**Simplifying Graph Convolutional Networks   
——简化的图卷积网络GCN（SGC）**[**https://arxiv.org/abs/1902.07153v1**](https://arxiv.org/abs/1902.07153v1)

来源：ICML 2019

作者：Felix Wu 张天一 Amauri Holanda de Souza Jr. Christopher Fifty Tao Yu

Kilian Q. Weinberger

# 1、SGC提出的背景

传统的机器学习方法的复杂度变化趋势都是从简单到复杂。GCN也是源于传统的机器学习方法，继承了这个复杂度的变化。此文的目的就是要把非线性的GCN转化成一个简单的线性模型SGC，通过反复消除GCN层之间的非线性并将得到的函数折叠成一个线性变换来减少GCNs的额外复杂度。

SGC中的特征提取等价于在每个特征的维度上应用了单个固定的过滤器。

# 2、Simple Graph Convolution 简化的图卷积

## 2．1、符号定义

图：G=(V,A)

A∈Rn×n 是对称的邻接矩阵

V是图的节点集

D=diag(d1,…,dn)代表度矩阵，di=∑jaij

yi∈{0,1}C 表示C维的节点one-hot标签

数据集中只知道部分节点的标签，目标是预测未知的节点的标签。

## 2．2、图卷积网络GCN vs多层感知器MLP

GCNs和MLPs相似，都是通过多层网络学习一个节点的特征向量Xi ，然后再把这个学到的特征向量送入的一个线性分类器中进行分类任务。一个k层GCN与k层MLP在应用于图中每个节点的特征向量Xi相同的情况，不同之处在于每个节点的隐藏表示在每一层的输入时是取它邻居的平均值。

每个图中的卷积层和节点表示都是使用三个策略来更新：

特征传播、线性转换、逐点非线性激活

## 2．3、SGC框架：

图一 GCN版本的示意图 SGC

GCN在K层中反复变换特征向量，然后在最终表示上应用线性分类器。

SGC将整个过程简化为简单的特征传播步骤，然后进行标准逻辑回归。

### 2.3.1、Feature propagation 特征传播

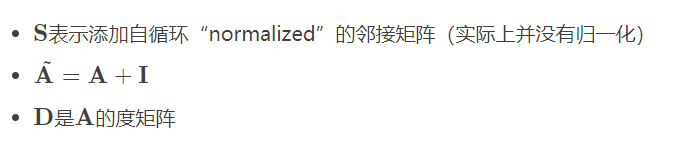
GCN的特征传播是区别MLP的，因为每一层的输入都是节点局部邻居的平均值：

di表示节点的度，hi表示节点特征，

normalization.py

def aug\_normalized\_adjacency(adj):  
 adj = adj + sp.eye(adj.shape[0])  
 adj = sp.coo\_matrix(adj)  
 row\_sum = np.array(adj.sum(1))  
 d\_inv\_sqrt = np.power(row\_sum, -0.5).flatten()  
 d\_inv\_sqrt[np.isinf(d\_inv\_sqrt)] = 0.  
 d\_mat\_inv\_sqrt = sp.diags(d\_inv\_sqrt)  
 return d\_mat\_inv\_sqrt.dot(adj).dot(d\_mat\_inv\_sqrt).tocoo()

用一个简单的矩阵运算来表示公式（2）的更新：



def fetch\_normalization(type):  
 switcher = {  
 'AugNormAdj': aug\_normalized\_adjacency, # A' = (D + I)^-1/2 \* ( A + I ) \* (D + I)^-1/2  
 }  
 func = switcher.get(type, lambda: "Invalid normalization technique.")  
 return func

