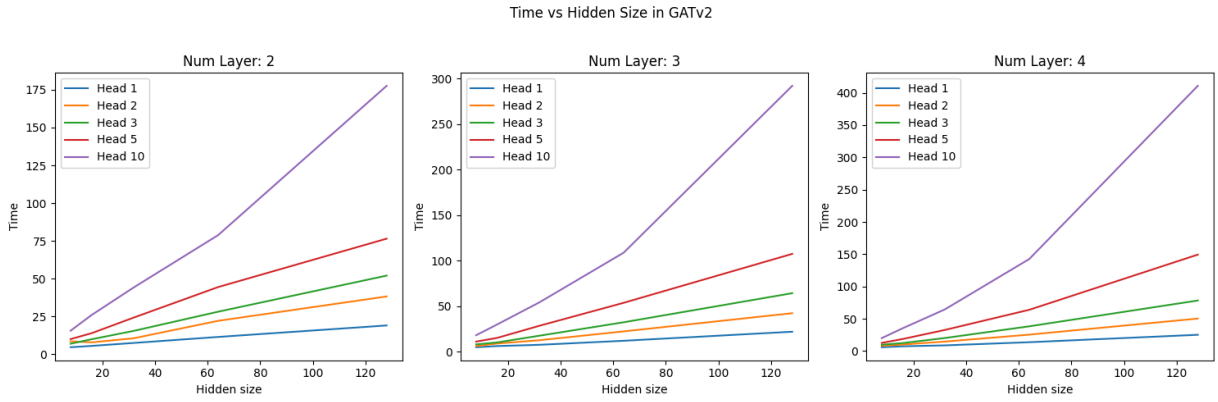
ممکن است در برخی نمونه‌ها با افزایش تعداد لایه، دقت کاهش یابد که به نظرم برای زیاد بودن مقدار head , یا hidden size مدل مشکل over fit دارد.

## زمان:

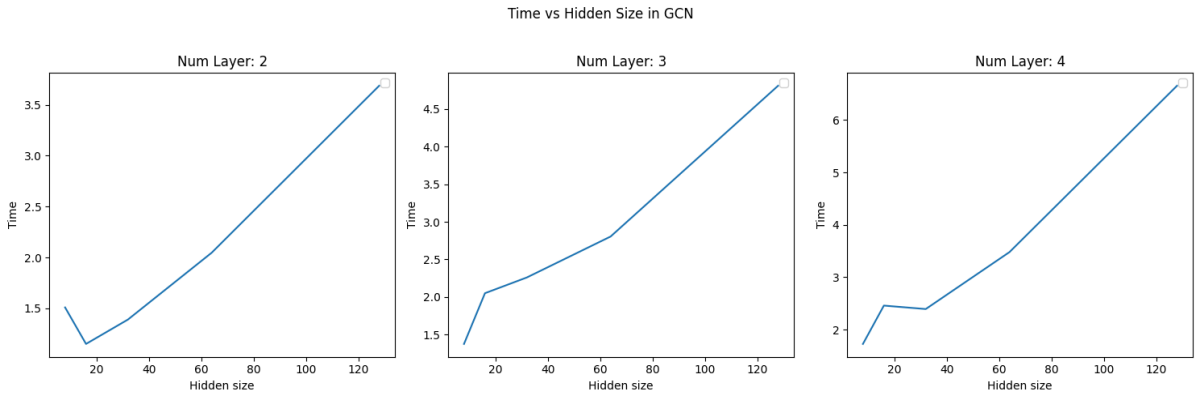
- نمودارهای زیر تاثیر لایه پنهان را بر زمان را نشان میدهند:

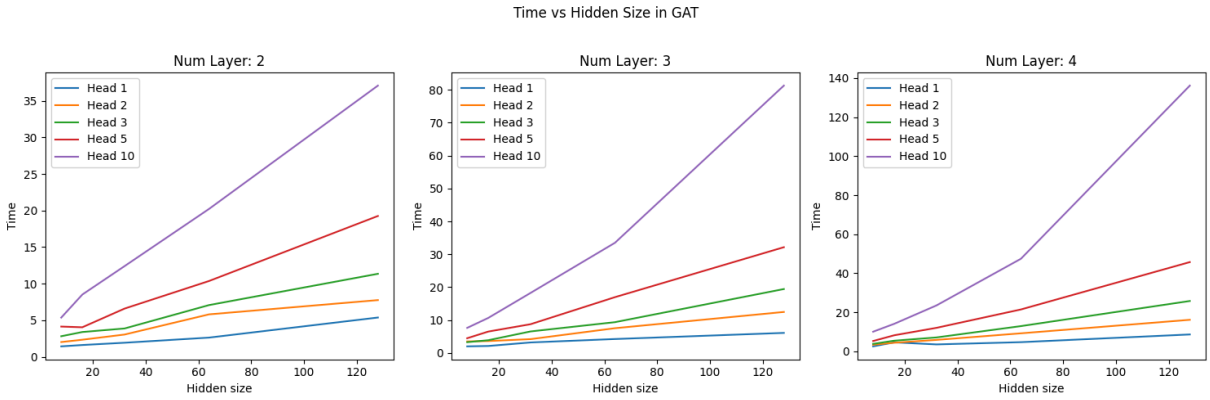
**CiteSeer:**

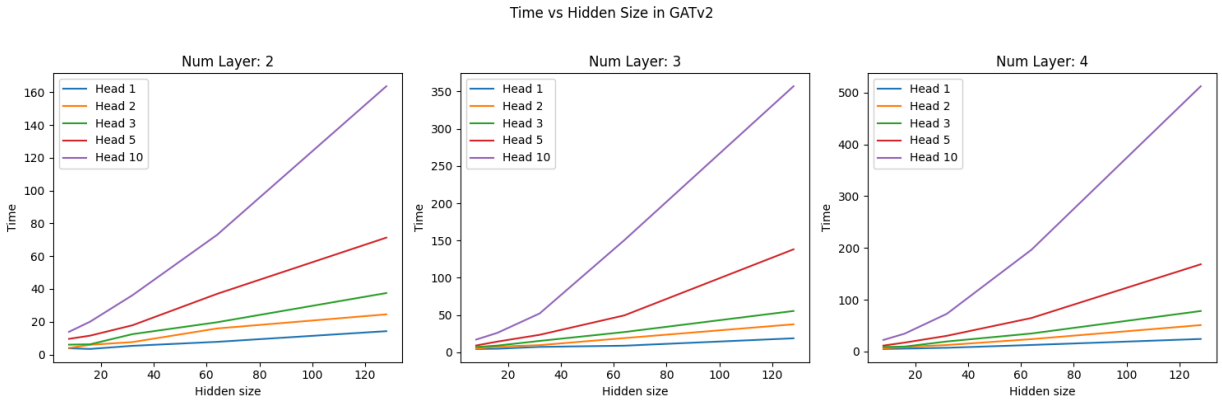



Time vs Hidden Size in GATv2
Num Layer: 2
Num Layer: 3
Num Layer: 4
Head 1
Head 2
Head 3
Head 5
Head 10
Time
Hidden size

**:CoraFull**


Time vs Hidden Size in GCN
Num Layer: 2
Num Layer: 3
Num Layer: 4
Time
Hidden size

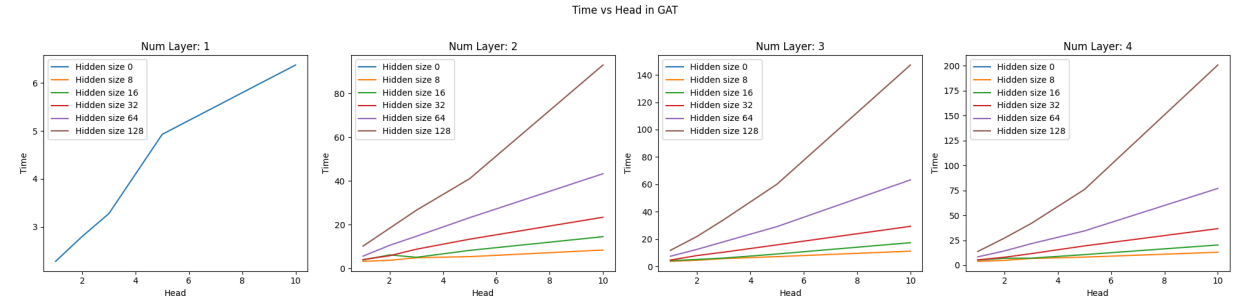

Time vs Hidden Size in GAT
Num Layer: 2
Num Layer: 3
Num Layer: 4
Head 1
Head 2
Head 3
Head 5
Head 10
Time
Hidden size

