UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

LUCAS GARCIA RUBIO

CLASSIFICAÇÃO DE ÁREA CIENTÍFICA DE ARTIGOS UTILIZANDO A REDE DE CITAÇÕES E GRAFOS CONVOLUCIONAIS

PATO BRANCO

2022

LUCAS GARCIA RUBIO

CLASSIFICAÇÃO DE ÁREA CIENTÍFICA DE ARTIGOS UTILIZANDO A REDE DE CITAÇÕES E GRAFOS CONVOLUCIONAIS

SCIENTIFIC AREA CLASSIFICATION OF ARTICLES USING CITATION NETWORK AND GRAPH CONVOLUTIONAL

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Dalcimar Casanova

PATO BRANCO

2022

LUCAS GARCIA RUBIO

CLASSIFICAÇÃO DE ÁREA CIENTÍFICA DE ARTIGOS UTILIZANDO A REDE DE CITAÇÕES E GRAFOS CONVOLUCIONAIS

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 28/junho/2022

Prof. Dr. Dalcimar Casanova
Doutorado em Física Computacional
Universidade Tecnológica Federal do Paraná (UTFPR) - Pato Branco

---

Prof. Dr. Marco Antonio de Castro Barbosa
Doutorado em Informática
Universidade Tecnológica Federal do Paraná (UTFPR) - Pato Branco

---

Prof. Dr. Ives Renê Venturini Pola
Pós-Doutorado em Ciências de Matemática e Computação
Universidade Tecnológica Federal do Paraná (UTFPR) - Pato Branco

PATO BRANCO

2022

Dedico este trabalho aos meus pais, Eraldo Garcia Rubio e Renata Lima Garcia Rubio, que sempre me apoiaram em todas as decisões da minha vida.

AGRADECIMENTOS

Agradeço a todos que caminharam comigo nesta jornada e me ajudaram a me preparar para as novas que virão por aí.

Agradeço a minha namorada, Maria Alice Scarmocin, por tornar meus dias mais leves e mais possíveis.

Agradeço o meu amigo Fernando Martins Ribeiro pelas madrugadas adentro estudando e "fritando"nas disciplinas mais difíceis deste curso.

Agradeço o meu amigo Felipe Arisi pelas intermináveis parcerias nos projetos da faculdade. Os resultados incríveis que obtive foram muito mais recompensadores por ter você ao lado.

Agradeço meus professores Prof. Marco Barbosa e Prof. Ives Renê pelas excelentes recomendações para este trabalho (não segui todas pois quero me livrar desse peso logo, mas foram excelentes!) e pelo exemplo que os senhores foram em sala de aula. Agradeço também em especial os professores Prof. Dalcimar Casanova e Prof. Pablo Cavalcanti pelas tardes de churrasco e por não terem me orientado somente durante minha jornada acadêmica, mas também na vida.

Por fim, agradeço infinitamente os meus pais, que me proporcionaram uma experiência de estudos incrível, sem ao menos precisar trabalhar pra isso. Me dediquei integralmente aos estudos e meus resultados são graças a vocês.

RESUMO

A busca por trabalhos científicos geralmente é feita por palavras-chaves, porém não há garantias de que estas palavras sejam exclusivas de um único tema. Essa imprecisão tende a gerar um grande volume de documentos nos processos de busca, dificultando assim o processo de pesquisa. Nesse sentido pesquisadores dirigiram esforços para incluir nesta busca as citações que estes documentos fizeram a outros documentos similares e/ou correlatos. A adição dessas informações tende a retornar documentos mais precisos e focados no tema de interesse. Do ponto de vista da computação esse problema pode ser visto como uma classificação de nós de um grafo, onde os artigos são os nós e as citações são as arestas. Uma das metodologias mais proeminentes nesta área é a convolução em grafos, inspirada no sucesso que as convoluções obtiveram com imagens e áudios. Seguindo esta linha, esse trabalho utiliza um modelo de grafo convolucional para classificar as áreas de um artigo científico utilizando como critério sua rede de citações e um contador de palavras-chaves mencionadas em seu texto. Os resultados obtidos foram satisfatórios.
Palavras-chave: grafos; gc; cora; classificação.