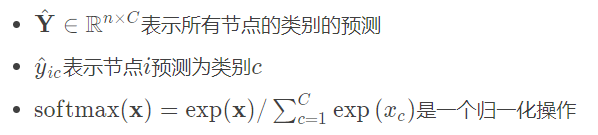
用公式（2）对所有节点进行同时更新，得到了一个简单的稀疏矩阵乘法：

这一步平滑了沿着图的边的局部隐藏表示，并最终支持在局部连接的节点之间进行类似的预测。

在局部平滑之后，一个GCN层就等于一个标准的MLP。每一个层对应一个可学习的权重矩阵Θ(k) ，所以平滑处理了的隐藏特征表示H¯(k)是线性转换的（后面乘一个参数矩阵是线性的）。最后在每个节点应用一个非线性激活函数，例如ReLU就可以得到输出的特征表示H(k)：

### 2.3.2、分类器

对于节点分类任务，最后一层和MLP相似，都是使用一个softmax分类器预测节点的标签，一个K层的GCN的所有节点的类别预测可以写作：

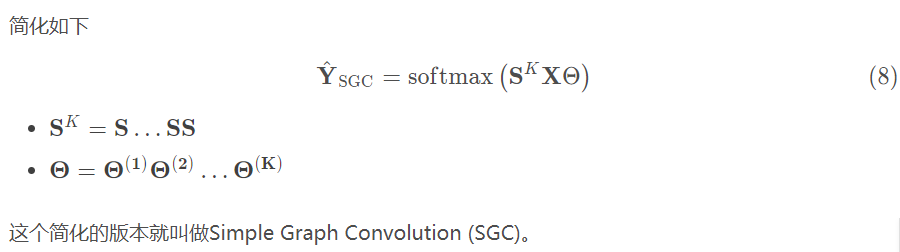


# 3、简化的图卷积SGC

在传统的MLP中，层数变深加强了网络的表达能力，因为它允许创建特征的层次结构，例如，第二层的特征构建在第一层特征的基础上。在GCNs中，每一层都有一个重要的函数:在每一层中，隐藏的表示在1跳距离的邻居之间求平均值。这意味着在k层之后，一个节点从图中所有k跳的节点处获得特征信息。这种效果类似于卷积神经网络，深度增加了内部特征的感受野[。](https://zhuanlan.zhihu.com/p/40267131)虽然卷积网络可以在层数加深时提升性能，但通常MLP的深度只限为3至4层。

## 3.1、线性化

假设GCN层之间的非线性不是最关键的，最关键的是局部邻居的平均聚合操作。因此，考虑删除每层之间的非线性转换函数（如ReLU），只保留最终的softmax(以获得概率输出)。得到的模型是线性的



def row\_normalize(mx):  
 rowsum = np.array(mx.sum(1))  
 r\_inv = np.power(rowsum, -1).flatten()  
 r\_inv[np.isinf(r\_inv)] = 0.  
 r\_mat\_inv = sp.diags(r\_inv)  
 mx = r\_mat\_inv.dot(mx)  
 return mx

## 3.2、逻辑回归

上述公式给了SGC的一个自然直观的解释：SGC由两部分组成一个固定的（没有参数）的特征提取器（或平滑器smoothing component）：X¯=SKX

特征提取器后是一个线性逻辑回归分类器Yˆ=softmax(X¯Θ)

可以看出，由于计算X¯ 不需要权值，因此可以把这部分计算作为特征的预处理步骤，整个模型的训练可以直接简化为对预处理特征X¯ 的多类逻辑回归。

## 3.3、优化细节

逻辑回归的训练是一个凸优化问题，可以用任何有效的二阶方法或随机梯度下降法进行执行。在图连通模式足够稀疏的情况下，SGD可以很自然地运用在非常大的图上，SGC的训练比GCN快得多。

