تمرین سوم درس تحلیل شبکه‌های پیچیده

(پروژه نهایی درس)

نام استاد درس: دکتر حقیرچهرقانی

نام دانشجو: مجید ادیبیان

شماره دانشجویی: ۴۰۲۱۳۱۹۱۰

زمستان ۱۴۰۲

سوال ۱:

الف) مجموعه داده‌های مورد نظر با استفاده از کتابخانه‌ی pytorch-geometric بارگذاری شده‌اند و سپس تابعی پیاده‌سازی شده که اطلاعات خواسته شده را از این داده‌ها استخراج کرده و نمایش می‌دهد:

نام داده	تعداد گره‌ها	تعداد یال‌ها	تعداد کلاس‌ها	تعداد ویژگی‌های هر گره
CoraFull	19793	126842	70	8710
CiteSeer	3327	9104	6	3703

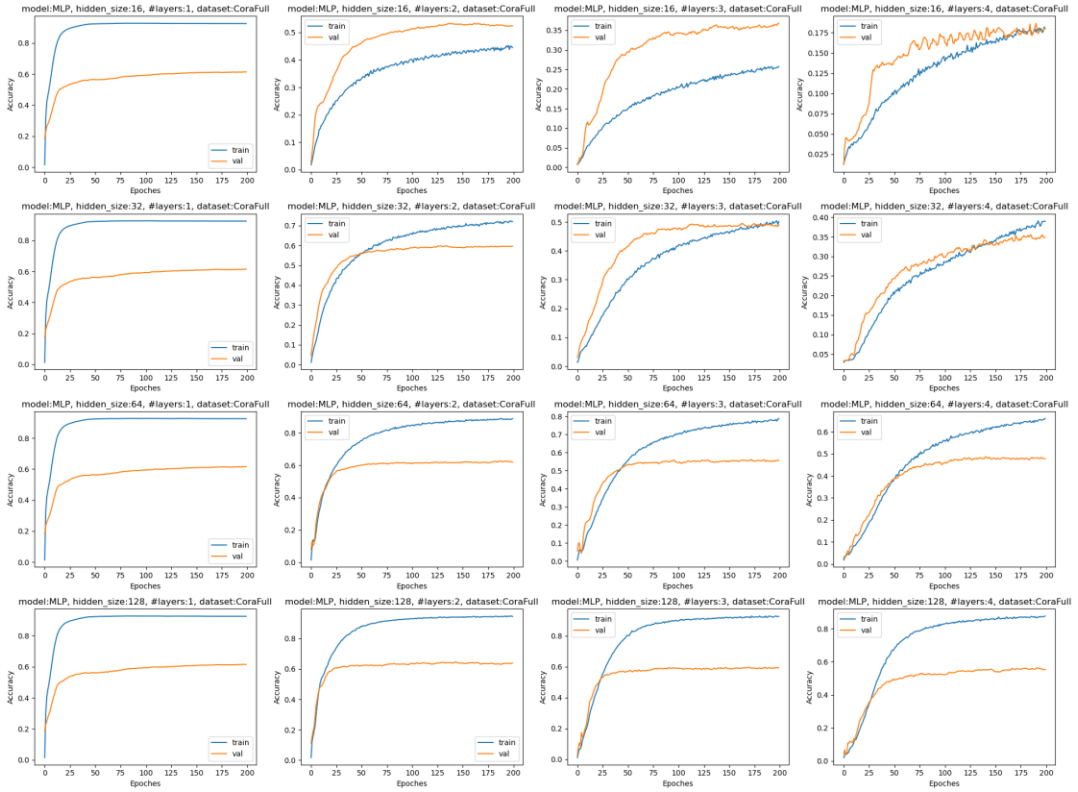
سپس جهت جداسازی داده‌های آموزشی، ارزیابی و آزمون از یکدیگر گره‌های مربوط به **هر کلاس** به سه دسته با درصدهای خواسته شده تقسیم شده‌اند که این فرایند در تابع split_data انجام شده است.

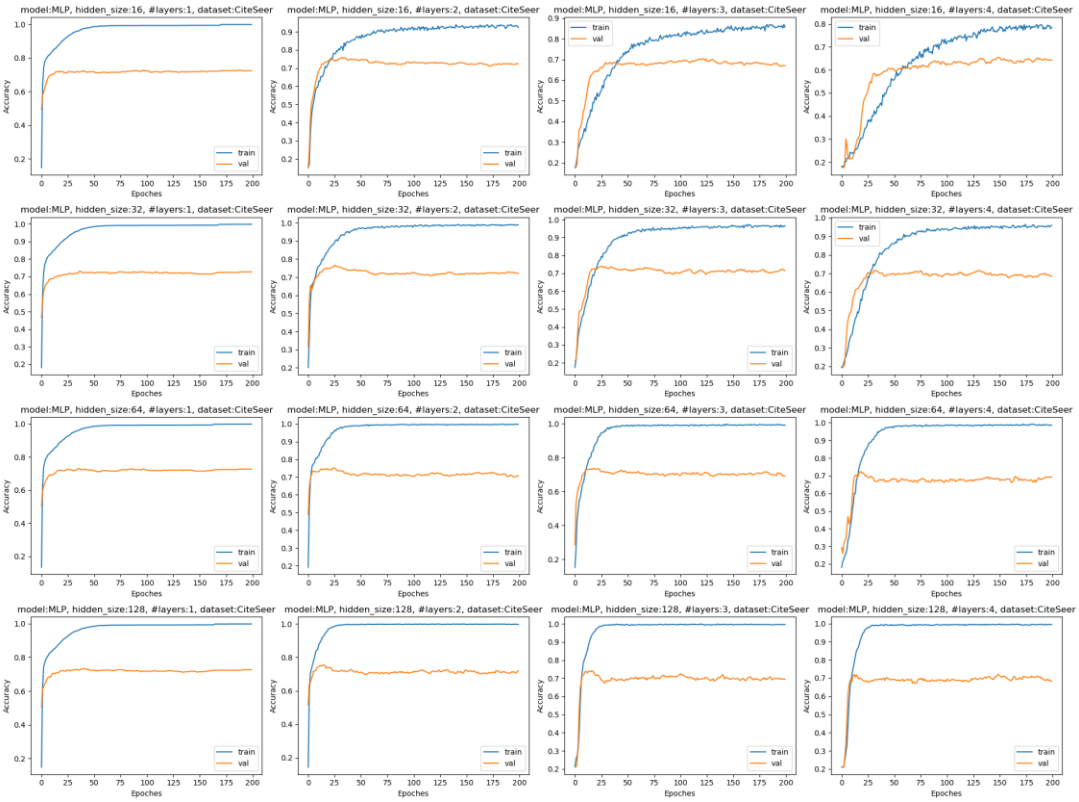
ب) در این قسمت مدل MLP ساخته شده است که در طراحی این مدل تعداد لایه‌های آن و ابعاد لایه‌های میانی به عنوان ورودی گرفته می‌شود. در این مدل هر بار ابتدا یک لایه خطی، یک Batch Normalization، یک تابع فعال‌سازی LeakyRelu و یک dropout با احتمال ۰.۵ قرار گرفته است. آموزش مدل با بهینه‌سازی Adam و نرخ‌یادگیری ۰.۰۰۵ انجام شده است.

جهت یافتن بهترین معماری برای تعداد لایه‌ها مقادیر ۱، ۲، ۳ و ۴ و برای ابعاد لایه‌های میانی ۱۶، ۳۲، ۶۴ و ۱۲۸ بررسی شده‌اند. آموزش تمام مدل‌ها تا ۲۰۰ epoch انجام شده است و پس از هر epoch دقت بر روی داده‌های آموزشی و ارزیابی به دست آمده است که نمودار تغییرات دقت در طی ۱۰۰ epoch آموزش برای تمام ۱۶ مدل آموزش دیده و برای هر داده به در صفحه بعد آورده شده است که جزئیات هر آموزش در عنوان هر نمودار درج شده است.

با توجه به نمودارهای به دست آمده می‌توان بهترین معماری را بر اساس بالاترین دقت به دست آمده از داده‌های ارزیابی به دست آورد که بهترین معماری در هر یک از داده‌ها در جدول زیر آورده شده است. سپس دقت مدل با بهترین معماری به دست آمده بر روی داده‌های آزمون محاسبه می‌شود که خروجی آن در جدول زیر دیده می‌شود:

نام داده	تعداد لایه در بهترین معماری MLP	ابعاد لایه‌های میانی در بهترین معماری MLP	دقت بهترین معماری بر روی داده‌های آزمون
CoraFull	۲	۱۲۸	۰.۶۲۸۰
CiteSeer	۲	۶۴	۰.۶۹۶۱


model:MLP, hidden_size:16, #layers:1, dataset:CoraFull
model:MLP, hidden_size:16, #layers:2, dataset:CoraFull
model:MLP, hidden_size:16, #layers:3, dataset:CoraFull
model:MLP, hidden_size:16, #layers:4, dataset:CoraFull
model:MLP, hidden_size:32, #layers:1, dataset:CoraFull
model:MLP, hidden_size:32, #layers:2, dataset:CoraFull
model:MLP, hidden_size:32, #layers:3, dataset:CoraFull
model:MLP, hidden_size:32, #layers:4, dataset:CoraFull
model:MLP, hidden_size:64, #layers:1, dataset:CoraFull
model:MLP, hidden_size:64, #layers:2, dataset:CoraFull
model:MLP, hidden_size:64, #layers:3, dataset:CoraFull
model:MLP, hidden_size:64, #layers:4, dataset:CoraFull
model:MLP, hidden_size:128, #layers:1, dataset:CoraFull
model:MLP, hidden_size:128, #layers:2, dataset:CoraFull
model:MLP, hidden_size:128, #layers:3, dataset:CoraFull
model:MLP, hidden_size:128, #layers:4, dataset:CoraFull
Accuracy
Epoches
train
val