ABSTRACT

The search for scientific papers is usually done by keywords, but there is no guarantee that these words are exclusive to a single topic. This imprecision tends to generate a large volume of documents in the search processes, thus making the search process difficult. In this sense, researchers made efforts to include in this search the citations that these documents made to other similar and/or related documents. The addition of this information tends to retrieve documents that are more accurate and focused on the topic of interest. From a computational point of view this problem can be seen as a graph node classification, where the articles are the nodes and the citations are the edges. One of the most prominent methodologies in this area is graph convolution, inspired by the success that convolutions have had with images and audio. Following this line, this work uses a convolutional graph model to classify the areas of a scientific article using as a criterion its citation network and a counter of keywords mentioned in its text. The results obtained were satisfactory.
Keywords: graph; gcn; cora; classification.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Siglas

ACM	*Adaptive Channel Mixing*
CNN	*Convolutional Neural Network*
ConvGNN	*Convolutional Graph Neural Network*
DGL	*Deep Graph Library*
DL	*Deep Learning*
GAE	*Graph autoencoder*
GCN	*Graph Convolutional Network*
GK	*Graph Kernel*
GKM	*Graph Kernel Method*
GNN	*Graph Neural Network*
kNN	*k-Nearest Neighbors*
LPA	*Label Propagation Algorithm*
ML	*Machine Learning*
MLP	*Multi-Layer Perceptron*
RecGNNs	*Recurrent Graph Neural Network*
STGNN	*Spatial-temporal Graph Neural Network*

SUMÁRIO

1 INTRODUÇÃO

De acordo com Landhuis (2016) o número de publicações científicas tem crescido de 8 a 9% ao ano ao longo das últimas décadas. Só no campo da biomedicina, mais de um milhão de artigos são publicados na base de dados PubMed por ano (aproximadamente duas publicações por minuto). Com um número tão grande de publicações disponíveis para se pesquisar, como ser mais assertivo quando se busca por um determinado tema ou conteúdo?

Os métodos tradicionais, segundo Rahman (2013), consistem em categorizar as publicações levando em conta principalmente o título e as palavras chaves, mas não existe garantia de que uma palavra chave seja exclusiva de um único tema, podendo dificultar a busca. Sendo assim, uma boa solução seria analisar um número maior de palavras chaves, levando em conta também a quantidade de vezes que ela apareceu, e se ela também foi frequente em outros artigos do mesmo tema. Mas esse tipo de análise é custosa computacionalmente, ainda mais considerando a quantidade imensa de publicações feitas diariamente.

Uma alternativa ao uso dos títulos e as palavras chaves é classificar os artigos segundo suas citações. Esta abordagem parte do pressuposto que artigos que tratam de temas semelhantes/correlatos irão citar trabalhos similares ou próximos.

Uma forma de se modelar as citações de um artigo é através de um grafo onde os vértices são os artigos e as arestas são as citações. Ao se modelar as redes de citações de trabalhos por grafos é possível, com auxílio de algum método estatístico/inteligente, mensurar o quão semelhantes as citações de dois trabalhos são, inferindo se trata-se de dois trabalhos da mesma área (i.e. correlatos) ou não.

Nessa linha de modelagem já existem vários trabalhos na literatura devotados a essa tarefa de classificar a área do artigo a partir de suas citações. Izadi *et al.* (2020), por exemplo, atingiu uma acurácia de 90.16% ao aplicar o uso de gradientes naturais no processo de otimização da GCN. Embora este e outros autores tenham alcançado bons resultados é necessário que tais inferências sejam mais precisas.

Nessa linha diversos autores, como por exemplo Wang *et al.* (2019) e Dabhi e Parmar (2020), vem sugerindo arquiteturas de rede alternativas, demostrando que é um problema ainda em aberto.

Em uma modelagem por grafos, diferente de um problema genérico de ML, onde as amostras são independentes umas das outras, aqui cada nó representa um artigo e as arestas representam a rede de citações existente entre eles. Este problema pode ser sumarizado como aprender um mapeamento $\mathcal{M} : \mathcal{V} \rightarrow \mathcal{L}$, onde $\mathcal{V}$ é o conjunto de nós e $\mathcal{L}$ o conjunto de rótulos possíveis. De forma genérica, é possível realizar aprendizado e inferência sobre esse tipo de modelagem utilizando métodos de grafos convolucionais.

A hipótese deste trabalho é de que grafos convolucionais, mais especificamente o modelo GCN (KIPF; WELLING, 2016a), são modelos adequados para realizar classificação de artigos em áreas do conhecimento baseando-se em suas citações.