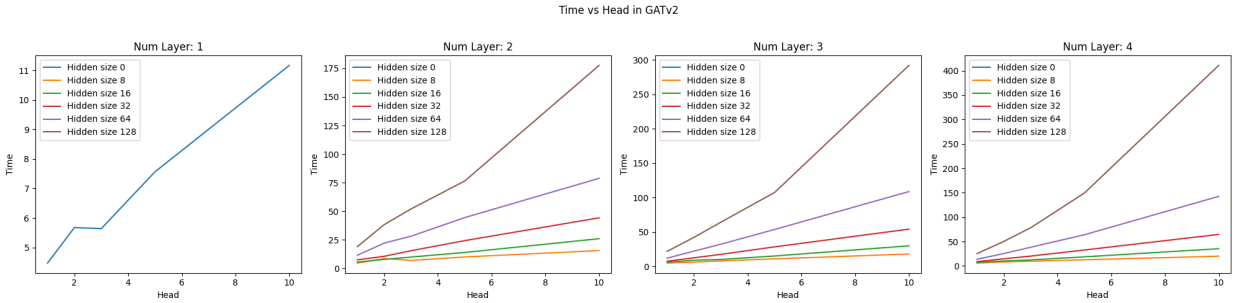



افزایش تعداد لایه پنهان منجر به افزایش زمان آموزش میشود و با افزایش تعداد لایه پنهان تاثیر head بر افزایش زمان آموزش بیشتر میشود.

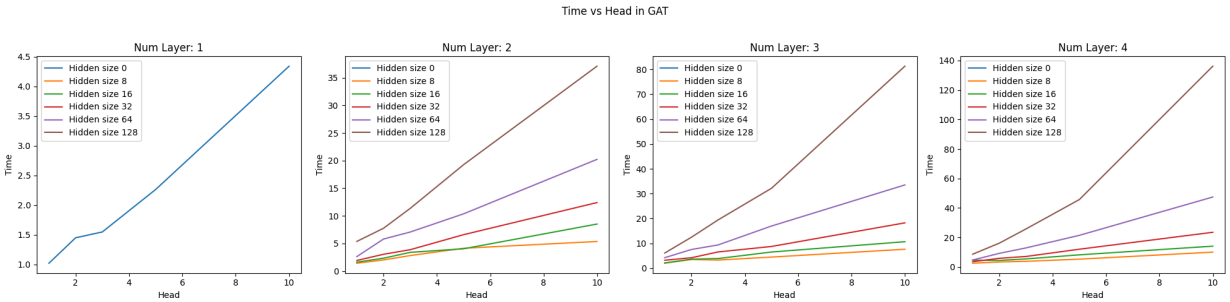
- نمودارهای زیر تاثیر سرهای توجه را بر زمان نشان میدهند:

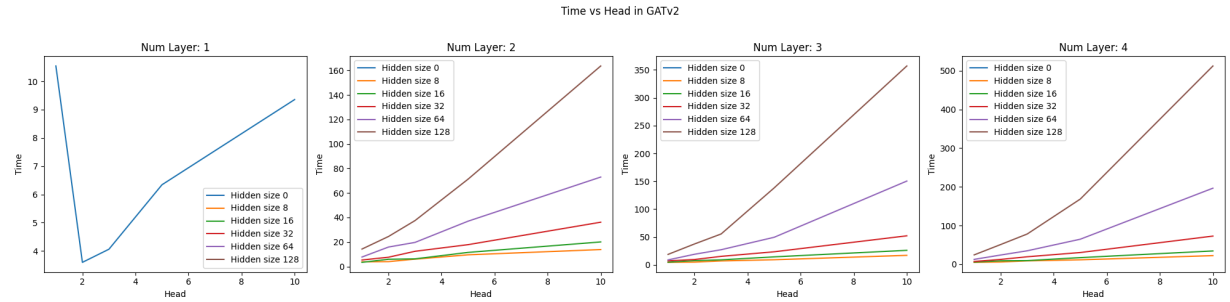
**:CiteSeer**




**:CoraFull**

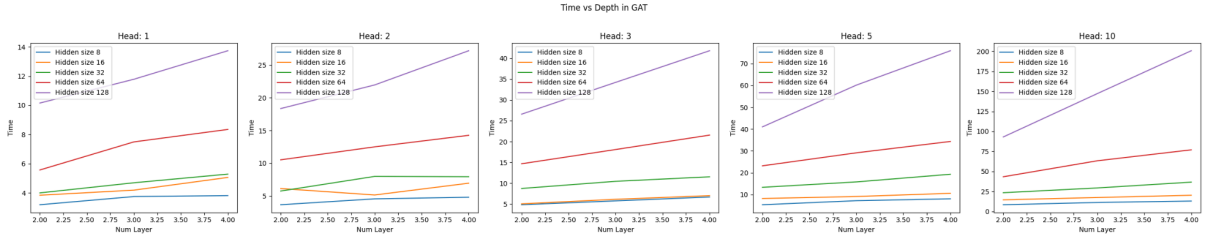


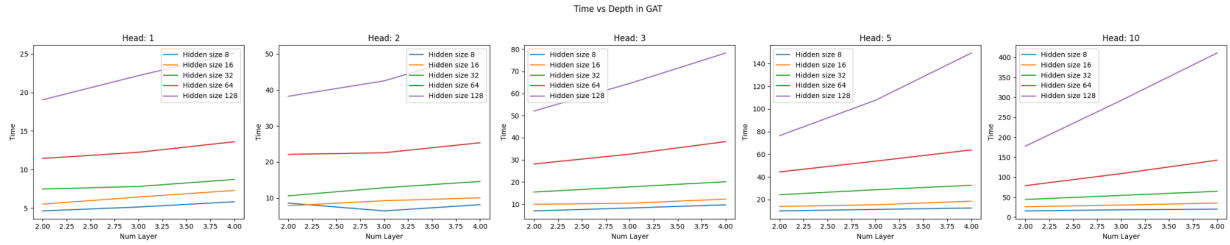




با افزایش تعداد سرهای توجه، سرعت آموزش مدل افزایش میابد.

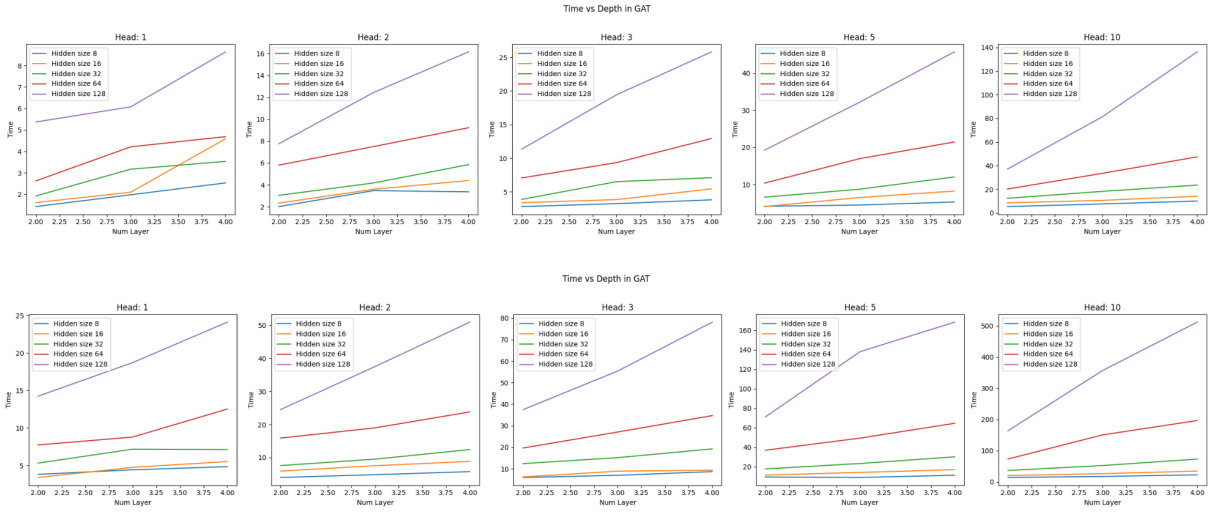
- نمودارهای زیر تاثیر عمق شبکه را بر زمان نشان میدهند:

**:CiteSeer**




**:CoraFull**




با افزایش تعداد لایه، زمان آموزش مدل افزایش میابد.