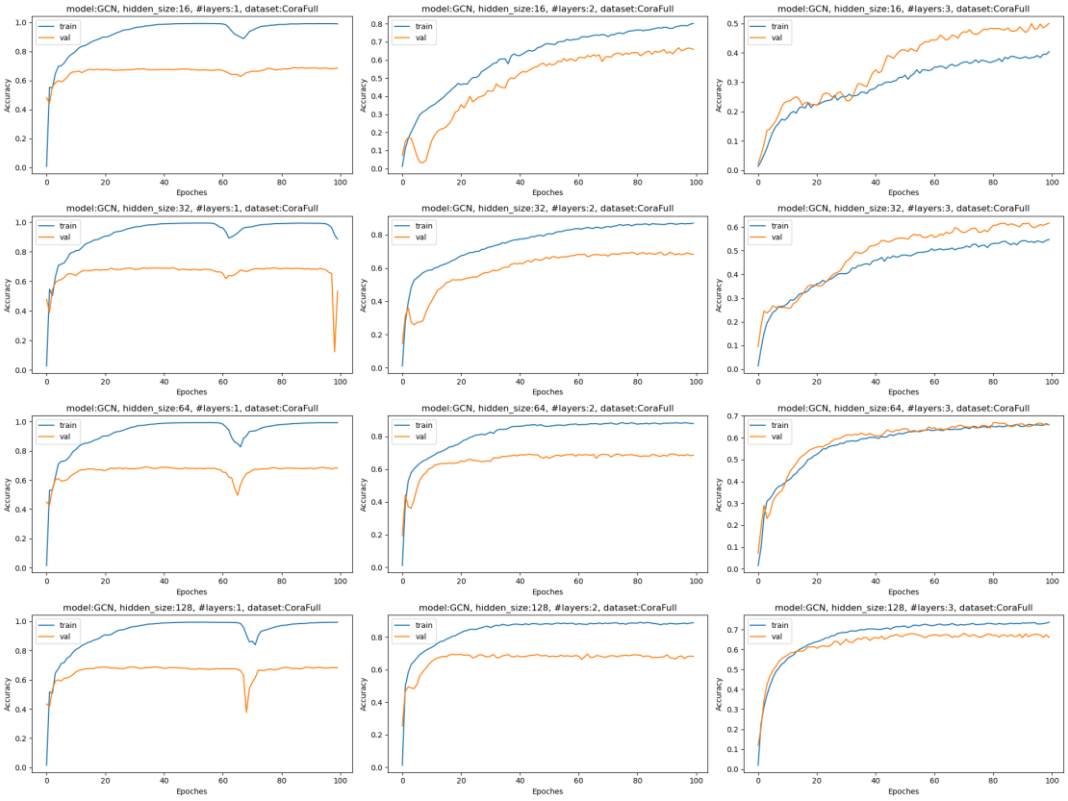

model:MLP, hidden_size:16, #layers:1, dataset:CiteSeer
model:MLP, hidden_size:16, #layers:2, dataset:CiteSeer
model:MLP, hidden_size:16, #layers:3, dataset:CiteSeer
model:MLP, hidden_size:16, #layers:4, dataset:CiteSeer
model:MLP, hidden_size:32, #layers:1, dataset:CiteSeer
model:MLP, hidden_size:32, #layers:2, dataset:CiteSeer
model:MLP, hidden_size:32, #layers:3, dataset:CiteSeer
model:MLP, hidden_size:32, #layers:4, dataset:CiteSeer
model:MLP, hidden_size:64, #layers:1, dataset:CiteSeer
model:MLP, hidden_size:64, #layers:2, dataset:CiteSeer
model:MLP, hidden_size:64, #layers:3, dataset:CiteSeer
model:MLP, hidden_size:64, #layers:4, dataset:CiteSeer
model:MLP, hidden_size:128, #layers:1, dataset:CiteSeer
model:MLP, hidden_size:128, #layers:2, dataset:CiteSeer
model:MLP, hidden_size:128, #layers:3, dataset:CiteSeer
model:MLP, hidden_size:128, #layers:4, dataset:CiteSeer
Accuracy
Epoches
train
val

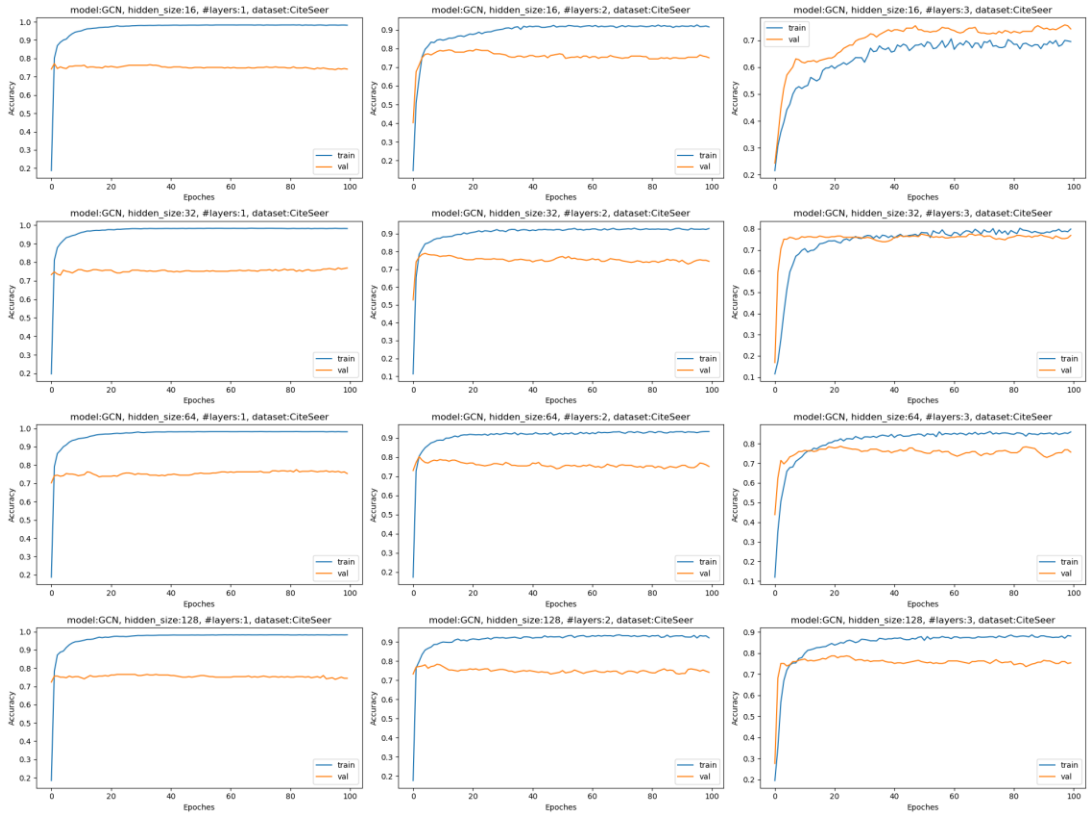
ج) در این قسمت ابتدا مدل GCN پیاده‌سازی شده است که در آن تعداد لایه‌ها و ابعاد لایه پنهان به عنوان ورودی گرفته می‌شود تا مدل متناسب با آن ساخته شود. سپس به انواع حالت‌های تعداد لایه‌ها با ۲، ۳ و ۴ لایه و ابعاد لایه پنهان ۱۶، ۳۲، ۶۴ و ۱۲۸ مدل GCN ساخته شده است. در هر لایه از مدل ساخته شده به ترتیب یک لایه خطی، یک Batch Normalization، یک dropout، یک تابع فعال‌سازی LeakyRelu و یک GCN قرار گرفته و در انتها نیز از soft-max استفاده شده است. آموزش هر حالت از مدل تا ۱۰۰ epoch انجام شده است.

مانند قسمت قبل تغییرات دقت بر روی داده‌های آموزشی و ارزیابی در نمودارهایی رسم شده است که خروجی این نمودارها برای تمام ۱۲ حالت گفته شده در صفحه بعد آورده شده است.

با بررسی نمودارهای تولید شده بهترین مقادیر برای تعداد لایه‌ها و ابعاد لایه پنهان برای هر یک از دو مجموعه داده مورد ارزیابی مشخص می‌شود که این مقادیر در جدول زیر آورده شده‌اند. سپس مدل با بهترین معماری به دست آمده بر روی داده‌های آزمون ارزیابی می‌شود که دقت آن در جدول زیر قابل مشاهده است.