1.1 Objetivos

Classificar artigos científicos conforme sua área utilizando como critério sua rede de citações utilizando um modelo de grafo convolucional.

1.1.1 Objetivos específicos

- Buscar e analisar os impactos dos hiperparâmetros no modelo GCN.
- Comparar os resultados obtidos com trabalhos nesta mesma linha de aplicação.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo é estruturado sobre tudo com base no trabalho de Wu *et al.* (2019).

2.1 *Deep Learning*

O desenvolvimento em *deep learning* (DL) permite avanços em muitas áreas da Inteligência Artificial, tais como processamento de fala e áudio, visão computacional e processamento de linguagem natural (Begum; Fatima; Sabahath, 2019). Todo esse sucesso é possível graças a uma série de fatores, sendo alguns deles o desenvolvimento de recursos computacionais (e.g., GPU) e a grande disponibilidade de dados para treinamento (WU *et al.*, 2019). Além disso, o DL se destaca bastante dos seus algoritmos predecessores de *machine learning* (ML) por ter a habilidade de extrair automaticamente as caraterísticas dos dados. Para os antigos algoritmos de ML, era necessário técnicas exaustivas de matemática e de engenharia para processar os dados e retirar deles os importantes fatores que seriam a base de treino para os modelos de redes neurais (Begum; Fatima; Sabahath, 2019). Dentre as aplicações do *deep learning* encontra-se o *healthcare*, que realiza diagnóstico de pacientes por imagem. Temos também sistemas de recomendação de filmes, livros e músicas baseados em comportamentos, utilizados por empresa como Netflix, Amazon e Spotify. Além desses, também é comum o aparecimento de *deep learning* em sites de banco e pagamento online, como Nubank e PayPal, e em assistentes pessoais, como o Google Assistente, a Cortana e a Siri.

Entretanto, o paradigma de DL encontrou desafios quando se tratava de dados no domínio não Euclidiano. As aplicações citadas anteriormente são todas Euclidianas, i.e., podem ser vetorizadas. Um exemplo de aplicação que não se encontra nesse domínio são os grafos. Segundo Kipf e Welling (2016a), grafos são boas representações de muitos conjuntos de dados da vida real, tais como: redes sociais, grafos de conhecimento, redes de interação de proteínas, a rede mundial de computadores (WWW - *World Wide Web*), etc.. Uma outra aplicação muito comum são os grafos de cidades. Muitas empresas como Amazon utilizam algoritmos de buscas em grafos para decidir os melhores caminhos a percorrer em entrega de produtos. Sem a ajuda desses algoritmos, a empresa poderia tomar decisões que aumentariam os gastos e o tempo, insatisfazendo seus clientes. Em redes sociais, como Facebook, LinkedIn e outros, grafos são utilizados para entender as conexões entre seus usuários, dessa forma fazendo melhores recomendações de anúncios, sugestões de amizades e eventos. Para essas e outras aplicações, pesquisadores têm desenvolvido novas soluções heurísticas: as GNNs (do inglês *graph neural networks*).

Bronstein *et al.* (2017) define o termo *deep learning* geométrico como ”um termo para técnicas emergentes que tentam generalizar modelos (estruturados) de *deep learning* em espaços não Euclidianos, tal como os grafos.” Nos últimos anos, muitos pesquisadores têm contribuído com artigos na área de DL em grafos. Duvenaud *et al.* (2015) desenvolveu uma rede neu-

ral convolucional de grafos que reconhece digitais, independentemente de seu tamanho. Jain *et al.* (2015) utilizou uma rede neural recorrente para prever comportamentos de um grafo que se modifica ao longo do tempo. Outros ainda, como Bruna *et al.* (2014) e Henaff, Bruna e LeCun (2015), utilizam convoluções de grafos conhecidas a partir da Teoria de Grafos Espectrais para definir filtros parametrizados a serem utilizados em modelos de redes neurais multi-camadas, semelhantes às clássicas redes neurais convolucionais já conhecidas (KIPF; WELLING, 2016a). São todas pesquisas recentes, demonstrando o crescimento e a importância do assunto para a área de ML.

A convolução é uma das operações mais importantes dentro de ML pois, dado duas funções, ela as mistura de uma maneira específica, facilitando muitos cálculos matemáticos (COSTA, 2019). Assim, ela melhora o tempo de aprendizado de uma máquina sem perder desempenho, o que contribuiu muito para o avanço na área. Convolução em grafos, por sua vez, opera com os nós e as características de seus vizinhos, gerando uma matriz de características a serem propagados na rede neural para atualização dos pesos.