#回归训练  
def train\_regression(model,  
 train\_features, train\_labels,  
 val\_features, val\_labels,  
 epochs=args.epochs, weight\_decay=args.weight\_decay,  
 lr=args.lr, dropout=args.dropout):  
 #优化器  
 optimizer = optim.Adam(model.parameters(), lr=lr,  
 weight\_decay=weight\_decay)  
 t = perf\_counter()  
 for epoch in range(epochs):  
 model.train()  
 optimizer.zero\_grad()  
 output = model(train\_features)  
 loss\_train = F.cross\_entropy(output, train\_labels)  
 loss\_train.backward()  
 optimizer.step()  
 train\_time = perf\_counter()-t  
  
 with torch.no\_grad():  
 model.eval()  
 output = model(val\_features)  
 acc\_val = accuracy(output, val\_labels)  
 return model, acc\_val, train\_time  
#回归测试  
def test\_regression(model, test\_features, test\_labels):  
 model.eval()  
 return accuracy(model(test\_features), test\_labels)

# 4、实验

## 4.1、数据集：

Cora、Citeseer、Pubmed、Reddit

### 4.1.1、Cora的数据集

包括2708份科学出版物，分为7类。引文网络由5429个链接组成。数据集中的每个发布都由一个0/1值的单词向量来描述，该向量表示字典中相应单词的存在或不存在。这部词典由1433个独特的单词组成

### 4.1.2、CiteSeer文献分类

CiteSeer数据集包括3312种科学出版物，分为6类。引文网络由4732条链接组成。数据集中的每个发布都由一个0/1值的单词向量来描述，该向量表示字典中相应单词的存在或不存在。该词典由3703个独特的单词组成。

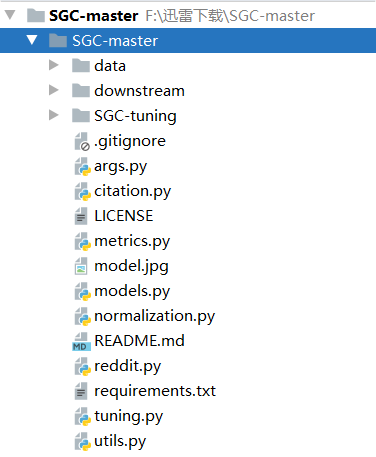
### 4.1.3、CiteSeer for Entity Resolution

为了实体解析，CiteSeer数据集包含1504个机器学习文档，其中2892个作者引用了165个作者实体。对于这个数据集，唯一可用的属性信息是作者名。完整的姓是给出的，在某些情况下，作者的全名和中间名是给出的，其他时候只给出首字母。

### 4.1.4、PubMed糖尿病数据库

由来自PubMed数据库的19717篇与糖尿病相关的科学出版物组成，分为三类。引文网络由44338个链接组成。数据集中的每个出版物都由一个TF/IDF加权词向量来描述，这个词向量来自一个包含500个唯一单词的字典。数据集中的自述文件提供了更多的细节。

## 4.2、代码实现<https://github.com/Tiiiger/SGC>



Data：数据集Cora、Citeseer、Pubmed、Reddit

Downstream：下游任务Text classification

SGC-tuning:本论文实验

Args.py

添加工具项并将结果返回

def get\_citation\_args()：

parser = argparse.ArgumentParser()

**seed**:随机数；**weight\_decay**：权重削减；dataset:数据集；

model:模型；feature:特征

Metrics.py

评测指标计算

def accuracy(output, labels):

def f1(output, labels):

Model.py

逻辑回归的简单PyTorch实现。  
      假设特征已通过k步图传播进行了预处理。

输入：节点特征，层次  
W:获得的模型  
class SGC(nn.Module):  
 def \_\_init\_\_(self, nfeat, nclass):  
 super(SGC, self).\_\_init\_\_()  
  
 self.W = nn.Linear(nfeat, nclass)

前馈运算迭代  
 def forward(self, x):  
 return self.W(x)

图卷积层（GCN）

class GraphConvolution(Module):

一个两层的GCN模型  
class GCN(nn.Module):