نام داده	تعداد لایه در بهترین معماری GCN	ابعاد لایه‌های میانی در بهترین معماری GCN	دقت بهترین معماری بر روی داده‌های آزمون
CoraFull	۳	۶۴	۰.۶۷۳۴
CiteSeer	۳	۳۲	۰.۷۳۲۰

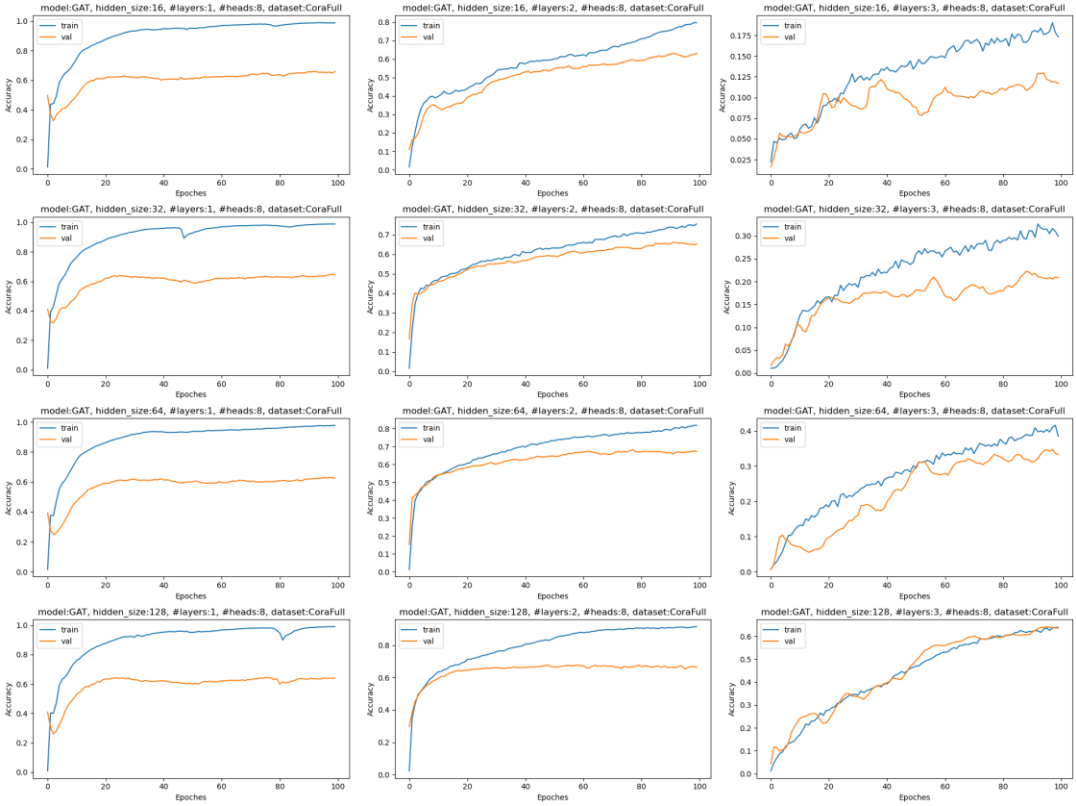


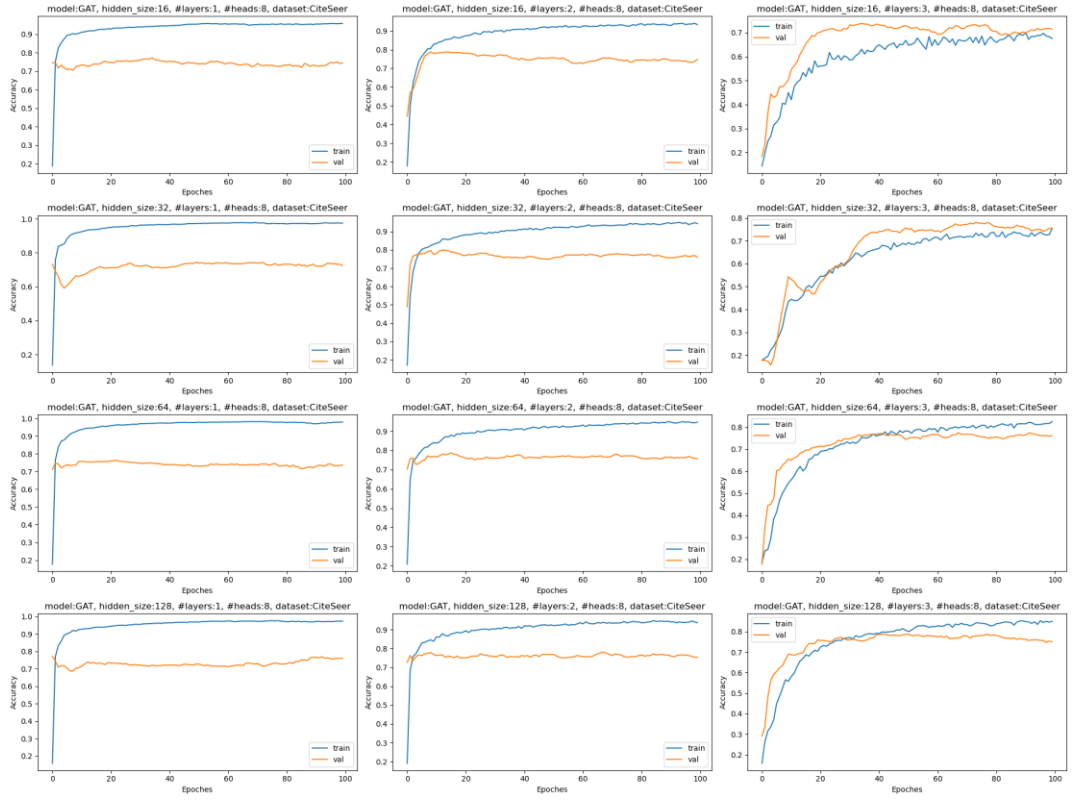


د) در این قسمت مانند قسمت قبل ابتدا مدل GAT ساخته شده است و سپس مدل با تعداد مختلف لایه‌ها و ابعاد مختلف لایه پنهان آموزش دیده است. هر لایه از مدل مانند قسمت قبل است و فقط به جای GCN از GAT استفاده شده است. در این آزمایش تعداد سرهای توجه ثابت و برابر ۱۶ در نظر گرفته شده تا در آزمایش بعد مقدار بهینه برای آن یافته شود. نمودارهایی که در ادامه آورده شده، تغییرات نمودارهای دقت داده‌های آموزشی و ارزیابی در طی ۱۰۰ epoch از آموزش برای هر دو مجموعه داده را نشان می‌دهند.

با بررسی این نمودارها مقادیر بهینه برای تعداد لایه‌ها و ابعاد لایه پنهان مشخص می‌شود که در جدول زیر آورده شده است.

نام داده	تعداد لایه در بهترین معماری GAT	ابعاد لایه‌های میانی در بهترین معماری GAT	تعداد سرهای توجه در بهترین معماری GAT	دقت بهترین معماری بر روی داده‌های آزمون
CoraFull	۲	۳۲	در قسمت بعد..	در قسمت بعد..
CiteSeer	۳	۳۲	در قسمت بعد..	در قسمت بعد..

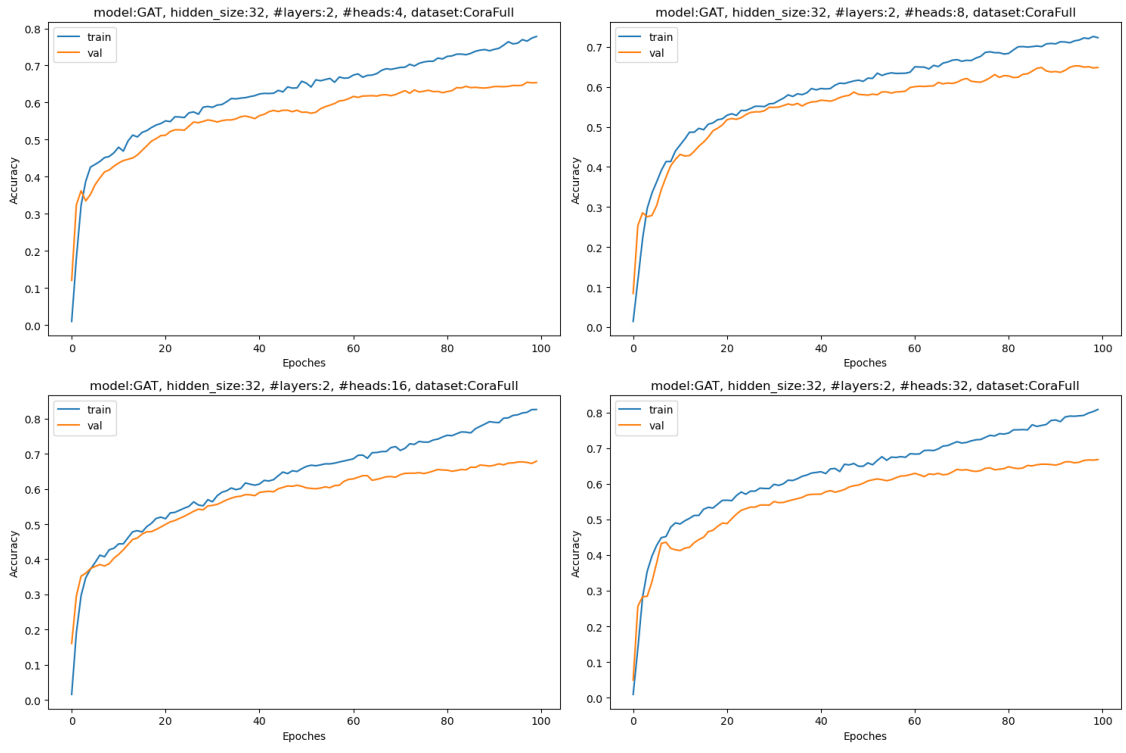

model:GAT, hidden_size:16, #layers:1, #heads:8, dataset:CoraFull
model:GAT, hidden_size:16, #layers:2, #heads:8, dataset:CoraFull
model:GAT, hidden_size:16, #layers:3, #heads:8, dataset:CoraFull
model:GAT, hidden_size:32, #layers:1, #heads:8, dataset:CoraFull
model:GAT, hidden_size:32, #layers:2, #heads:8, dataset:CoraFull
model:GAT, hidden_size:32, #layers:3, #heads:8, dataset:CoraFull
model:GAT, hidden_size:64, #layers:1, #heads:8, dataset:CoraFull
model:GAT, hidden_size:64, #layers:2, #heads:8, dataset:CoraFull
model:GAT, hidden_size:64, #layers:3, #heads:8, dataset:CoraFull
model:GAT, hidden_size:128, #layers:1, #heads:8, dataset:CoraFull
model:GAT, hidden_size:128, #layers:2, #heads:8, dataset:CoraFull
model:GAT, hidden_size:128, #layers:3, #heads:8, dataset:CoraFull
train
val
Accuracy
Epoches