## ن

**:GCN**

[Time and Space Complexity of Graph Convolutional Networks]

L = تعداد لایه

زمان: O(Lmd + Lnd^2)

حافظه: O(Lnd + Ld^2)

محاسبات را می توان به سه عملیات اصلی تقسیم کرد:

- تبدیل ویژگی :

پیچیدگی زمانی برای تبدیل ویژگی (O(Nd^2)) است زیرا شامل یک ضرب ماتریس متراکم اندازه (N \ بار d) و (d \ بار d) است.

- تجمع همسایگی:

دارای پیچیدگی زمانی (O(|m|d)) می شود.

- تابع فعال سازی:

دارای پیچیدگی زمانی (O(n)) است.

**GAT:**

https://appliednetsci.springeropen.com/articles/10.1007/s41109-021-00420-4

H: تعداد سرهای توجه

پیچیدگی زمانی GAT با لایه‌های L برابر است با $O(lhnd^2 + lh|m|d)$ است، که در آن: عبارت اول، $O(lhnd^2)$، مراحل تبدیل ویژگی را به حساب می آورد. عبارت دوم، $O(lh|m|d)$، مربوط به هزینه مکانیسم توجه عمومی است.

نیازهای حافظه: برای مرحله آموزش GAT، به دلیل نیاز به ذخیره مقادیر توابع توجه، که نیاز به سربار $O(l|m|)$ دارد، نیاز حافظه در ضریب h ضرب می شود. علاوه بر این، نیاز به حافظه شامل فضایی برای ماتریس‌های وزن و حالت‌های پنهان است که به پیچیدگی $O(ld^2 + lnd)$ برای GAT کمک می‌کند.

پیچیدگی محاسباتی این نوع مکانیسم توجه $O(n^2 * d)$ است. برای هر گره، وزن توجه را برای هر یک از همسایگان آن با انجام یک تبدیل خطی بر روی الحاق جاسازی گره و جاسازی همسایه آن محاسبه می کنیم که در مجموع تبدیل خطی $n*(n-1)$ به وجود می آید. هر عملیات تبدیل خطی دارای پیچیدگی محاسباتی $O(d^2)$ است. بنابراین، کل پیچیدگی محاسباتی این نوع مکانیسم توجه $O(n^2* d^2)$ است. این پیچیدگی محاسباتی برای یک سر توجه است. در عمل، GAT ها اغلب از توجه چند سر استفاده می کنند که پیچیدگی محاسباتی آن برابر $O(n^2* d^2*h)$ است.

**GATv2:**

https://ar5iv.labs.arxiv.org/html/2105.14491

پیچیدگی زمانی و استفاده از حافظه GAT و GATv2 کاملاً مشابه است، زیرا GATv2 ویژگی های محاسباتی GAT را به ارث می برد. با این حال، GATv2 توجه پویا را معرفی می‌کند که به انعطاف‌پذیری بیشتری اجازه می‌دهد و به طور بالقوه می‌تواند به تفاوت‌هایی در پیچیدگی زمانی و استفاده از حافظه در شرایط خاص منجر شود.

با مشاهده نمودارهای قسمت قبل نیز متوجه میشویم که gat , gav2 نمودارهای مشابه زمانی دارند و به طور کلی پیچیدگی حافظه و زمان gat, gatv2 مشابه و از gcn بزرگتر است.

# سوال سوم: پیاده‌سازی مقاله دوم

## الف)

این مقاله برای حل مشکل over-fitting, over-smoothing در توسعه شبکه‌های پیچیده گراف ارائه شده است و ایده مقاله DropEdge این است که به طور تصادفی تعداد معینی از یال ها را از گراف ورودی در هر دوره آموزشی حذف می کند که انجام این کار مانند یک تقویت کننده داده (data augmenter) و همچنین یک کاهش دهنده ارسال پیام (message-passing reducer) عمل میکند.

data augmenter: در واقع کپی با شکل‌های متعدد از گراف اصلی تولید میکند که تصادفی بودن و تنوع داده‌های ورودی را افزایش می‌دهیم، بنابراین از over-fitting جلوگیری می‌کند

message-passing reducer: پیام عبوری بین گره های مجاور در امتداد یال‌های گراف هدایت می شود. حذف برخی گره‌ها باعث می‌شود اتصالات گره‌ها پراکنده‌تر شود و از این رو از over-smoothing در زمانی که GCN بسیار عمیق است تا حدی اجتناب می‌شود.

روش Dropout حذف کردن تصادفی برخی از ویژگی های ماتریس ورودی اثر over-fitting را کاهش می‌دهد، اما هیچ کمکی به جلوگیری از over-smoothing نمی کند. DropEdge را می توان به عنوان نسلی از Dropout از کاهش ابعاد ویژگی با استفاده از حذف یالها در نظر گرفت، که over-fittingو over-smoothing را کاهش می دهد. در واقع تاثیرات Dropout و DropEdge مکمل یکدیگر هستند.

## ب)

ایده مقاله DropEdge این است که به طور تصادفی تعداد معینی از یال ها را از گراف ورودی در هر دوره آموزشی حذف می کند، برای پیاده سازی این ایده سه را پیشنهاد شده است:

**گراف تصادفی:**

```python
def random_edge_sampler_random_graph(data, p):
    # Get the list of training nodes
    train_nodes = torch.nonzero(data.train_mask, as_tuple=False).squeeze().numpy()

    # Get the edge indices
    edges = data.edge_index.numpy()

    # Calculate the number of edges to sample
    num_edges = int((len(train_nodes) * (len(train_nodes) - 1) / 2) * (1-p))

    # Sample random edges from the training nodes
    random_edges = np.random.choice(train_nodes, (num_edges, 2), replace=True)

    # Remove self-loops
    random_edges = random_edges[random_edges[:, 0] != random_edges[:, 1]]
```

```
    # Mask out the selected edges from the original edge indices
    mask = np.isin(edges.T, random_edges).all(axis=1)
    result_array = edges.T[~mask].T

    return torch.tensor(result_array)
```

برای پیاده سازی این ایده، گره های درون داده آموزشی استخراج شده است و میان این گره‌ها با استفاده از مقدار احتمالاتی یال ایجاد میشود، که در نهایت خروجی یک گراف تصادفی می باشد، سپس یالهای ایجاد شده از گراف اصلی حذف شده است.