对模型的初始化  
def get\_model(model\_opt, nfeat, nclass, nhid=0, dropout=0, cuda=True):  
 if model\_opt == **"GCN"**:  
 model = GCN(nfeat=nfeat,  
 nhid=nhid,  
 nclass=nclass,  
 dropout=dropout)  
 elif model\_opt == **"SGC"**:  
 model = SGC(nfeat=nfeat,  
 nclass=nclass)  
 else:  
 raise NotImplementedError(**'model:{} is not implemented!'**.format(model\_opt))  
  
 if cuda: model.cuda()  
 return model

Utils.py

分别获取数据集citation，**cora**，Reddit，并对数据集进行预处理

Normalization.py

#更新输入  
def aug\_normalized\_adjacency(adj):

#矩阵S的运算表示公式2的更新  
def fetch\_normalization(type):

行规范化稀疏矩阵  
def row\_normalize(mx):

Citation.py

设置随机参数

set\_seed(args.seed, args.cuda)  
获取数据值  
adj, features, labels, idx\_train, idx\_val, idx\_test = load\_citation(args.dataset, args.normalization, args.cuda)  
调用模型

model = get\_model(args.model, features.size(1), labels.max().item()+1, args.hidden, args.dropout, args.cuda)  
if args.model == **"SGC"**: features, precompute\_time = sgc\_precompute(features, adj, args.degree)  
print(**"{:.4f}s"**.format(precompute\_time))

#回归训练  
def train\_regression(model,  
 train\_features, train\_labels,  
 val\_features, val\_labels,  
 epochs=args.epochs, weight\_decay=args.weight\_decay,  
 lr=args.lr, dropout=args.dropout):  
  
 #优化器  
 optimizer = optim.Adam(model.parameters(), lr=lr,  
 weight\_decay=weight\_decay)  
 t = perf\_counter()  
 for epoch in range(epochs):  
 model.train()  
 optimizer.zero\_grad()  
 output = model(train\_features)  
 loss\_train = F.cross\_entropy(output, train\_labels)  
 loss\_train.backward()  
 optimizer.step()  
 train\_time = perf\_counter()-t  
  
 with torch.no\_grad():  
 model.eval()  
 output = model(val\_features)  
 acc\_val = accuracy(output, val\_labels)  
  
 return model, acc\_val, train\_time

Tuning.py:优化器的函数实现

# 超参数优化  
space = {**'weight\_decay'** : hp.loguniform(**'weight\_decay'**, log(1e-10), log(1e-4))}  
  
adj, features, labels, idx\_train, idx\_val, idx\_test = load\_citation(args.dataset, args.normalization, args.cuda)  
if args.model == **"SGC"**: features, precompute\_time = sgc\_precompute(features, adj, args.degree)

#随机梯度下降法  
def sgc\_objective(space):  
 model = get\_model(args.model, features.size(1), labels.max().item()+1, args.hidden, args.dropout, args.cuda)  
 model, acc\_val, \_ = train\_regression(model, features[idx\_train], labels[idx\_train], features[idx\_val], labels[idx\_val],  
 args.epochs, space[**'weight\_decay'**], args.lr, args.dropout)  
 print(**'weight decay: {:.2e} '**.format(space[**'weight\_decay'**]) + **'accuracy: {:.4f}'**.format(acc\_val))  
 return {**'loss'**: -acc\_val, **'status'**: STATUS\_OK}  
  
best = fmin(sgc\_objective, space=space, algo=tpe.suggest, max\_evals=60)  
print(**"Best weight decay: {:.2e}"**.format(best[**"weight\_decay"**]))  
  
os.makedirs(**"./{}-tuning"**.format(args.model), exist\_ok=True)  
path = **'{}-tuning/{}.txt'**.format(args.model, args.dataset)  
with open(path, **'wb'**) as f: pkl.dump(best, f)

返回到citation.py  
#回归测试  
def test\_regression(model, test\_features, test\_labels):  
 model.eval()  
 return accuracy(model(test\_features), test\_labels)