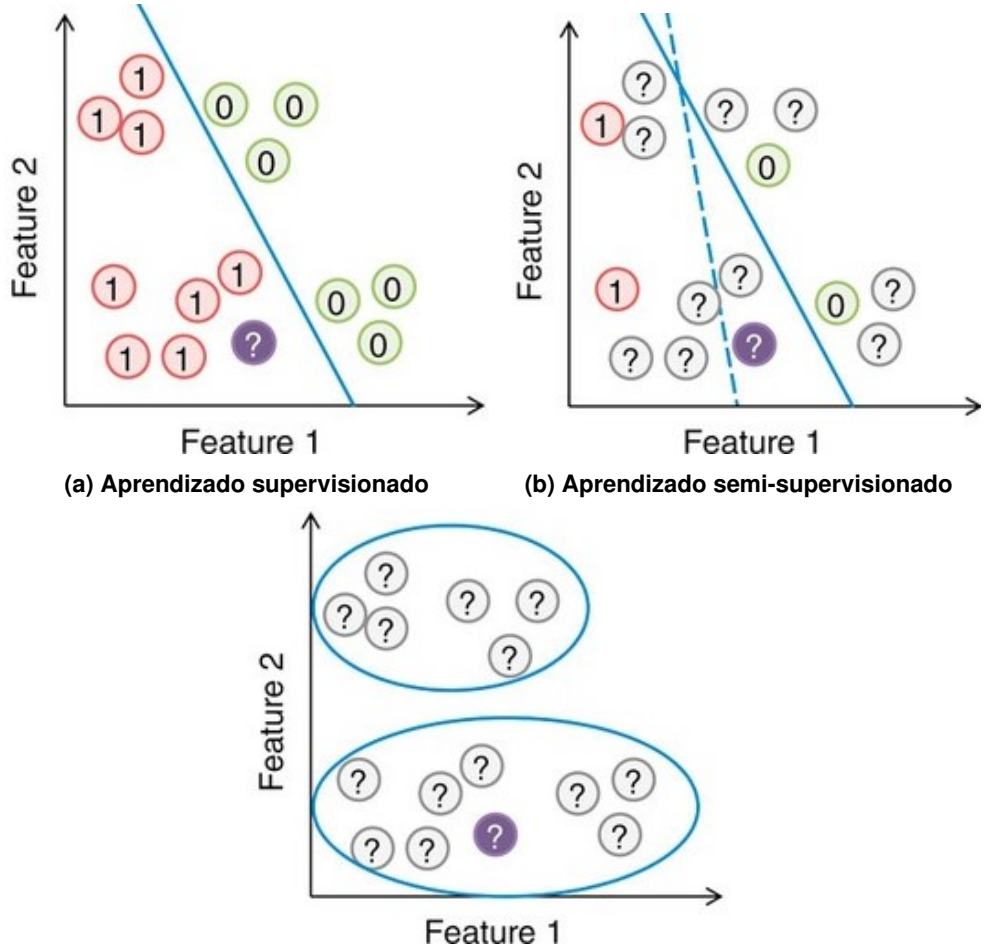
Dentre as aplicações de convolução em grafo tem-se: (1) classificação de grafos, (2) classificação de nós e (3) classificação de arestas, referidos no próximo capítulo deste texto como classificações *graph-level*, *node-level* e *edge-level*, respectivamente. Este trabalho tem por objetivo realizar a tarefa número 2, também conhecida na literatura como *node prediction*. Esta tarefa tem por objetivo analisar características dos nós em um grafo para prever seus comportamentos ou ainda agrupá-los segundo suas relações uns com os outros.

A proposta deste trabalho é utilizar a GCN (do inglês *graph convolutional network*, explicado mais tarde neste texto) para classificar os nós do grafo de uma rede de citações de artigos e publicações científicas. Nesta base de dados, a rede de citações é representado por um grafo, onde os nós são os artigos e as arestas são as citações entre estas publicações.

2.2 Aprendizado Semi-Supervisionado em Deep Learning

O termo Aprendizado Semi-Supervisionado refere-se a um tipo de problema de aprendizado que envolve uma pequena quantidade de dados rotulados e uma grande quantidade de dados não rotulados disponíveis para o algoritmo de aprendizado durante o treinamento. O entendimento torna-se mais fácil quando comparado com os métodos supervisionado e não supervisionado.