**گراف تصادفی بدون جهت:**

```
def random_edge_sampler_random_graph_undirect(data, p):
    # Get the list of training nodes
    train_nodes = torch.nonzero(data.train_mask, as_tuple=False).squeeze().numpy()

    # Get the edge indices
    edges = data.edge_index.numpy()

    # Calculate the number of edges to sample
    num_edges = int((len(train_nodes) * (len(train_nodes) - 1) / 2) * (1-p))

    # Sample random edges from the training nodes
    random_edges = np.random.choice(train_nodes, (num_edges, 2), replace=True)

    # Remove self-loops
    random_edges = random_edges[random_edges[:, 0] != random_edges[:, 1]]

    # Remove duplicate edges
    random_edges = np.unique(np.sort(random_edges, axis=1), axis=0)
    reversed_array = np.flip(random_edges, axis=1)
    random_edges = np.concatenate((reversed_array, random_edges), axis=0)

    # Mask out the selected edges from the original edge indices
    mask = np.isin(edges.T, random_edges).all(axis=1)
    result_array = edges.T[~mask].T

    return torch.tensor(result_array)
```

تفاوت این روش با روش قبلی این است که با توجه به اینکه گراف ورودی بدون جهت است و برای هر یال (a,b) برعکس آن یعنی (b,a) نیز وجود دارد، عکس هر یال در گراف تصادفی محاسبه میشود و در نهایت از مدل آموزشی حذف می‌شود.

**حذف یال از کل دیتاست:**

```
def random_edge_sampler(data, p):

    num = int((1-p)* (data.num_edges))
    drop_num = np.random.randint(data.num_edges,size=num)
    edges = data.edge_index.numpy().T
    sample_edges = np.delete(edges, drop_num, axis=0)
```

```
    return torch.tensor(sample_edges.T)
```

در این روش از کل دیتاست با توجه به احتمال داده شده، تعدادی یال به صورت تصادفی حذف میشوند (ممکن است یال مربوط به داده آموزش، ارزیابی و اعتبار سنجی باشد) و براساس این داده مدل آموزش میبیند ولی داده اصلی که در آن هیچ یالی حذف نشده است برای تست و اعتبار سنجی استفاده می‌شود.

مدل در نظر گرفته شده GCN دو لایه، به صورت زیر میباشد:

```
class GCN(torch.nn.Module):
    def __init__(self, hidden_size):
        super().__init__()
        self.conv1 = GCNConv(dataset.num_node_features, hidden_size)
        self.conv2 = GCNConv(hidden_size, dataset.num_classes)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index

        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x,inf.dropout, training=self.training)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

برای اجرا و آموزش مدل از تابع gcn_model استفاده می‌شود که دیتاست، نام آن، استفاده از روش sampling و اینکه کدام تابع به عنوان sampling استفاده شود را به عنوان ورودی می‌گیرد.

کلاس EarlyStopper برای پیاده سازی early stopping استفاده میشود.

## ج)

حاصل آموزش و ارزیابی مدل روی دو مجموعه داده corafull , citeseer به صورت زیر می باشد (احتمالات [0.001,0.05,0.2,0.5,0.7,0.8] در هر مورد پیاده سازی شده است و در نهایت بالاترین دقت در داده اعتبار سنجی برای مدل تست استفاده و گزارش شده است):

**Citeseer:**

Num. nodes: 3327 (train=2204, val=332, test=667)

Num Edges: 9104

- **GCN**
    - test: 0.73,on the validation data: 0.77, time: 3.01, hidden size : 8
- **DROPEDGE+ GCN(random_edge func)**
    - test: 0.74,on the validation data: 0.79, time: 2.92, hidden size : 8, probability: 0.05

- **DROPEDGE+ + GCN(random_edge_sampler_random_graph func)**
    - test: 0.77,on the validation data: 0.81, time: 12.35, hidden size : 8, probability: 0.8
- **DROPEDGE+ GCN(random_edge_sampler_random_graph_undirect func)**
    - test: 0.76,on the validation data: 0.80, time: 407.96, hidden size : 8, probability: 0.05

**:CoraFull**

Num. nodes: 2708 (train=1747, val=270, test=543)

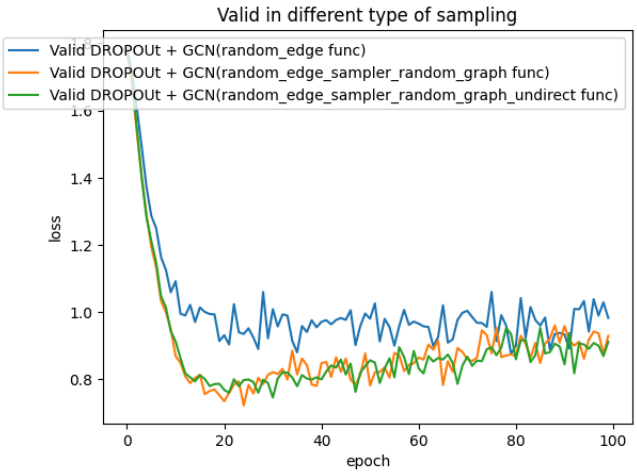
Num Edges: 10556

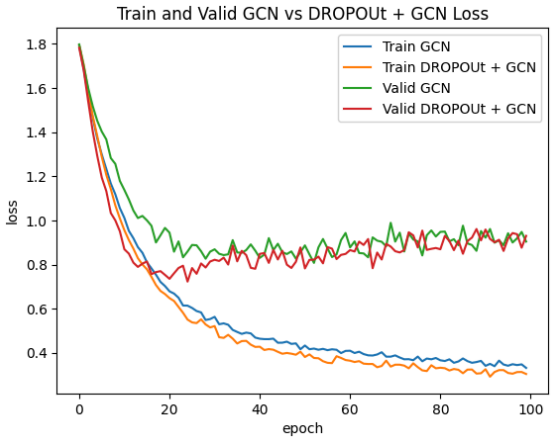
- **GCN**
    - test: 0.86,on the validation data: 0.87, time: 2.71, hidden size : 64
- **DROPEDGE+ GCN(random_edge func)**
    - test : 0.87,on the validation data: 0.87, time: 2.48, hidden size : 64, probability: 0.05
- **DROPEDGE+ GCN(random_edge_sampler_random_graph func)**
    - test: 0.85,on the validation data: 0.87, time: 26.55, hidden size : 64, probability: 0.05
- **DROPEDGE+ GCN(random_edge_sampler_random_graph_undirect func)**