#调用模型进行训练：测试准确性，测试精度，预处理时间，训练时间，总得时间  
if args.model == **"SGC"**:  
 model, acc\_val, train\_time = train\_regression(model, features[idx\_train], labels[idx\_train], features[idx\_val], labels[idx\_val],  
 args.epochs, args.weight\_decay, args.lr, args.dropout)  
 acc\_test = test\_regression(model, features[idx\_test], labels[idx\_test])  
  
#输出实验结果：  
print(**"Validation Accuracy: {:.4f} Test Accuracy: {:.4f}"**.format(acc\_val, acc\_test))  
print(**"Pre-compute time: {:.4f}s, train time: {:.4f}s, total: {:.4f}s"**.format(pre

Reddit.py:

在Reddit社交网络数据集上的实验, 测得的F1测度

## 4.3、实验结果：

表1.在引用网络上运行10次的平均测试精确度（％）。由于存在高方差，因此在计算其离散值时，我们删除了异常值（准确性<75/65/75％）

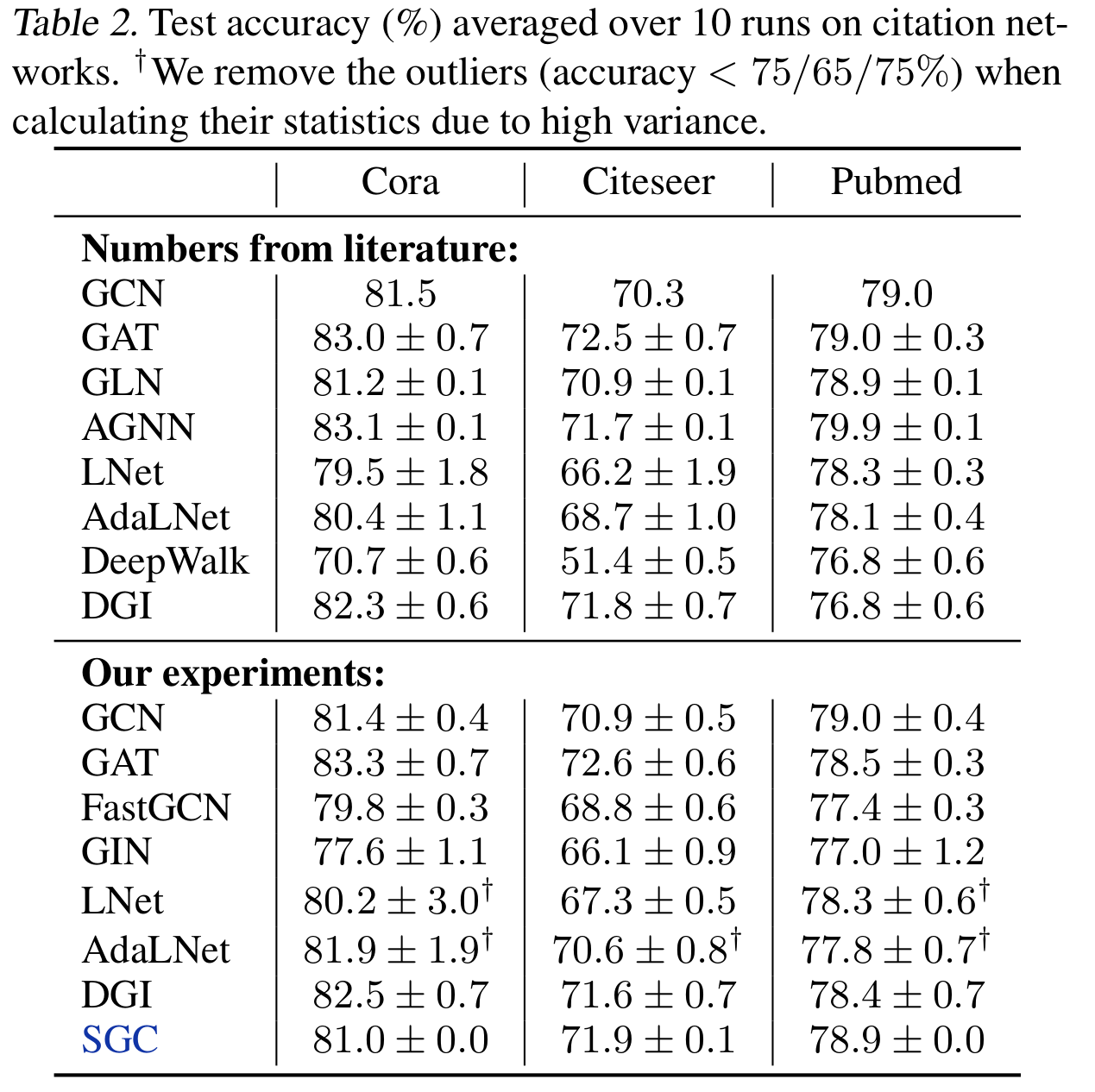


表1中，可以看出SGC和GCN的性能相当，并且在Citeseer数据集上比GCN高1%。性能提升的原因可能是由于SGC的参数更少，减少了GCN中的过拟合情况