No aprendizado supervisionado todos os dados são rotulados e, em geral, quanto maior a quantidade de amostras apresentadas ao algoritmo de aprendizado, maior é sua generalização do problema. Nesse tipo de problema, o algoritmo treina somente com os dados que ele conhece o rótulo, pois somente assim ele consegue calcular o erro e realizar *backpropagation*, processo que consiste em calcular as derivadas parciais do erro com relação aos pesos de cada camada do modelo para atualizá-los. Neste aprendizado, possíveis dados com rótulos desconhecidos são desconsiderados da base de dados. Ver Figura 1(a).



(a) Aprendizado supervisionado **(b) Aprendizado semi-supervisionado**

(c) Aprendizado não supervisionado

Fonte: autoria própria.

Figura 1 – Comparação entre aprendizado supervisionado, não supervisionado e semi-supervisionado.

O aprendizado não supervisionado é um problema de caráter oposto ao do supervisionado. Nenhum dos dados de treinamento aqui possuem rótulos e o processo de aprendizado se dá por agrupamento. O algoritmo de aprendizado se encarrega de analisar as características dos dados e separá-los em conjuntos. Em alguns modelos é possível passar como parâmetro a quantidade de categorias em que se quer classificar os dados não rotulados, dizendo assim ao modelo em quantos grupos ele precisa separar o conjunto. Ver Figura 1(c).

O aprendizado semi-supervisionado é algo entre esses dois problemas. Não possui todos os dados rotulados como no supervisionado, mas também não possui todos os dados sem rótulos como no não supervisionado. Aqui, os dados rotulados servem como um indicador de como o modelo deve aprender a classificá-los. Ver Figura 1(b).

O problema com uma modelagem baseada em grafos é um caso bem representativo para um aprendizado semi-supervisionado pois, diferentemente do que acontece no supervisionado, onde pode-se simplesmente eliminar alguma amostra sem rótulo, em um grafo não se deve eliminar nós não rotulados, pois assim o grafo teria sua estrutura alterada.

2.3 Grafo

Um grafo é representado como $G = (V, E)$, onde V é o conjunto de nós e E o conjunto de arestas. Seja $v_i \in V$ a denotação de um nó e $e_{ij} = (v_i, v_j) \in E$ a denotação de uma aresta. A matriz de adjacência $\vec{A}$ é entendida como uma matriz $n \times n$ com $A_{ij} = 1$ se $e_{ij} \in E$ e $A_{ij} = 0$ se $e_{ij} \notin E$. Um grafo pode ter atributos de nós $\vec{X}$, onde $\vec{X} \in \mathbb{R}^{n \times d}$ é uma matriz de característica de nós com $\vec{x}_v \in \mathbb{R}^d$ representando o vetor de característica de um nó v. Adicionalmente, um grafo pode ter atributos de arestas $\vec{X}^e$, onde $\vec{X}^e \in \mathbb{R}^{n \times c}$ é uma matriz de características de arestas com $\vec{x}^e_{vu} \in \mathbb{R}^c$ representando o vetor de características de uma aresta (v, u).

2.3.0.1 Grafo direcionado

Um grafo direcionado é um grafo com todas as arestas apontando de um nó para o outro. Um grafo não direcionado é considerado um caso especial de grafos direcionados onde existe um par de arestas com direções inversas se dois nós estão conectados. A matriz de adjacência de um grafo direcionado pode ser assimétrico, ou seja, $\vec{A} \neq \vec{A}^T$.

2.3.0.2 Grafo espaço-temporal

Um grafo espaço-temporal é um grafo com atributos onde os nós de entradas alteram-se dinamicamente ao longo do tempo. O grafo espaço-temporal é definido como $G^{(t)} = (V, E, \vec{X}^{(t)})$ com $\vec{X}^{(t)} \in \mathbb{R}^{n \times d}$.