    - test: 0.85,on the validation data: 0.87, time: 203.69, hidden size : 64, probability: 0.2

**د)**

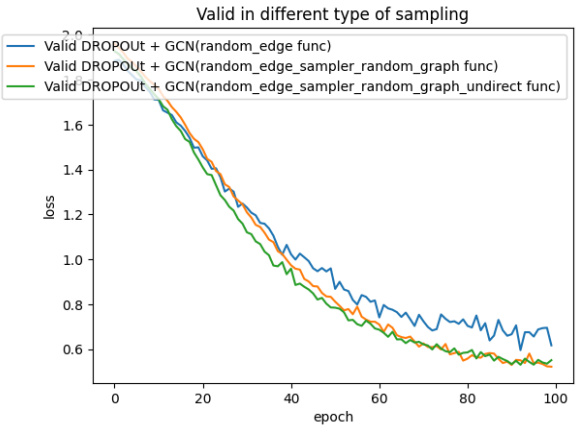
در شکل ها به اشتباه DROPOUT نوشته شده است و صحیح آن DROPEDGE است.

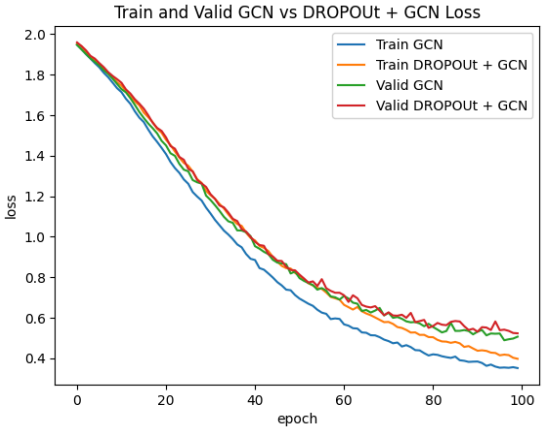
**CiteSeer:**




**CoraFull**:




با مشاهده نتایج حال از پیاده سازی مدل با استفاده از DROPEDGE و بدون استفاده از آن، متوجه میشویم که نتایج بسیار نزدیک به یکدیگرند و حتی در دیتا ست CoraFull نتایج دقت تنها درصدی با یکدیگر تفاوت دارند و از میان توابع متفاوت پیاده سازی شده برای انتخاب تصادفی گره ها در مرحله آموزش random_edge func بهترین دقت را دارد.

روش DROPEDGE برای حل مشکل overfit, over sampeling می باشد و با توجه به اینکه مدل پیاده سازی شده تنها دارای دو لایه میباشد، مشکل oversampeling را ندارد و تعداد لایه پنهان برابر با بهترین دقتی که در قسمت قبل گزارش شده قرار گرفته است، مشکل overfit را ندارد، با این وجود استفاده از روش DROPEDGE نه تنها دقت مدل را کاهش نداده بلکه باعث بهبود در حد کوچکی شده است.

## ه)

**نتایج گزارش شده در مقاله:**

| Dataset | Backbone | 2 layers | |
|---|---|---|---|
| | | Orignal | DropEdge |
| Cora | GCN | 86.10 | **86.50** |
| | ResGCN | - | - |
| | JKNet | - | - |
| | IncepGCN | - | - |
| | GraphSAGE | 87.80 | **88.10** |
| Citeseer | GCN | 75.90 | **78.70** |
| | ResGCN | - | - |
| | JKNet | - | - |
| | IncepGCN | - | - |
| | GraphSAGE | 78.40 | **80.00** |

**نتایج پیاده سازی شده:**

Cora: Original: 0.86 DROPEDGE : 0.87

CiteSeer: Original: 0.73 DROPEDGE : 0.77

نتایج پیاده‌سازی عینا با نتایج مقاله مطابقت ندارد ولی نزدیک به یکدیگر می باشند در مقاله GCN با توابع نرمالسازی مختلف پیاده سازی شده است و همچنین مدل دارای پارامترهای lr, drop_out,weight_decay میباشد که به طور دقیق برای این مدلها در مقاله ذکر نشده است. دقت در مدل پیاده سازی شده با استفاده از DROPEDGE و مقاله در مدل دو لایه کاهش پیدا نکرده است و تنها گاهی اوقات چند درصد افزایش دارد.

## و)

یکی از مشکلاتی که با افزایش عمق در شبکه‌های GCN به وجود میاید over-smoothing است. با توجه به پدیده small world (فاصله‌ی گره ها در گراف‌های دنیای واقعی حداکثر ۶ است)با افزایش عمق در شبکه‌های عصبی گرافی، اشتراک میان گره‌ها افزایش میابد، به طوری که، اگر تعداد لایه‌ها از حد معینی بیشتر شود به جای اینکه دقت و عملکرد شبکه افزایش یابد کاهش میابد زیر تمام نمایش‌های گره‌ها به یک نقطه ثابت همگرا می‌شوند و ممکن است بسیاری از گره‌ها با یکدیگر متفاوت باشند ولی افزایش عمق باعث یکسان شدن تعداد گره‌های اشتراکی برای آنها شده و در نهایت ارائه یکسانی پیدا می‌کنند و علت اینکه با افزایش عمق دقت در مدل GCN کاهش میابد پدیده over-smoothing میباشد.

چند روش علمی برای تشخیص این پدیده:

مشاهده تغییرات در عملکرد:

اگر با افزایش تعداد لایه‌ها، عملکرد شبکه افزایش نیابد یا حتی کاهش یابد، ممکن است باشد که over-smoothing رخ داده باشد.

تحلیل میزان اطلاعات:

اگر متوجه شویم که با افزایش تعداد لایه‌ها، میزان اطلاعات موجود در نمایش‌های میانی کاهش می‌یابد، این نشان دهنده‌ی احتمال over-smoothing است.

استفاده از روش‌های اندازه‌گیری اطلاعات مجاورت:

استفاده از معیارهایی مانند اندازه‌گیری میانگین فاصله (average distance)، اندازه‌گیری اطلاعات مشترک (mutual information) و یا اندازه‌گیری اطلاعات کاهش یافته (diminished information) است.

DropEdge به عنوان کاهش دهنده ارسال پیام در نظر گرفته می‌شود. در GCN ها، پیام عبوری بین گره های مجاور در امتداد یالها هدایت می شود و حذف برخی لبه‌ها باعث می‌شود اتصالات گره‌ها پراکنده‌تر شود و از این رو از over-smoothing در زمانی که GCN بسیار عمیق است تا حدی اجتناب می‌شود.

## ن

مدل لایه پیاده سازی شده، به صورت زیر میباشد:

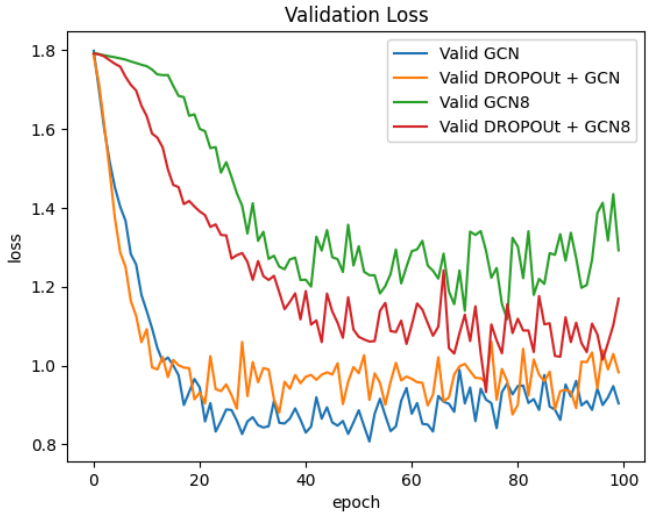
```
class GCN8(torch.nn.Module):
   def __init__(self, hidden_size):
       super().__init__()
       self.conv1 = GCNConv(dataset.num_node_features, hidden_size)
       self.conv2 = GCNConv(hidden_size, hidden_size)
       self.conv3 = GCNConv(hidden_size, hidden_size)
       self.conv4 = GCNConv(hidden_size, hidden_size)
       self.conv5 = GCNConv(hidden_size, hidden_size)
       self.conv6 = GCNConv(hidden_size, hidden_size)
       self.conv7 = GCNConv(hidden_size, hidden_size)
       self.conv8 = GCNConv(hidden_size, dataset.num_classes)