GIN中就有严重的过拟合

LNet and AdaLNet在引文网络上不稳定

表2.在Reddit数据集上进行10次测试时测得的F1测度（％）， 模型的性能引用自其原始论文。

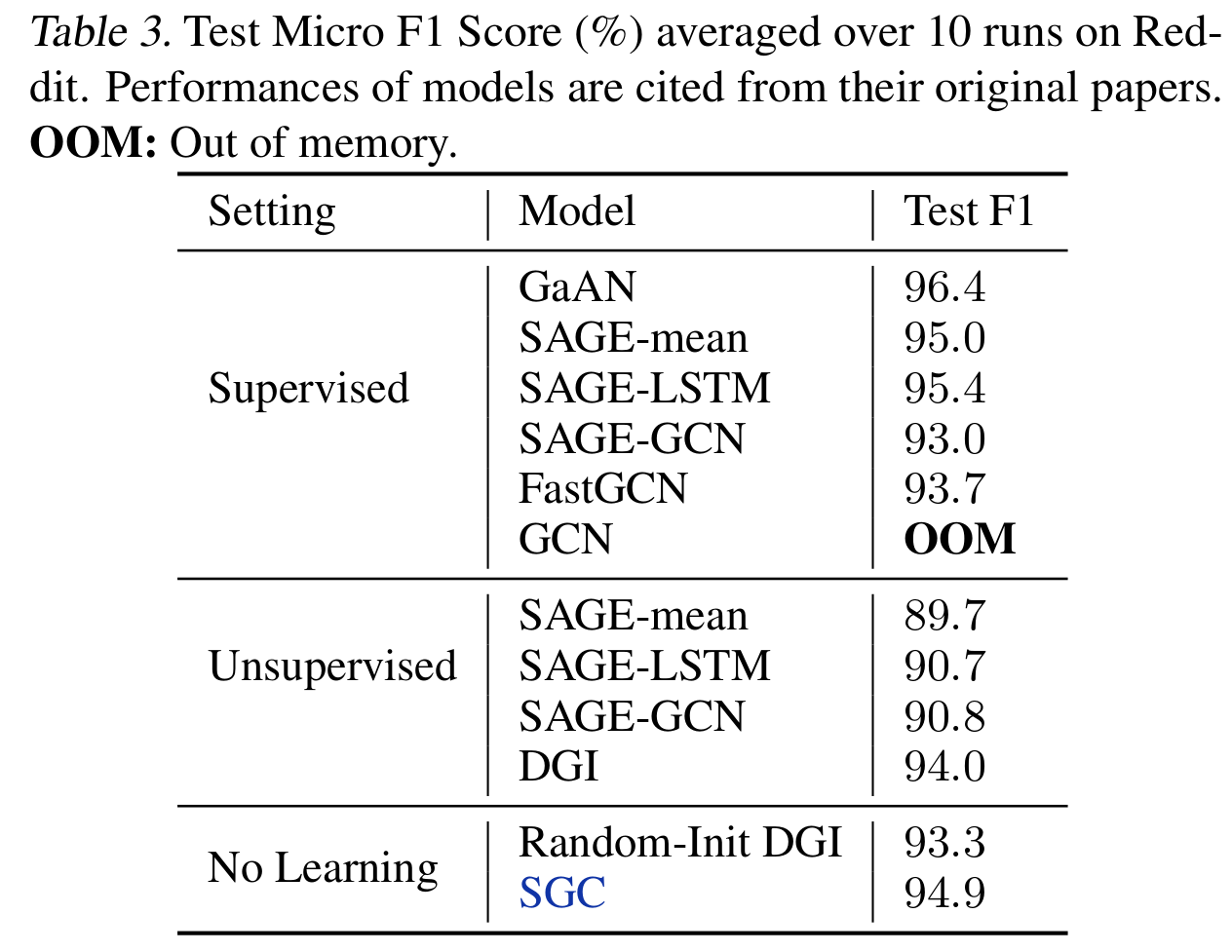


表2是在Reddit社交网络数据集上的实验

SGC比基于GCN的变种GraphSAGE和FastGCN高1%

DGI论文中表示随机初始化的DGI编码器的性能几乎与经过训练的编码器的性能相匹配;然而，这两种模型在Reddit上的表现都不如SGC。这个结果可能表明，额外的权重和非线性激活在DGI编码器中是多余的

# 5、下游任务

使用5个下游任务来研究SGC的适应性，表明这些简化不会对许多下游应用的准确性产生负面影响

* text classification
* semi-supervised user geolocation
* relation extraction
* zero-shot image classification
* graph classification

Text classification (Graph convolutional networks for text classification，2019)使用2层的GCN来实现了一个state-of-the-art的结果，创建了一个语料库图，该图将文档和单词都视为图中的节点。Word-word边的权值为点信息的互信息(point twise mutual information, PMI)，word-document边的权值为标准化的TF-IDF socre。

结论：一个SGC (K = 2)在5个基准数据集上与他们的模型竞争，同时达到了83.6倍的速度。