2.3.1 Matriz de Adjacência

Seja $G = G(V,E)$ um grafo com n vértices. A matriz de adjacência $\vec{A}$ de G é a matriz quadrada de ordem n cujas entradas são

$$a_{ij} = \begin{cases} 1, & \text{se } v_i, v_j \in E \text{ para } v_i, v_j \in V \\ 0, & \text{nos outros casos} \end{cases}$$

2.3.2 Matriz Diagonal e Matriz de Graus

Uma matriz quadrada é chamada de matriz diagonal se todas as suas entradas fora da diagonal principal são iguais a zero. No contexto de grafos, uma matriz diagonal armazena em cada posição da sua diagonal principal o grau dos vértices de um grafo, e pode ser definida como

$$\vec{D_{ii}} = \sum_{j} (\vec{A_{ij}})$$

2.3.3 Matriz Laplaciana

Seja $\vec{D}$ a matriz diagonal dos graus dos vértices de um grafo G (ou seja, a matriz $\vec{D}$ tal que $\vec{D}_{ii} = d(v_i)$) e seja a matriz $\vec{A}$ a matriz de adjacência de G. A matriz

$$L = \vec{D} - A$$

é chamada matriz Laplaciana ou Laplaciano do grafo G.

2.4 Trabalhos relacionados

Sperduti e Starita (1997) foram os primeiros a aplicar redes neurais em grafos direcionados acíclicos, o que motivou estudos sobre GNNs. A noção de GNNs começou a ser desenvolvida em Gori, Monfardini e Scarselli (2005), e depois elaborada em Scarselli *et al.* (2009) e em Gallicchio e Micheli (2010). Esses recentes estudos são categorizados como GNNs recorrentes (RecGNNs). Eles aprendem a representação de um nó propagando iterativamente as informações de seus vizinhos até alcançar uma estabilidade. Esse processo é caro computacionalmente, e recentemente tem aumentado os esforços para superar esses desafios.

Encorajados pelo sucesso das CNNs com visão computacional, um grande número de métodos que redefinem a notação de convolução para dados de grafos estão sendo desenvolvidos. Essas abordagens estão sob o conceito de GNNs convolucionais (ConvGNNs). ConvGNNs são divididos em duas principais abordagens: a espectral e a espacial. A primeira pesquisa em ConvGNNs espectrais foi apresentada em Bruna *et al.* (2014), que desenvolveu um grafo convolucional baseado na Teoria dos Grafos Espectrais. Desde então, têm aumentado as melhorias, as extensões e as aproximações em ConvGNNs espectrais. A pesquisa sobre ConvGNNs espaciais começou mais cedo que as ConvGNNs espectrais. Em 2009, Micheli et al. abordou a dependência mútua de grafos através da arquitetura composta por camadas não recursivas enquanto herdava ideias de passagem de mensagem das RecGNNs. Entretanto, a importância desse trabalho havia sido esquecida, até que recentemente muitas ConvGNNs surgiram. Separados das RecGNNs e ConvGNNs, muitas GNNs alternativas tem sido desenvolvida no últimos anos, incluindo GAEs (do inglês *graph autoencoders*) e STGNNs (do inglês *spatial-temporal graph neural networks*). Esses *frameworks* de aprendizado podem ser construídos em RecGNNs, ConvGNNs, ou outras arquiteturas neurais para modelagem de grafos. Detalhes da categorização desses métodos são dados mais adiante nesta seção.

2.4.1 GNNs VS *Network Embedding*

A pesquisa sobre GNNs está relacionada a *graph embedding* ou *network embedding*, outro tópico que tem atraído atenção das comunidades de *data mining* e *machine learning*. *Network embedding* visa representar os nós de uma rede como um vetor de dimensão reduzida, preservando a topologia da rede e as informações de cada nó, para que tarefas analíticas subsequentes de grafos, tais como classificação e *clustering* possam ser facilmente realizadas usando algoritmos simples de ML pronto para o uso. Enquanto isso, GNNs são modelos de DL que visam realizar tarefas relacionadas a grafos de uma maneira *end-to-end*. Muitas GNNs extraem explicitamente representações de alto nível. A principal diferença entre GNNs e *network embedding* é que GNNs são um grupo de modelos de redes neurais que são designadas para várias tarefas, enquanto que *network embedding* cobre vários tipos de métodos que visam uma mesma tarefa. Assim sendo, GNNs podem resolver o problema das *networks embeddings* através de um *framework* de GAE. Por outro lado, *network embedding* contém outros métodos que não utilizam DL, tal como fatorização de matrizes (SHEN *et al.*, 2018) e *random walks* (PEROZZI; AL-RFOU'; SKIENA, 2014).