   def forward(self, data):
       x, edge_index = data.x, data.edge_index

       x = F.relu(self.conv1(x, edge_index))
       x = F.dropout(x,inf.dropout, training=self.training)
       x = F.relu(self.conv2(x, edge_index))
       x = F.dropout(x,inf.dropout, training=self.training)
       x = F.relu(self.conv3(x, edge_index))
       x = F.dropout(x,inf.dropout, training=self.training)
       x = F.relu(self.conv4(x, edge_index))
       x = F.dropout(x,inf.dropout, training=self.training)
       x = F.relu(self.conv5(x, edge_index))
       x = F.dropout(x,inf.dropout, training=self.training)
       x = F.relu(self.conv6(x, edge_index))
       x = F.dropout(x,inf.dropout, training=self.training)
```

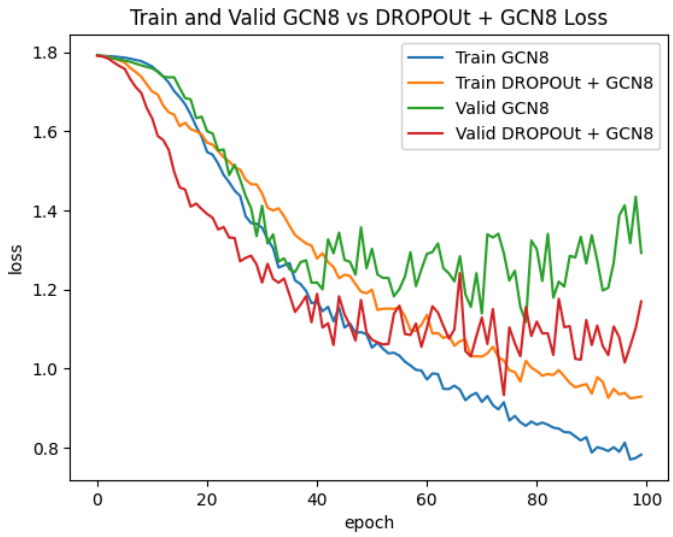
```
        x = F.relu(self.conv7(x, edge_index))
        x = F.dropout(x,inf.dropout, training=self.training)
        x = self.conv8(x, edge_index)
        return F.log_softmax(x, dim=1)
```

پیاده سازی مدل ۸ لایه با استفاده از DROPEDGE در تابع gcn8_model میباشدکه دیتاست، نام آن، استفاده و یا عدم استفاده از DROPEDGE, نوع تابع پیاده سازی را به عنوان ورودی میگیرد. امکان استفاده از dropout , early stopping در این تابع وجود دارد و مدل برای احتمال انتخاب یال [0.001,0.05,0.2,0.5,0.7,0.8] تست شده و بهترین به عنوان خروجی گزارش میشود. خروجی این تابع مقدار loss در داده آموزش و اعتبار سنجی در هر ایپاک میباشد.

**CiteSeer:**

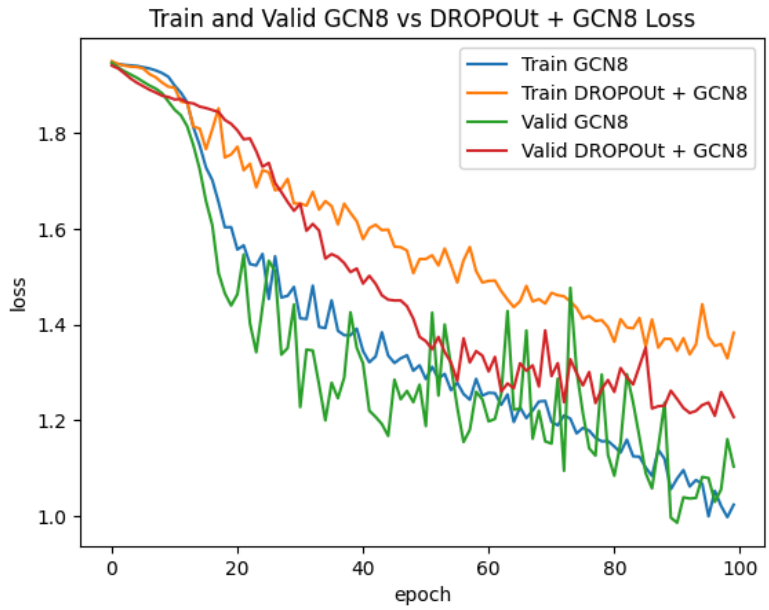
- **GCN8**
  - test: 0.73,on the validation data: 0.71, time: 6.07, hidden size : 16
- **DROPEDGE+ GCN8(random_edge func)**
  - test: 0.80,on the validation data: 0.77, time: 5.02, hidden size : 16, probability: 0.8
- **DROPEDGE+ GCN8(random_edge_sampler_random_graph func):**
  - test: 0.78,on the validation data: 0.76, time: 14.22, hidden size : 16, probability: 0.8

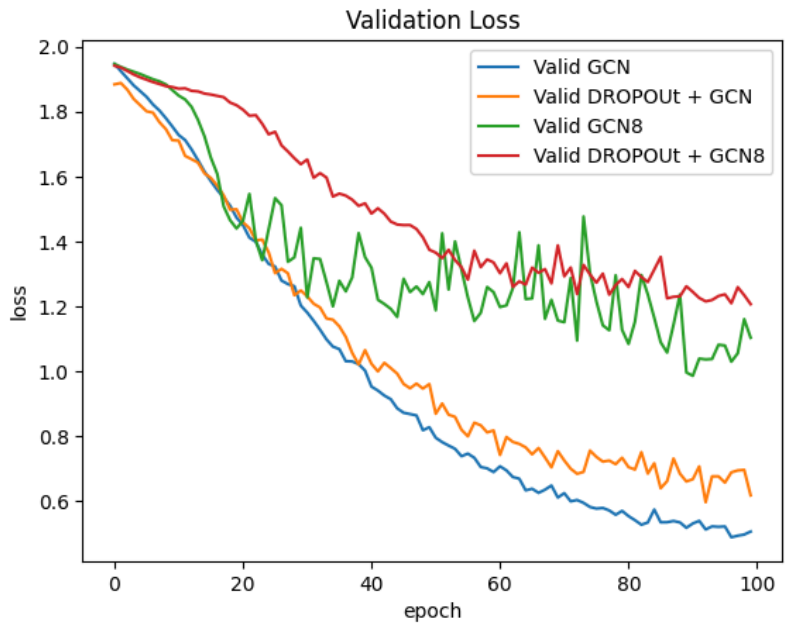





**CoraFull**:

- **GCN8**
  - test: 0.63,on the validation data: 0.64, time: 3.20, hidden size : 16
- **DROPEDGE+ GCN8(random_edge func)**
  - test: 0.73,on the validation data: 0.74, time: 2.89, hidden size : 16, probability: 0.5
- **DROPEDGE+ GCN8(random_edge_sampler_random_graph func)**:
  - test: 0.75,on the validation data: 0.71, time: 19.02, hidden size : 16, probability: 0.5






**نتایج مقاله:**

Cora: Original: 0.78 DROPEDGE : 0.85

CiteSeer: Original: 0.74 DROPEDGE : 0.77

**نتایج پیاده سازی شده:**

Cora: Original: 0.63 DROPEDGE : 0.75

CiteSeer: Original: 0.73 DROPEDGE : 0.80

با مشاهده نتایج فوق، متوجه میشویم با افزایش تعداد لایه از ۲ به ۸ دقت علیرغم اضافه شدن لایه و ویژگی های جدید کاهش یافته است که به دلیل پدیده over smoothing میباشد.

با اضافه کردن drop edge در هر ایپاک از آموزش مدل ۸ لایه دقت در مدل افزایش میابد و استفاده از drop edge برای جلوگیری از پدیده over smoothing , over fit موثر است.

در دیتاست cora، در مقاله و پیاده سازی افزایش دقت در حدود ۱۰ درصد نسبت به مدل بدون استفاده از drop edge وجود دارد و در دیتاست citeseer بهبود دقت در مقاله ۳ درد و در پیاده سازی ۷ درصد میباشد.

## ج)

پیاده سازی مدل skip connection به صورت زیر میباشد:

```
class GCN8Skip(nn.Module):
   def __init__(self, in_channels, hidden_dim, out_channels, num_layers):
       super(GCN8Skip, self).__init__()

       self.num_layers = num_layers

       self.num_layers = num_layers
       self.conv1 = GCNConv(in_channels, hidden_dim)
        self.convs = nn.ModuleList([GCNConv(hidden_dim, hidden_dim) for _ in range(num_layers -
2)])
       self.conv2 = GCNConv(hidden_dim, out_channels)
           self.skip_connections = nn.ModuleList([nn.Linear(hidden_dim, hidden_dim) for _ in
range(num_layers - 1)])