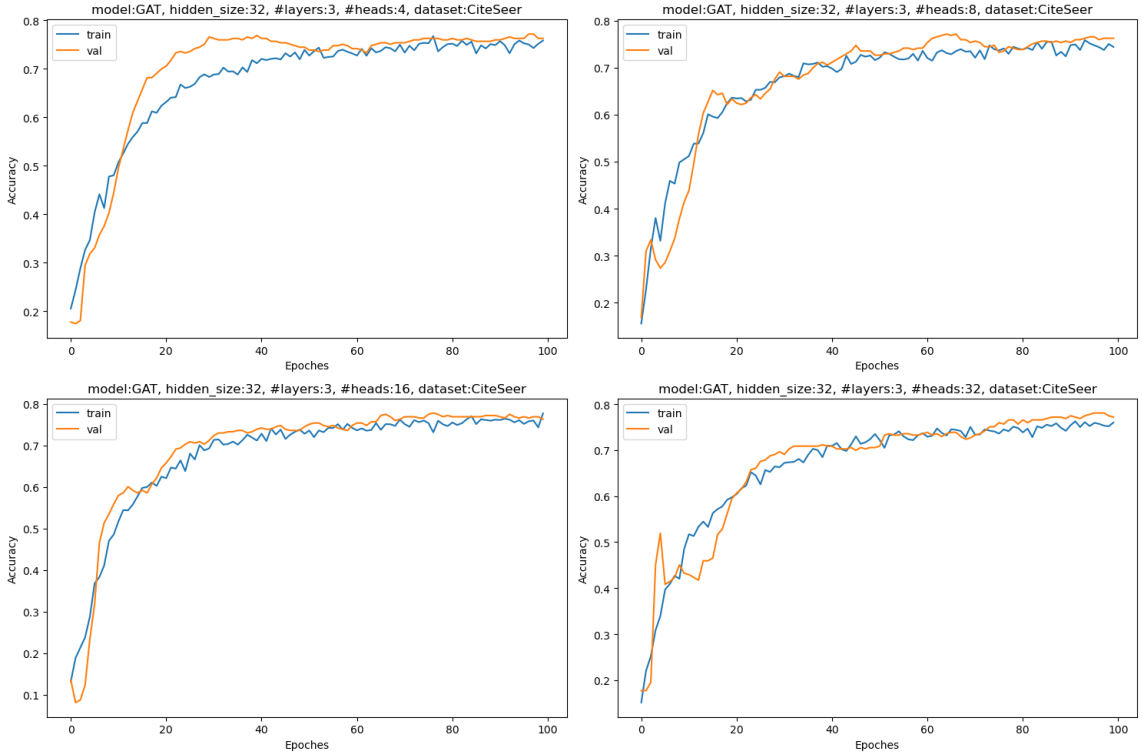

model:GAT, hidden_size:16, #layers:1, #heads:8, dataset:CiteSeer
model:GAT, hidden_size:16, #layers:2, #heads:8, dataset:CiteSeer
model:GAT, hidden_size:16, #layers:3, #heads:8, dataset:CiteSeer
model:GAT, hidden_size:32, #layers:1, #heads:8, dataset:CiteSeer
model:GAT, hidden_size:32, #layers:2, #heads:8, dataset:CiteSeer
model:GAT, hidden_size:32, #layers:3, #heads:8, dataset:CiteSeer
model:GAT, hidden_size:64, #layers:1, #heads:8, dataset:CiteSeer
model:GAT, hidden_size:64, #layers:2, #heads:8, dataset:CiteSeer
model:GAT, hidden_size:64, #layers:3, #heads:8, dataset:CiteSeer
model:GAT, hidden_size:128, #layers:1, #heads:8, dataset:CiteSeer
model:GAT, hidden_size:128, #layers:2, #heads:8, dataset:CiteSeer
model:GAT, hidden_size:128, #layers:3, #heads:8, dataset:CiteSeer
train
val
Accuracy
Epoches

در ادامه‌ی قسمت «د» پس از مشخص شدن بهترین مقادی برای تعداد لایه‌ها و ابعاد لایه پنهان، بهترین مقدار برای تعداد سرهای توجه را به دست می‌آوریم. برای این کار مدل GAT را با بهترین مقادیر به دست آمده برای تعداد لایه‌ها و ابعاد لایه پنهان و حالت‌های مختلف تعداد سرهای توجه شامل ۴، ۸، ۱۶ و ۳۲ آموزش می‌دهیم تا بهترین مقدار برای تعداد سرهای توجه نیز یافته شود.

نمودارهای تغییرات دقت بر روی داده‌های آموزشی و ارزیابی در هر دو مجموعه داده در نمودارها صفحه بعد آورده شده است. با بررسی این نمودارها بهترین مقدار برای تعداد سرهای توجه به دست می‌آید که در جدول زیر درج شده است. در نهایت دقت بهترین معماری به دست آمده بر روی داده‌های آزمون محاسبه می‌شود.

نام داده	تعداد لایه در بهترین معماری GAT	ابعاد لایه‌های میانی در بهترین معماری GAT	تعداد سرهای توجه در بهترین معماری GAT	دقت بهترین معماری بر روی داده‌های آزمون
CoraFull	۲	۳۲	۱۶	۰.۶۹۳۳
CiteSeer	۳	۳۲	۳۲	۰.۷۱۸۶

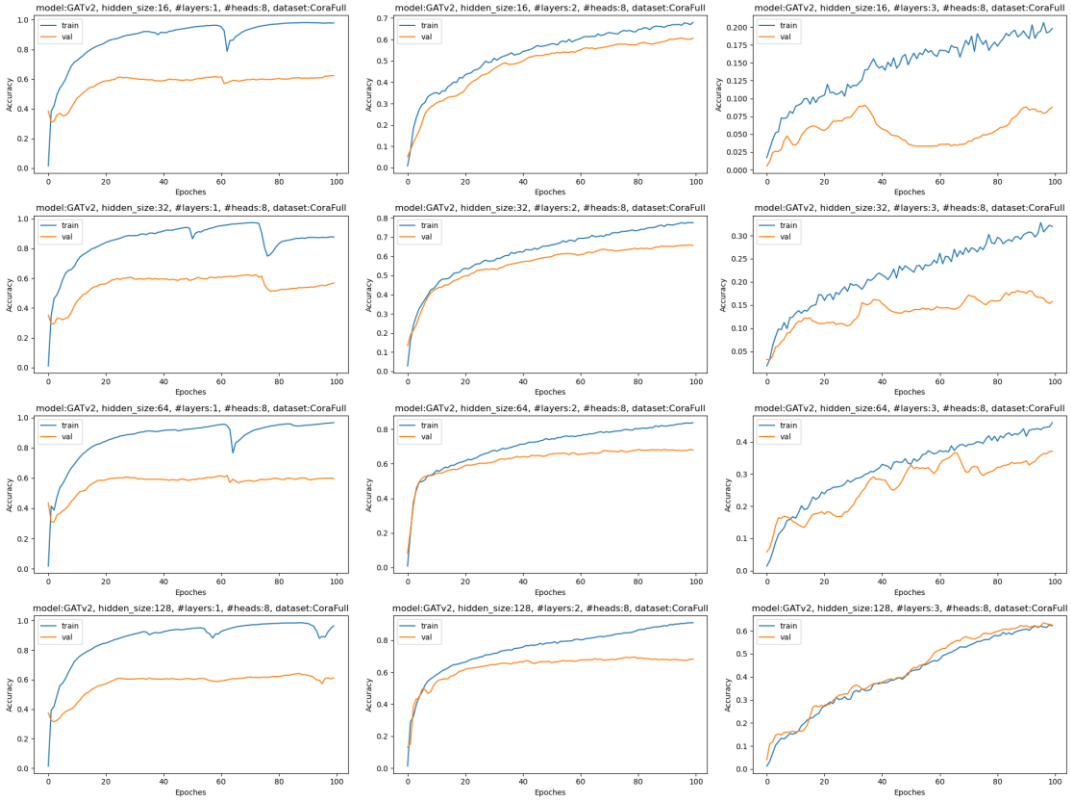


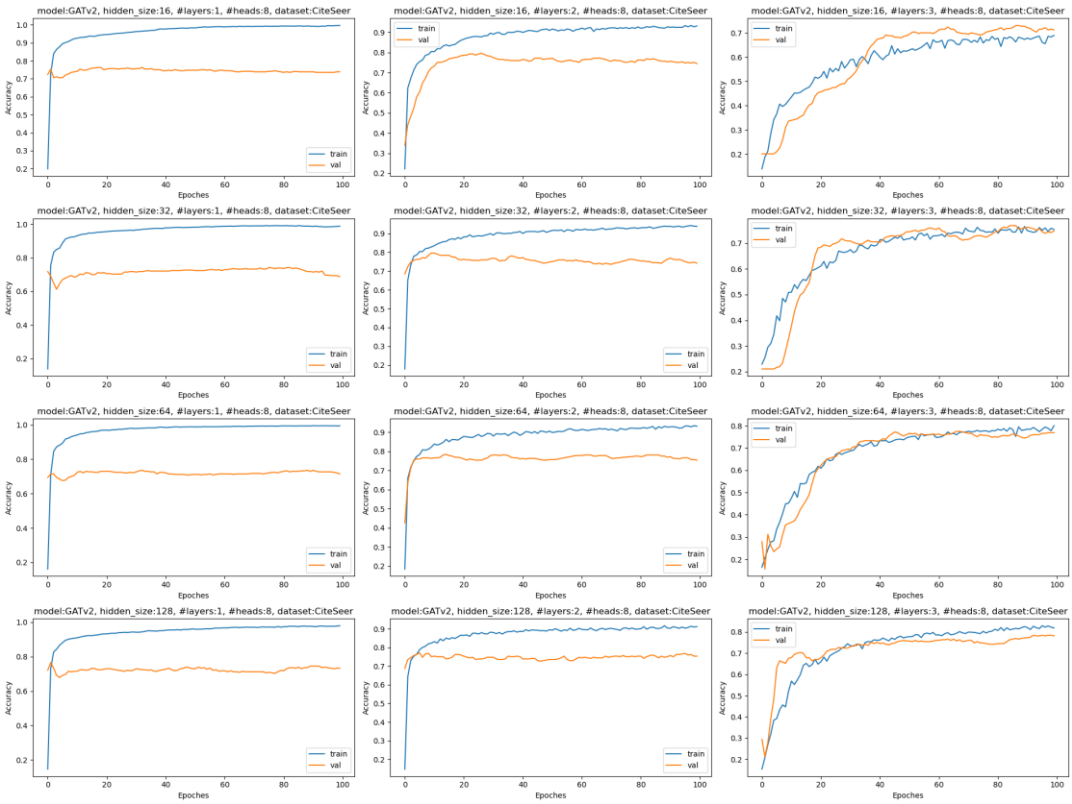


ه) قسمت قبل را برای مدل GAT-v2 تکرار می‌کنیم. در این مدل نیز مانند مدل قبل تعداد لایه‌ها و ابعاد لایه پنهان و تعداد سرهای توجه به عنوان ورودی گرفته می‌شود تا مدل متناسب با آن‌ها ساخته شود. هر لایه از مدل مانند قبل می‌باشد و فقط به جای GAT از GAT-v2 استفاده شده است.