Semi-supervised user geolocation（半监督用户地理定位)根据用户发布的帖子、用户之间的关系以及少数被标记的用户，来定位用户在社交媒体上的“家”的位置

结论：SGC在GEOTEXT、TWITTERUS和TWITTER-WORLD 的高速公路连接方面优于GCN，同时在TWITTER-WORLD上节省了30多个小时。

Relation extraction关系抽取包括预测句子中主语和宾语之间的关系 C-GCN采用LSTM，后面接着GCN和MLP层。

实验将GCN替换为SGC (K = 2)将得到的模型称为C-SGC

Zero-shot image classification（zero-shot 图像分类）包括学习一个图像分类器，不需要从测试类别中获取任何图像或标签GCNZ使用GCN将类别名称映射到图像特征域，并查找与查询图像特征向量最相似的类别

结论：使用MLP替换GCN，然后使用SGC可以提高性能，同时将参数数量减少55%为了将预先训练好的GloVe向量映射到由ResNet-50提取的视觉特征空间，需要一个MLP特征提取器同样，这个下游应用证明了学习图卷积滤波器是多余的

Graph classification（图分类）要求模型使用图结构对图进行分类（How powerful are graph neural networks?，ICLR 2019）从理论上证明了GCNs不足以区分特定的图结构，并证明了GIN更具表现力，在各种图分类数据集上获得了 state-of-the-art的结果将DCGCN中的GCN替换为SGC，分别获得NCI1和COLLAB数据集上的71.0%和76.2%，这与GCN相当，但远远落后于GIN在QM8量子化学数据集上，更高级的AdaLNet和LNet在QM8上得到0.01MAE，远远超过SGC的0.03 MAE。