   def forward(self, data):
       x, edge_index = data.x, data.edge_index

       x = F.relu(self.conv1(x, edge_index))
       x = F.dropout(x,inf.dropout, training=self.training)
       skip_connections = [x]
       for i, conv in enumerate(self.convs):
           x = F.relu(conv(x, edge_index) + self.skip_connections[i](skip_connections[-1]))
           x = F.dropout(x,inf.dropout, training=self.training)
           skip_connections.append(x)
       x = self.conv2(x, edge_index)
       return F.log_softmax(x, dim=1)
```

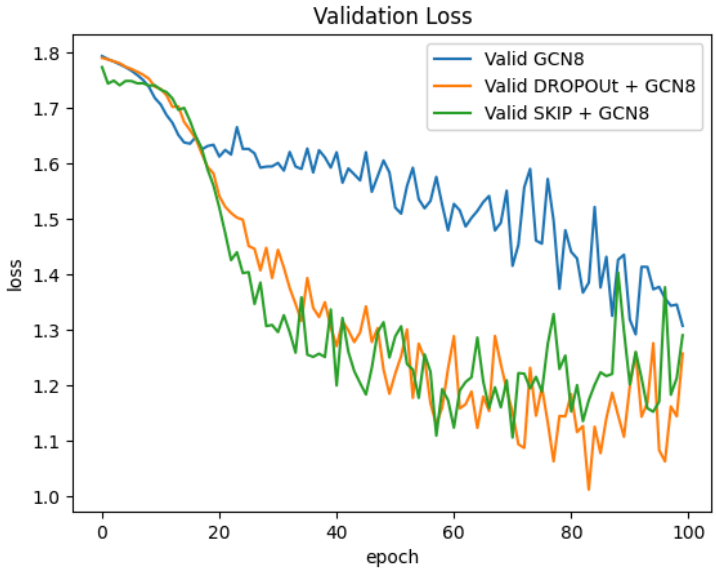
پیاده سازی مدل ۸ skip connection لایه با استفاده از DROPEDGE در تابعgcn8Sskip_model میباشدکه دیتاست، نام آن، استفاده و یا عدم استفاده از DROPEDGE را به عنوان ورودی میگیرد. امکان استفاده از , dropout early stopping در این تابع وجود دارد و مدل برای احتمال انتخاب یال [0.001,0.05,0.2,0.5,0.7,0.8] تست شده و بهترین به عنوان خروجی گزارش میشود. خروجی این تابع مقدار loss در داده آموزش و اعتبار سنجی در هر ایپاک میباشد.

**Citeseer:**

- **GCN8**
    - test : 0.73,on the validation data: 0.71, time: 8.64, hidden size : 16
- **DROPEDGE + GCN8(random_edge func)**

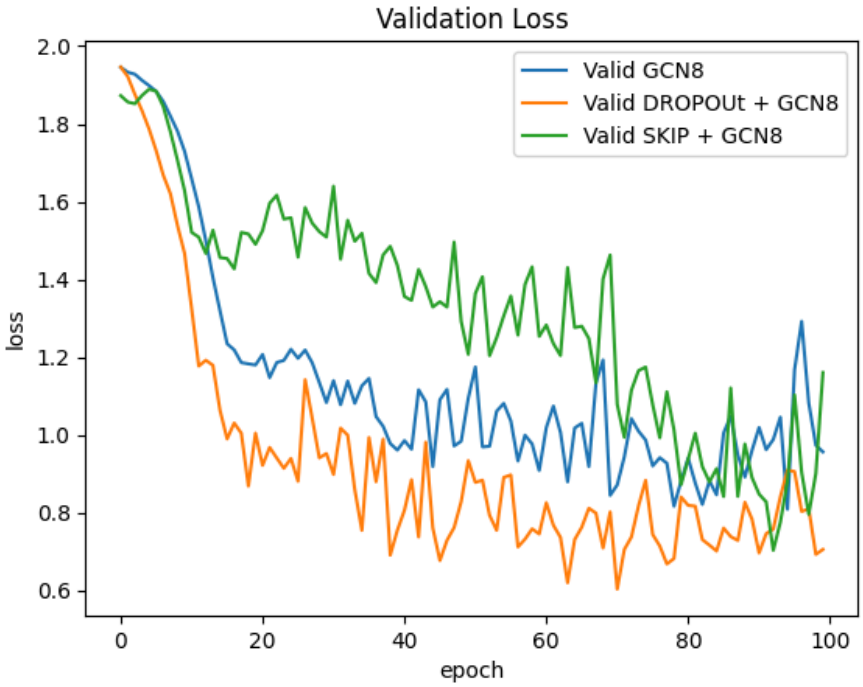
- test:0.78,on the validation data: 0.78, time: 6.73, hidden size : 16, probability: 0.8

- **GCNSKIP**
  - test: 0.76,on the validation data: 0.74, time: 7.38, hidden size : 16




**CoraFull**:

- **GCN8**
  - test: 0.75,on the validation data: 0.74, time: 6.10, hidden size : 16
- **DROPEDGE + GCN8(random_edge func)**
  - test: 0.84,on the validation data: 0.84, time: 5.25, hidden size : 16, probability: 0.8
- **GCNSKIP**
  - test: 0.84,on the validation data: 0.84, time: 4.36, hidden size : 16




- 

با توجه به نتایج ارائه شده، در دیتاست CoraFull استفاده از drop edge, skip connection دارای خروجی یکسانی میباشد و در دیتاست citeseer روش drop edge خروجی بهتری دارد.

Skip connection با ایجاد اتالات مستقیم در مدل از مشکل ناپدید شدن گرادیان جلوگیری میکند و برای حل مشکل over smoothing موثر است ولی نمیتواند برای over fit را بهبود دهد و با توجه به اینکه روش drop edge برای هر دو مشکل overfit , over smoothing میباشد استفاده از آن منطقی‌تر است.

## ط)

در این بخش ترکیب skip connection , gat, gatv2 با drop edge پیاده سازی شده است:

**SKIP Connection:**

**Citeseer:**

- **GCN8**
  - test : 0.73,on the validation data: 0.71, time: 8.64, hidden size : 16
- **DROPEDGE + GCN8(random_edge func)**

- test:0.78,on the validation data: 0.78, time: 6.73, hidden size : 16, probability: 0.8

- **GCNSKIP**
  - test: 0.76,on the validation data: 0.74, time: 7.38, hidden size : 16
- **DROPEDGE + GCNSKIP8(random_edge func)**
  - test: 0.77,on the validation data: 0.81, time: 6.54, hidden size : 16, probability: 0.8

**CoraFull**:

- **GCN8**
  - test: 0.75,on the validation data: 0.74, time: 6.10, hidden size : 16
- **DROPEDGE + GCN8(random_edge func)**