نمودارهای یافتن بهترین مقادیر برای تعداد لایه‌ها و ابعاد لایه پنهان در ادامه آورده شده است که بهترین مقادیر برای آن‌ها برای هر مجموعه داده به صورت زیر است:

نام داده	تعداد لایه در بهترین معماری GAT-v2	ابعاد لایه‌های میانی در بهترین معماری GAT-v2	تعداد سرهای توجه در بهترین معماری GAT-v2	دقت بهترین معماری بر روی داده‌های آزمون
CoraFull	۲	۳۲	در قسمت بعد..	در قسمت بعد..
CiteSeer	۳	۱۲۸	در قسمت بعد..	در قسمت بعد..

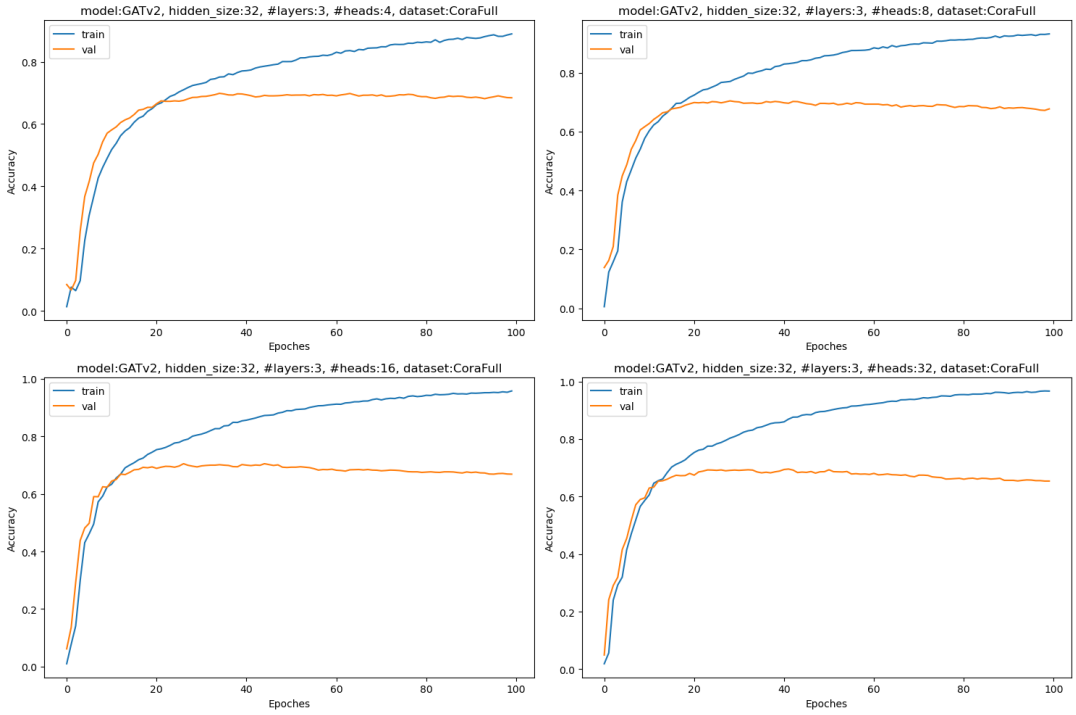

model:GATv2, hidden_size:16, #layers:1, #heads:8, dataset:CoraFull
model:GATv2, hidden_size:16, #layers:2, #heads:8, dataset:CoraFull
model:GATv2, hidden_size:16, #layers:3, #heads:8, dataset:CoraFull
model:GATv2, hidden_size:32, #layers:1, #heads:8, dataset:CoraFull
model:GATv2, hidden_size:32, #layers:2, #heads:8, dataset:CoraFull
model:GATv2, hidden_size:32, #layers:3, #heads:8, dataset:CoraFull
model:GATv2, hidden_size:64, #layers:1, #heads:8, dataset:CoraFull
model:GATv2, hidden_size:64, #layers:2, #heads:8, dataset:CoraFull
model:GATv2, hidden_size:64, #layers:3, #heads:8, dataset:CoraFull
model:GATv2, hidden_size:128, #layers:1, #heads:8, dataset:CoraFull
model:GATv2, hidden_size:128, #layers:2, #heads:8, dataset:CoraFull
model:GATv2, hidden_size:128, #layers:3, #heads:8, dataset:CoraFull
train
val
Accuracy
Epoches

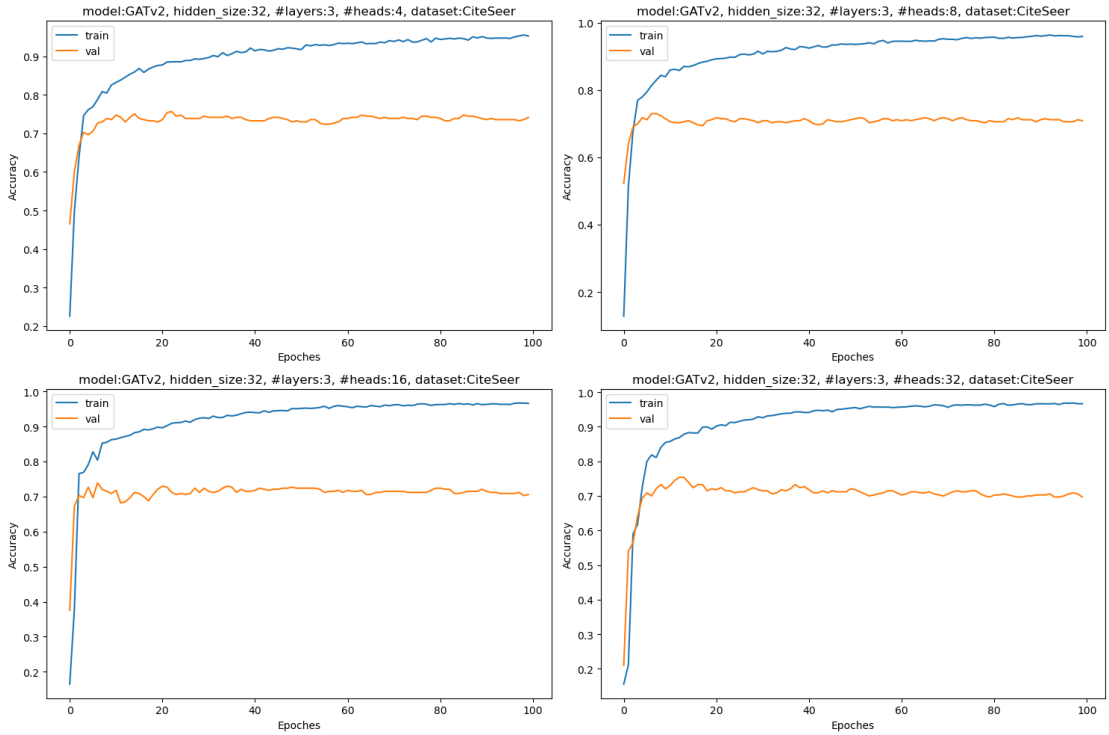

model:GATv2, hidden_size:16, #layers:1, #heads:8, dataset:CiteSeer
model:GATv2, hidden_size:16, #layers:2, #heads:8, dataset:CiteSeer
model:GATv2, hidden_size:16, #layers:3, #heads:8, dataset:CiteSeer
model:GATv2, hidden_size:32, #layers:1, #heads:8, dataset:CiteSeer
model:GATv2, hidden_size:32, #layers:2, #heads:8, dataset:CiteSeer
model:GATv2, hidden_size:32, #layers:3, #heads:8, dataset:CiteSeer
model:GATv2, hidden_size:64, #layers:1, #heads:8, dataset:CiteSeer
model:GATv2, hidden_size:64, #layers:2, #heads:8, dataset:CiteSeer
model:GATv2, hidden_size:64, #layers:3, #heads:8, dataset:CiteSeer
model:GATv2, hidden_size:128, #layers:1, #heads:8, dataset:CiteSeer
model:GATv2, hidden_size:128, #layers:2, #heads:8, dataset:CiteSeer
model:GATv2, hidden_size:128, #layers:3, #heads:8, dataset:CiteSeer
train
val
Accuracy
Epoches

سپس مدل با بهترین مقادیر به دست آمده از قسمت قبل و با مقادیر مختلف برای تعداد سرهای توجه آموزش داده می‌شود که نتایج تغییرات این نمودارها در صفحه بعد آورده شده است.