  - test: 0.84,on the validation data: 0.84, time: 5.25, hidden size : 16, probability: 0.8
- **GCNSKIP**
  - test: 0.84,on the validation data: 0.84, time: 4.36, hidden size : 16

**DROPEDGE + GCNSKIP8(random_edge func)**

- test: 0.86,on the validation data: 0.87, time: 5.76, hidden size : 16, probability: 0.8

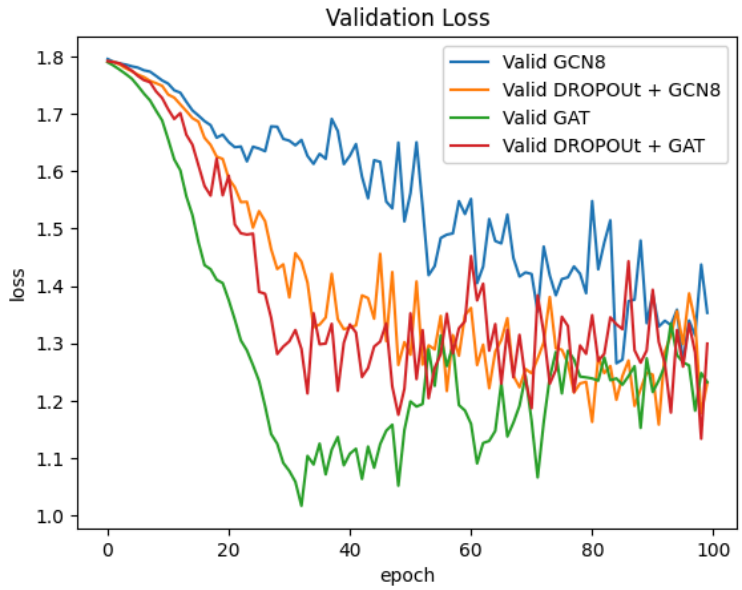
در مدل CoraFull ترکیب SKIP , DROP EDGE دقت بهتری دارد.

**GAT:**

**Citeseer**:

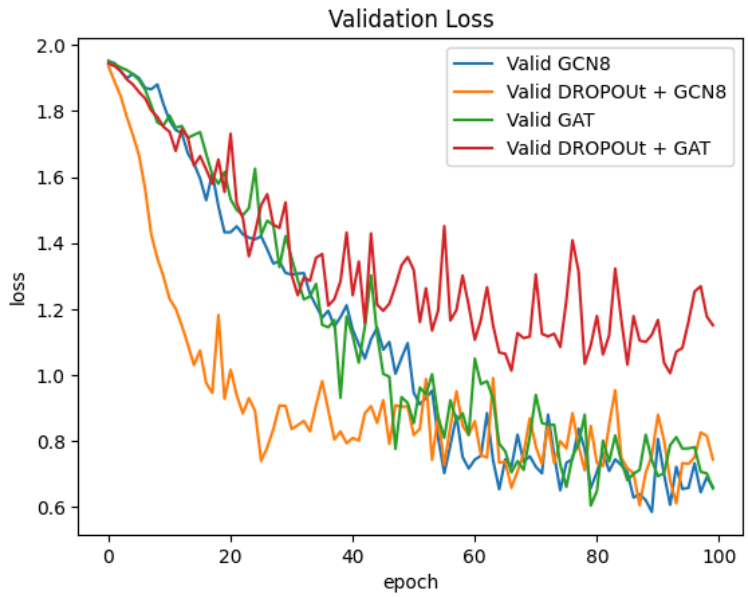
- **GCN8**
  - test:: 0.73,on the validation data: 0.67, time: 8.01, hidden size : 16
- **DROPEDGE + GCN8(random_edge func)**
  - the test: 0.78,on the validation data: 0.75, time: 6.10, hidden size : 16, probability: 0.7
- **GAT**
  - test:: 0.69,on the validation data: 0.65, time: 8.74, hidden size : 16
- **DROEDGE + GAT**

- test:: 0.80,on the validation data: 0.76, time: 9.62, hidden size : 16, probability: 0.5




**CoraFull**:

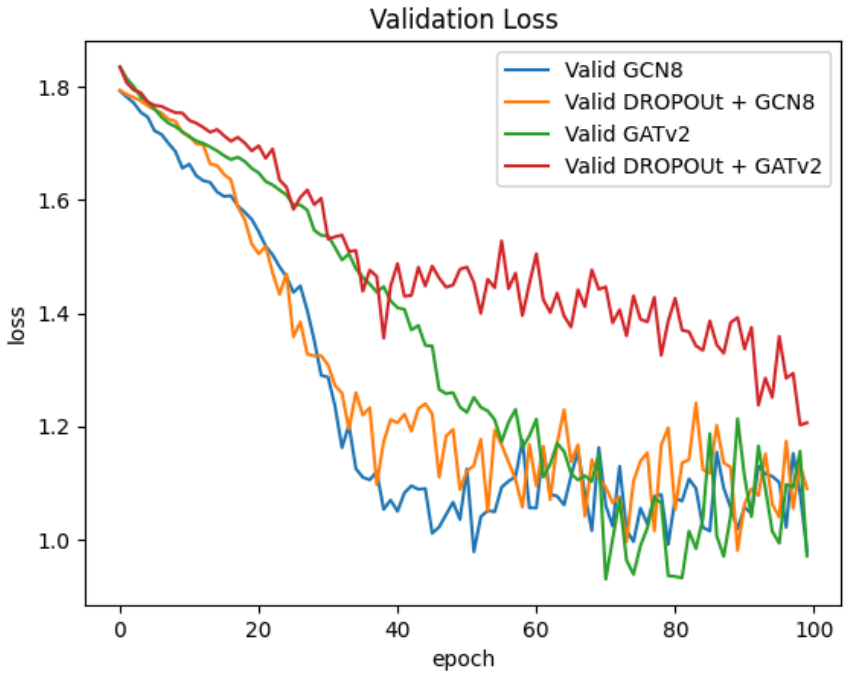
- **GCN8**
  - test: 0.83,on the validation data: 0.82, time: 4.09, hidden size : 16
- **DROPEDGE + GCN8(random_edge func)**
  - test: 0.84,on the validation data: 0.84, time: 5.07, hidden size : 16, probability: 0.7
- **GAT**:
  - test: 0.86,on the validation data: 0.83, time: 7.17, hidden size : 16
- **DROEDGE + GAT**
  - test:: 0.87,on the validation data: 0.86, time: 4.87, hidden size : 16, probability: 0.05




**GATv2:**

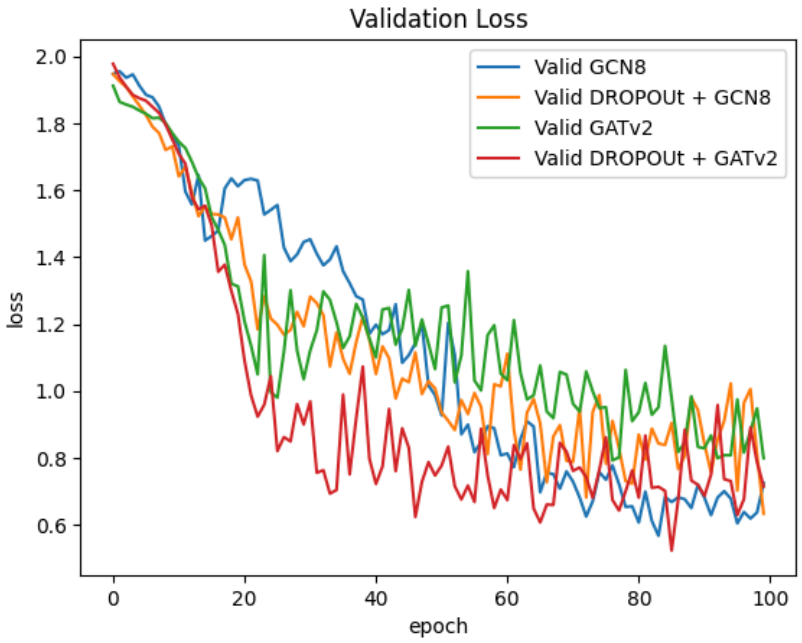
**Citeseer:**

- **GCN8**
  - test: 0.73,on the validation data: 0.75, time: 9.29, hidden size : 16
- **DROPEDGE + GCN8(random_edge func)**
  - test: 0.78,on the validation data: 0.78, time: 6.72, hidden size : 16, probability: 0.5
- **GATv2:**
  - test: 0.77,on the validation data: 0.77, time: 14.19, hidden size : 16
- **DROEDGE + GAT v2**
  - test: 0.77,on the validation data: 0.79, time: 11.55, hidden size : 16, probability: 0.05




**CoraFull**:

- **GCN8**
  - test:: 0.79,on the validation data: 0.83, time: 4.32, hidden size : 16
- **DROPEDGE + GCN8(random_edge func)**
  - test: 0.82,on the validation data: 0.88, time: 3.96, hidden size : 16, probability: 0.8
- **GATv2**:
  - test: 0.69,on the validation data: 0.74, time: 9.98, hidden size : 16
- **DROEDGE + GAT v2**
  - test: 0.80,on the validation data: 0.87, time: 9.71, hidden size : 16, probability: 0.8




| | **GCN8** | **GCN8 +DE** | **SKIP C** | **SKIP C + DE** | **GAT** | **GAT +DE** | **GATv2** | **GATv2 +DE** |
|---|---|---|---|---|---|---|---|---|
| **Cite Seer** | 0.73 | 0.78 | 0.76 | 0.77 | 0.69 | 0.8 | 0.77 | 0.77 |
| **Cora** | 0.75 | 0.84 | 0.84 | 0.86 | 0.86 | 0.87 | 0.69 | 0.8 |

با توجه به جدول فوق، برای هر دو دیتاست ترکیب GAT با DROP EDGE دارای بالاترین میزان دقت میباشد.

**مدل پیشنهادی : حذف تصادفی یالها و اثر گذاری یالهای باقی مانده براساس ویژگی‌های آنها و افزودن مکانیزم توجه به هر یال**