در نهایت با مشخص شدن بهترین معماری برای این سه مقدار دقت بهترین مدل بر روی داده‌های آزمون محاسبه می‌شود که این نتایج در جدول زیر قابل مشاهده است:

نام داده	تعداد لایه در بهترین معماری GAT-v2	ابعاد لایه‌های میانی در بهترین معماری GAT-v2	تعداد سرهای توجه در بهترین معماری GAT-v2	دقت بهترین معماری بر روی داده‌های آزمون
CoraFull	۲	۳۲	۸	۰.۶۷۵۲
CiteSeer	۳	۱۲۸	۴	۰.۷۵۳۰





و) بهترین معماری مدل‌های MLP، GCN، GAT و GAT-v2 در قسمت‌های قبلی به دست آمد و دقت هر یک بر روی داده‌های آزمون محاسبه شد. حال زمان لازم برای آموزش و استنتاج از این مدل‌ها را بررسی کرده و تفاوت دقت‌ها و زمان‌ها را تحلیل می‌کنیم. در این بررسی زمان آموزش بر روی داده‌های آموزشی و زمان استنتاج (پیش‌بینی برچسب) بر روی داده‌های آزمون استفاده خواهد شد و برای محاسبه این زمان‌ها از یک واحد پردازش گرافی Nvidia GeForce RTX 3070 Ti استفاده شده است.

جدول زیر اطلاعات گفته شده را به طور کامل نشان می‌دهد:

نام داده	مدل	تعداد لایه‌ها	ابعاد لایه پنهان	تعداد سرهای توجه	دقت بر روی داده‌های آزمون	زمان آموزش داده‌های آموزشی (ثانیه)	زمان استنتاج داده‌های آزمون (ثانیه)
CoraFull	MLP	۲	۱۲۸	–	۰.۶۲۸۵	**۱.۵۸۹۵**	**۰.۰۰۳۸**
	GCN	۳	۶۴	–	**۰.۶۷۹۲**	۶۶.۵۳۳۱	۰.۳۳۵۶
	GAT	۲	۳۲	۱۶	۰.۶۷۸۷	**۶۵.۹۶۵۱**	**۰.۲۲۳۹**
	GAT-v2	۲	۳۲	۸	۰.۶۷۵۲	۶۷.۲۷۲۶	۰.۲۲۷۲
CiteSeer	MLP	۲	۶۴	–	۰.۷۰۸۱	**۰.۱۵۱۳**	**۰.۰۰۰۴**
	GCN	۳	۳۲	–	۰.۷۴۱۰	**۲.۷۹۱۶**	**۰.۰۰۸۷**
	GAT	۳	۳۲	۳۲	**۰.۷۵۷۵**	۲.۹۵۷۶	۰.۰۰۹۱
	GAT-v2	۳	۱۲۸	۴	۰.۷۵۳۰	۳.۲۰۵۸	۰.۰۰۹۶

با مقایسه نتایج به دست آمده دیده می‌شود که مدل MLP که از اطلاعات گراف استفاده نمی‌کند و تنها با استفاده از ویژگی‌های گره‌ها پیش‌بینی را انجام می‌دهد نسبت به مدل‌های گرافی دقت پایین‌تری دارد. در مدل‌های گرافی دقت مدل‌ها بر روی داده‌های CuraFull به هم نزدیک بوده و اختلاف محسوسی ندارند و حدود ۵ درصد نسبت به مدل MLP بهبود یافته‌اند. دقت این مدل‌ها بر روی داده‌های CiteSeer هم حدود ۵ درصد بهبود را نشان می‌دهد که البته در این داده مدل GAT و GAT-v2 توانسته‌اند دقت بهتری نسبت به بقیه کسب کنند. در بررسی سرعت هر یک از مدل‌ها همان‌طور که انتظار می‌رفت مدل MLP با توجه به پیچیدگی محاسباتی کمتر از مدل‌های گرافی سرعت آموزش و استنتاج بسیار بیشتری دارد. سه مدل دیگر به لحاظ سرعت اختلاف کمی با یکدیگر دارند که علت اصلی این اختلاف هم تفاوت در تعداد پارامترهای آن‌ها به دلیل تفاوت در تعداد لایه‌ها و ابعاد لایه مخفی استفاده شده در آن‌هاست.

ز) پیچیدگی زمانی و حافظه‌ای برای مدل‌های GCN، GAT و GAT-v2 به صورت زیر است:

GCN: در این مدل ابتدا بردار اولیه‌ی تمام گره‌ها به ابعاد d در یک ماتریس قابل آموزش به ابعاد d در f ضرب می‌شود و سپس بردارهای همسایه‌های هر گره با هم میانگین گرفته می‌شوند. در نتیجه ضرب ماتریس برای تمام گره‌ها از O(ndf) و محاسبه میانگین همسایه‌های هر گره برای تمام گره‌ها شامل محاسبه‌ی جمع بردار f بعدی به تعداد دو برابر یال‌های گراف است که از O(mf) خواهد بود. پس پیچیدگی زمانی GCN از $O(ndf + mf)$ است.

با توجه به فرایند گفته شده نیاز است که یک ماتریس d در f برای تبدیل بردار گره‌های همسایه و یک بردار d بعدی برای تبدیل بردار خود گره مرکزی داشته باشیم که در این حالت پیچیدگی حافظه‌ای از $O(df)$ است.

GAT: در این مدل ابتدا مانند مدل GCN بردار d بعدی هر گره با ضرب در یک ماتریس قابل آموزش به بردار f بعدی تبدیل می‌شود (O(df)). همچنین مقدار توجه هر گره به هر گره همسایه محاسبه می‌شود که در حالت یک راسی این فرایند نیازمند یک عمل concatenation بر روی بردارهای دو گره و یک ضرب ماتریسی (یک لایه خطی) است (O(2d)). در انتها مانند GCN یک aggregation بر روی همسایه‌های گره انجام می‌شود. انجام تمام فرایند گفته شده برای تمام گره‌ها از $O(ndf + nd + mf)$ است. در حالت داشتن چند راس توجه محاسبه‌ی توجه هر گره به هر گره همسایه نیاز به k ماتریس مجزا دارد که در نتیجه در محاسبه توجه‌ها k ضرب ماتریسی و در aggregation برای هر راس یک aggregation مجزا انجام می‌شود که پیچیدگی زمانی در این حالت برابر $O(ndf + knd + kmf) = O(ndf + kmf)$ خواهد بود.

با توجه به این فرایند برای علاوه بر حافظه لازم مانند روش GCN از k ماتریس مجزا برای محاسبه‌ی راس‌های توجه هم استفاده می‌شود. پس پیچیدگی حافظه‌ای این روش از $O(df + kd)$ است.

GAT-v2: طبق آنچه در مقاله GAT-v2 بیان شده است (قسمت appendix-G مقاله به طور کامل توضیح داده شده) پیچیدگی زمان و حافظه‌ای مدل GAT-v2 مشابه با GAT است.

به طور خلاصه می‌توان پیچیدگی‌های زمانی و حافظه‌ای را برای این سه مدل به صورت زیر نوشت: (k تعداد راس‌های توجه است)

	پیچیدگی زمانی	پیچیدگی حافظه‌ای
GCN	$O(ndf + mf)$	$O(df)$
GAT	$O(ndf + kmf)$	$O(df + kd)$
GATv2	$O(ndf + kmf)$	$O(df + kd)$

سوال ۳:

الف) همان‌طور که می‌دانیم در مدل‌های GNN با افزایش تعداد لایه‌ها، گره‌های با فاصله بیشتر از گره فعلی در ساخت بردار تعبیه‌ی این گره استفاده خواهند شد. این فرایند باعث می‌شود تعداد گره‌های مشترک تاثیرگذار بر روی تعبیه‌ی گره‌های گراف افزایش یابد و بردار تعبیه آن‌ها مشابه شوند که به این مشکل over-smoothing می‌گویند. مقاله فعلی برای حل این مشکل روشی را پیشنهاد داده است که بر اساس آن در هر گام از آموزش مدل تعدادی از یال‌های گراف به صورت تصادفی حذف می‌شوند و سپس یک نرمال‌سازی بر روی ماتریس مجاورت گراف اعمال می‌شود. این فرایند باعث می‌شود که در هر گام از آموزش گره‌های متفاوتی در ساخت تعبیه‌ی گره‌ها تاثیرگذار باشند و مشکل over-smoothing تا حد امکان برطرف شود. همچنین در مقاله مطرح شده است که فرایند حذف تصادفی یال‌ها می‌تواند پس از هر یال انجام گیرد. علاوه بر این، حذف احتمالاتی برخی از یال‌های گراف در هر گام آموزش یا هر لایه از مدل باعث می‌شود مدل اطلاعات جزئی و خاص آن گراف را در یادگیری حفظ نکند و از overfitting جلوگیری می‌شود.

تفاوت‌ها و شباهت‌ها با روش Dropout: طبق توضیحات داده شده در قسمت DISCUSSIONS مقاله در روش dropout جهت جلوگیری از overfitting در ماتریس ویژگی‌های مدل به صورت تصادفی برخی از مقادیر به صفر تغییر می‌کنند. این روش برای جلوگیری از overfitting موثر است ولی در مشکل overestimation کمکی نمی‌کند چرا که ماتریس مجاورت گراف را تغییری نمی‌دهد و در نتیجه مانند حالت عادی با چند لایه متوالی در مدل بیشتر گره‌های گراف تاثیرگذار خواهند بود و بردار تعبیه گره‌ها مشابه می‌شود. در حالی که روش پیشنهادی مقاله که در آن سعی شده است به صورت تصادفی هر بار برخی از یال‌های گراف حذف شود را می‌توان نسل جدیدی از dropout دانست که به جای حذف برخی مقادیر در ماتریس ویژگی برخی یال‌ها را به صورت تصادفی حذف می‌کند. این روش باعث می‌شود از یک طرف مانند روش dropout مدل بر روی داده‌های آموزشی overfit نشود

و از طرفی دیگر ماتریس مجاورت گراف نیز تغییر می‌کند و تعداد گره‌های تاثیرگذار و مشترک بین گره‌ها کاهش خواهد یافت و تا حد امکان از over-smoothing نیز جلوگیری می‌شود.

ب) پیاده‌سازی این مقاله بر اساس پیاده‌سازی انجام شده در این لینک انجام گرفته است که البته بخش محدودی از پیاده‌سازی انجام شده برگرفته از آن است و سایر قسمت‌ها به صورت اختصاصی متناسب با سوالات خواسته شده در این پروژه پیاده‌سازی شده است. بر این اساس کدهای مربوط به sampling از یال‌های گراف که در فایل sampling.py قرار دارد و کدهای مربوط به توابع نرمال‌سازی ماتریس مجاورت گراف که در فایل normalization.py قرار دارد مستقیما از لینک گفته شده دریافت شده است ولی کدهای پیاده‌سازی مدل (فایل model.py)، آموزش مدل (فایل train.py) و پیش‌پردازش داده‌ها (فایل utils.py) در این پروژه پیاده‌سازی شده‌اند.

ج) مجموعه داده‌های گفته شده در سوال ۱ را استفاده کرده‌ایم و مطابق آن سه قسمت آموزشی و ارزیابی و آزمون را به دست آورده‌ایم. سپس مدل پیاده‌سازی شده بر روی این داده‌ها آموزش دیده که نتایج آن در جدول زیر آمده است. در این نتایج ستون دوم از راست دقت GCN دو لایه پیاده‌سازی شده در سوال ۱ بر روی داده‌های آزمون است و ستون سوم و چهارم دقت GCN ای است که پیاده‌سازی آن در قسمت قبل توضیح داده شد که در دومی از ۰.۰۵ یال‌ها نمونه‌برداری شده و در اولی هیچ نمونه‌برداری و drop edge ای انجام نشده است ولی از سایر مزایای پیاده‌سازی مقاله مانند نرمال‌سازی‌ها استفاده می‌کند. جهت مقایسه مناسب مدل‌ها تمامی مدل‌ها با هایپرپارامترهای یکسان آموزش داده شده‌اند.

مجموعه داده	تعداد لایه‌ها	ابعاد لایه پنهان	تعداد epochها	نرخ یادگیری	dropout
CoraFull	۲	۲۵۶	۲۰۰	۰.۰۰۱	۰.۹
CiteSeer	۲	۲۵۶	۲۰۰	۰.۰۰۹	۰.۹

مجموعه داده	GCN پیاده‌سازی شده در سوال ۱	DropEdge GCN, Edge sampling=1	DropEdge GCN, Edge sampling=0.05
CoraFull	۰.۶۸۷۲	۰.۶۸۴۷	۰.۷۲۴۳
CiteSeer	۰.۷۰۸۱	۰.۷۱۴۱	۰.۷۶۶۴

در این قسمت نتایج مدل DropEdge را در دو حالت بدون حذف هیچ یالی و با حذف ۹۵ درصد یال‌ها مقایسه می‌کنیم (مقایسه با نتایج پیاده‌سازی سوال ۱ در قسمت بعد). با مقایسه نتایج به دست آمده دیده می‌شود که مدل DropEdge با حذف تصادفی یال‌ها به خوبی توانسته است دقت را بر روی هر دو دسته داده بهبود قابل توجهی دهد. به طوری که مدل DropEdge نسبت به همین مدل بدون حذف هیچ یالی (و در عین حال استفاده

از سایر مزایای آن از جمله نرمال‌سازی‌های ماتریس مجاورت) باعث حدود ۴ درصد بهبود در داده‌های CoraFull و حدود ۶ درصد بهبود در داده‌های CiteSeer شده است.

د) نتایج پیاده‌سازی سوال ۱ با ۲ لایه GCN در قسمت قبل در جدول آورده شد. با مقایسه نتایج این مدل با مدل DropEdge دیده می‌شود که مدل DropEdge با حذف تصادفی یال‌ها و نرمال‌سازی‌های انجام شده توانسته است به خوبی دقت را بر روی هر دو مجموعه داده بهبود دهد. همچنین دیده می‌شود که تفاوت کمی بین مدل پیاده‌سازی شده در سوال ۱ و مدل DropEdge بدون حذف هیچ یالی است که دلیل آن شباهت این دو پیاده‌سازی در مدل و نحوه آموزش است.

ه) در جدول زیر نتایج گزارش شده از مقاله و نتایج به دست آمده از مدل پیاده‌سازی شده برای یک مدل دو لایه بر روی داده‌های CiteSeer دیده می‌شود.

	GCN دو لایه با DropEdge	GCN دو لایه بدون DropEdge
گزارش مقاله	۰.۷۸۷۰	۰.۷۵۹۰
پیاده‌سازی انجام شده	۰.۷۶۶۴	۰.۷۱۴۱

نتایج گزارش شده نشان می‌دهد که در حالت استفاده از DropEdge نتیج گزارش شده از مقاله و پیاده‌سازی شده در این پروژه شباهت زیادی به هم دارند ولی در حالت بدون DropEdge تفاوت نسبتا زیادی بین آن‌ها به وجود آمده است. دلیل این تفاوت می‌تواند تفاوت در تقسیم‌بندی داده‌ها به سه دسته‌ی آموزشی، ارزیابی و آزمون است. به طوری که در این پروژه از تقسیم‌بندی مجزایی استفاده شده است و با توجه به تعداد کم داده‌ها و وجود تنها 3327 گره در این گراف تفاوت در داده‌های آموزشی و آزمون می‌تواند تفاوت قابل توجهی را به وجود آورد.

و) همان‌طور که در قسمت الف نیز توضیح داده شد در مدل‌های GNN با افزایش تعداد لایه‌ها، گره‌های با فاصله بیشتر از گره فعلی در ساخت بردار تعبیه‌ی این گره استفاده خواهند شد. این فرایند باعث می‌شود تعداد گره‌های مشترک تاثیرگذار بر روی تعبیه‌ی گره‌های گراف افزایش یابد و بردار تعبیه آن‌ها مشابه شوند که به این مشکل over-smoothing می‌گویند. در نتیجه در مدل‌های گرافی افزایش تعداد لایه‌ها این مشکل را به وجود می‌آورد که راه‌کاری‌های متعددی برای حل این مشکل ارائه شده است.

راه‌کار عملی در تشخیص این پدیده آن است که تعداد لایه‌های مدل مورد نظر را به مرور افزایش دهیم و هر بار مدل آموزش دیده و دقت بر روی داده‌های آزمون به دست آید. در نتیجه نمودار تغییرات دقت در طی افزایش تعداد لایه‌ها به دست می‌آید که نشان خواهد داد تا چه تعداد افزایش لایه‌ها دقت را افزایش داده و سپس دقت سیر کاهش گرفته است و نقطه شروع به کاهش دقت شروع مشکل over-smoothing خواهد بود. همچنین از یک

معیار فاصله برای محاسبه میانگنی فاصله بردار تعبیه‌ی گره‌ها می‌توان استفاده کرد که از یک جایی به بعد این میانگین فاصله کاهشی می‌شود و بردار تعبیه‌ی گره‌ها به هم نزدیک می‌شوند.

ز) در پیاده‌سازی انجام شده تعداد لایه‌های مدل به عنوان ورودی گفته می‌شود که در نتیجه به سادگی می‌توان تعداد لایه‌های مدل را به ۸ لایه تغییر داد. در این قسمت هم مانند قسمت‌ها قبل برای هر مجموعه داده آموزش به سه روش انجام شده است که در روش اول از پیاده‌سازی سوال ۱ استفاده شده، در روش دوم از پیاده‌سازی DropEdge استفاده شده ولی هیچ یالی حذف نشده است و در روش سوم پیاده‌سازی DropEdge با حذف تصادفی درصدی از یال‌ها استفاده شده است.

جهت مقایسه صحیح هر سه مدل در یک مجموعه داده با هایپرپارامترهای مشابه آموزش دیده‌اند که به صورت زیر می‌باشد:

مجموعه داده	تعداد لایه‌ها	ابعاد لایه پنهان	تعداد epochها	نرخ یادگیری	dropout
CoraFull	۸	۲۵۶	۲۰۰	۰.۰۰۱	۰.۲
CiteSeer	۸	۶۴	۵۰۰	۰.۰۰۹	۰.۲

نتایج به دست آمده از آموزش بر روی داده‌های آموزشی و سپس ارزیابی بر روی داده‌های آزمون برای سه مدل گفته شده و هر یک از دو مجموعه داده در جدول زیر مشخص شده است:

مجموعه داده	GCN پیاده‌سازی شده در سوال ۱	DropEdge GCN, Edge sampling=1	DropEdge GCN, Edge sampling=0.05
CoraFull	۰.۴۷۹۹	۰.۶۳۳۵	۰.۶۸۸۵
CiteSeer	۰.۵۶۲۹	۰.۶۸۸۶	۰.۷۳۶۵

همان‌طور که دیده می‌شود در اثر افزایش تعداد لایه‌ها دقت‌های به دست آمده نسبت به مدل با تعداد لایه‌های کمتر که در جدول قبلی مطرح شد کاهش داشته است که به دلیل همان مشکل over-smoothing است. در این جدول دیده می‌شود که با فزایش تعداد لایه‌های مدل پیاده‌سازی شده در سوال یک دقت به شدت افت داشته است ولی این افت در پیاده‌سازی مربوط به مدل DropEdge دیده نمی‌شود که دلیل آن برخی نرمال‌سازی‌ها در این پیاده‌سازی است. همچنین مشخص است که در اثر حذف تصادفی برخی از یال‌ها (ستون آخر) دقت کاهش‌یافته در مدل‌های ساده بهبود یافته و تا حدی تاثیر مشکل over-smoothing برطرف شده است.

ح) در مدل پیاده‌سازی شده در این سوال و سوال یک می‌توان ورودی skip-connection را برابر True قرار داد تا نحوه اتصالات لایه‌ها با skip-connection باشد. حال یک مدل ۸ لایه با skip-connection ساخته می‌شود و بر روی داده‌های آموزشی آموزش داده شده و بر روی داده‌های آزمون ارزیابی می‌شود. برای آموزش مدل از هر دو پیاده‌سازی سوال یک و پیاده‌سازی DropEdge بدون حذف هیچ یالی استفاده شده است.

هایپرپارامترهای تمامی مدل‌ها مشابه با مدل‌های قسمت قبل تنظیم شده‌اند تا امکان مقایسه دقیق بین تمام این مدل‌ها وجود داشته باشد.

مجموعه داده	تعداد لایه‌ها	ابعاد لایه پنهان	تعداد epochها	نرخ یادگیری	dropout
CoraFull	۸	۲۵۶	۲۰۰	۰.۰۰۱	۰.۲
CiteSeer	۸	۶۴	۵۰۰	۰.۰۰۹	۰.۲

نتایج دقت‌های به دست آمده از دو مدل گفته شده (دو سطر آخر) به همراه نتایج به دست آمده از تاثیر DropEdge که در قسمت قبل به دست آمده (سه سطر اول) را در جدول زیر می‌بینیم:

	CoraFull	CiteSeer
GCN پیاده‌سازی شده در سوال یک	۰.۴۷۹۹	۰.۵۶۲۹
DropEdge GCN, Edge sampling=1	۰.۶۳۳۵	۰.۶۸۸۶
DropEdge GCN, Edge sampling=0.05	۰.۶۸۸۵	۰.۷۳۶۵
GCN پیاده‌سازی شده در سوال یک با skip-connection	۰.۶۹۸۸	۰.۷۲۳۱
DropEdge GCN, Edge sampling=1, Using skip-connection	۰.۶۶۷۹	۰.۷۰۸۱

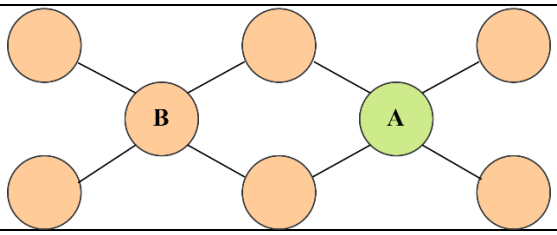
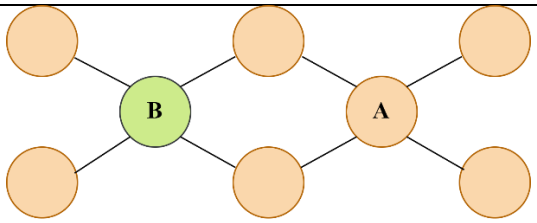
نتایج به دست آمده نشان می‌دهد که استفاده از skip-connection توانسته تا حدی مشکل کاهش دقت در اثر افزایش تعداد لایه‌های مدل را حل کند. همچنین دیده می‌شود که استفاده از skip-connection در پیاده‌سازی DropEdge بدون حذف هیچ یالی نسبت به حذف تصادفی یال‌ها تاثیر کمتری دارد و جبران دقت مدل در حالت اول کمتر بوده است. علاوه بر این استفاده از روش skip-connection در مدل پیاده‌سازی شده در سوال یک توانسته دقت این مدل را بهبود چشم‌گیری دهد و در داده‌های CoraFull تاثیر بیشتری از روش DropEdge داشته باشد ولی در داده‌های CiteSeer با وجود بهبود نتایج نتوانسته از روش DropEdge بهتر عمل کند. با مقایسه

کلی نتایج این دو روش می‌توان گفت روش DropEdge از روش skip-connection ماثرتر بوده و مشکل over-smoothing را با عملکرد بهتری برطرف می‌کند.

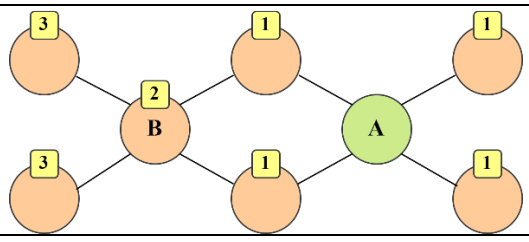
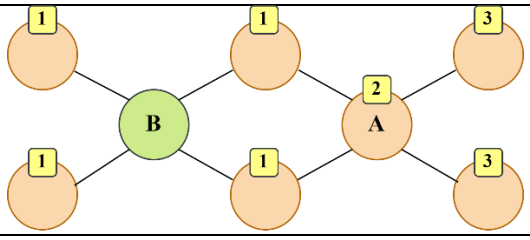
ط) جهت برطرف کردن مشکل over-smoothing ایده‌ای پیاده‌سازی شده است که در ادامه این ایده و نتایج آن را مطرح می‌کنیم.

همان طور که می‌دانیم دلیل over-smoothing تاثیرگذاشتن گره‌های با فاصله بیشتر از گره فعلی در اثر افزایش تعداد لایه‌های مدل است که باعث می‌شود تعداد زیادی از گره‌های موثر در تولید تعبیه‌ی یک گره با گره‌های دیگر مشترک باشند و تعبیه آن‌ها مشابه شود. یک راه‌کار برای حل این مشکل می‌تواند آن باشد که گره‌های با فاصله‌ی L از یک گره در تاثیرگذاری خود فاصله تا گره مورد نظر را هم لحاظ کنند.

به عنوان مثال گراف زیر را در نظر می‌گیریم. اگر بخواهیم با یک مدل سه لایه برای گره‌های A و B بردار تعبیه بسازیم تعبیه‌ی تمام گره‌های گراف در تولید تعبیه‌ی این دو گره استفاده شده و تعبیه آن‌ها مشابه خواهند شد.

تولید بردار تعبیه‌ی گره A در روش عادی	تولید بردار تعبیه‌ی گره B در روش عادی
	

ولی اگر در تولید بردار تعبیه‌ی هر گره بردار تعبیه‌ی گره‌های با فاصله‌ی L را به همراه برچسب L برای آن استفاده کنیم، تاثیر گره‌های گراف در تولید بردار تعبیه‌ی گره‌های A و B متفاوت بوده و مشابه با شکل زیر هر گره بردارهای مختلف با برچسب‌های مختلفی را در تولید بردار تعبیه خود استفاده می‌کند.

تولید بردار تعبیه‌ی گره A در ایده جدید	تولید بردار تعبیه‌ی گره B در ایده جدید
	

برای پیاده‌سازی این ایده مدل جدیدی در فایل model.py قرار گرفته است که کلیت آن مانند مدل GCN است ولی پس از هر لایه برچسبی که شماره آن لایه است با خروجی آن لایه concatenate کرده و با یک لایه خطی به ابعاد قبلی برمی‌گرداند.

جهت مقایسه مناسب نتایج این مدل با نتایج قبلی هایپرپارامترهای این مدل هم همانند آن‌ها تنظیم شده است که در قسمت قبلی این هایپرپارامترها بیان شد.

برای آموزش مدل از داده‌های آموزشی استفاده شده و نتایج ارزیابی نهایی بر روی داده‌های آزمون به دست آمده است که این نتایج به صورت زیر هستند:

	CoraFull	CiteSeer
GCN پیاده‌سازی شده در سوال یک	۰.۴۶۴۱	۰.۵۸۳۸
DropEdge GCN, Edge sampling=1	۰.۶۳۳۵	۰.۶۸۸۶
DropEdge GCN, Edge sampling=0.05	۰.۶۸۸۵	۰.۷۳۶۵
GCN پیاده‌سازی شده در سوال یک با skip-connection	۰.۶۹۵۳	۰.۶۹۹۱
DropEdge GCN, Edge sampling=1, Using skip-connection	۰.۶۶۷۹	۰.۷۰۸۱
GCN پیاده‌سازی شده در سوال یک با ایده جدید	۰.۷۰۶۶	۰.۷۱۲۶
DropEdge GCN, Edge sampling=1, Using new idea	۰.۶۳۳۵	۰.۷۰۵۰

نتایج به دست آمده نشان می‌دهد که ایده مطرح شده توانسته است مدل ۸ لایه ساده را بهبود قابل قبولی بدهد. به طوری که مدل پیاده‌سازی شده در سوال ۱ با استفاده از ایده مطرح شده توانسته دقت را در داده‌های CoraFull از ۴۶ درصد به ۷۰ درصد و در داده‌های CiteSeer از ۵۸ درصد به ۷۱ درصد برساند. همچنین پیاده‌سازی DropEdge بدون حذف هیچ یالی و تنها با استفاده از ایده جدید مطرح شده در این قسمت، توانسته دقت بر روی داده‌های CiteSeer را از ۶۸ درصد به ۷۰ درصد برساند ولی بر روی داده‌های CoraFull تاثیری نداشته است. به طور کلی این نتایج نشان می‌دهد ایده ارائه شده جهت حل مشکل over-smoothing می‌تواند تاثیر مفیدی داشته باشد. با مقایسه این نتایج با نتایج به دست آمده از روش DropEdge و skip-connection دیده می‌شود که در اثر

استفاده از پیاده‌سازی انجام شده در سوال یک، ایده جدید مطرح شده از skip-connection نتایج بهتری کسب کرده است ولی در صورت استفاده از پیاده‌سازی مدل DropEdge برای این ایده تاثیر این روش کاهش یافته است که دلیل این موضوع می‌تواند استفاده از مورادی همچون نرمال‌سازی‌ها بر روی ماتریس مجاورت گراف دانست که می‌تواند عملکرد این روش را تحت تاثیر قرار دهد.