2.4.2 GNNs VS GKM (do inglês *graph kernel method*)

Graph Kernels (GKs) são técnicas historicamente dominantes para resolver o problema de classificação de grafos (VISHWANATHAN *et al.*, 2008) (SHERVASHIDZE *et al.*, 2011). Esses métodos empregam uma função *kernel* para medir a similaridade entre pares de grafos, o que significa que esses algoritmos, como *support vector machines*, podem ser utilizados para aprendizados. Similar a GNNs, GKs podem transformar grafos ou nós em um vetor de espaços através de uma função de mapeamento. A diferença é que essa função de mapeamento é determinística ao invés de aprendível. Devido ao cálculo de semelhança em pares, GKMs sofrem significantemente com gargalos computacionais. GNNs, por outro lado, realizam classificação de grafos baseado na extração de suas representações e por isso são muito mais eficientes que GKMs.

2.4.3 Trabalhos na base de dados Cora

Dos trabalhos que tentaram resolver o problema do Cora Dataset, pode-se citar Wang e Leskovec (2020) que utilizam um algoritmo de propagação de rótulos, conhecido como LPA (*Label Propagation Algorithm* ((WANG; ZHANG, 2008))) como complementação da GCN, introduzido por Kipf e Welling (2016a). Outro que propõe melhorias no uso da GCN é Sun, Zhu e Lin (2021), introduzindo a AdaGCN, que possui a habilidade de extrair informações de forma eficaz dos nós de um grafo, agrupá-los e atualizar os pesos do modelo de forma interativa, graças a ar-

quitetura AdaBoost acoplado na GCN. Um outro trabalho interessante e que não utiliza a GCN é Dabhi e Parmar (2020), que propõe o uso da NodeNet para resolver problemas de classificação em um grafo. O NodeNet, por sua vez, propõe melhorias em pré-processamento, arquitetura de rede e regularização para o já conhecido trabalho Bui, Ravi e Ramavajjala (2018). Um dos trabalhos mais proeminentes é Luan *et al.* (2021), que propõe o uso de Adaptive Channel Mixing (ACM) GNNs para classificação em grafos heterogêneos e alcança uma acurácia de 82.07% +- 1.04% na base de dados Cora. Entretanto, atualmente o trabalho que alcançou melhores resultados foi Izadi *et al.* (2020), atingindo uma acurácia de 90.16% +- 0.59%, ao aplicar o uso de gradientes naturais no processo de otimização da GCN. Por outro lado, dentre as menores acurácias de uma tabela top 65 publicado pelo Papers With Code, encontra-se Abu-El-Haija *et al.* (2018), que atingiu 67.9% utilizando o modelo AttentionWalk e Tran (2018), que atingiu 78.3% utilizando uma arquitetura Autoencoder.

2.5 Categorização e tipos de problemas

Nesta seção, é apresentado uma taxonomia de GNNs em recorrentes (RecGNNs), GNNs convolucionais (ConvGNNs), grafos auto-codificadores (GAEs) e GNNs espaço-temporais (STGNNs).

2.5.1 Taxonomia de GNNs

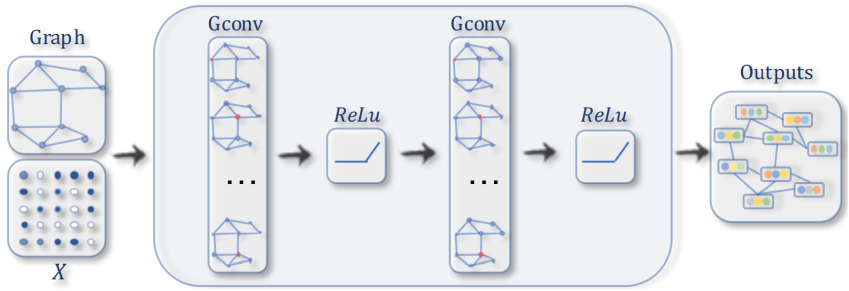
GNNs recorrentes (RecGNNs) são pioneiras nos trabalhos de GNNs. RecGNNs visam o aprendizado das representações dos nós através de arquiteturas neurais recorrentes. Elas assumem que um nó em um grafo troca informações/mensagens constantemente com seus vizinhos até que um equilíbrio estável seja alcançado. RecGNNs são conceitualmente importantes e inspiraram pesquisas futuras sobre ConvGNNs. Particularmente, a ideia de propagação de informação (ou passagem de mensagem) é, mais tarde, herdada por ConvGNNs espaciais.

GNNs convolucionais (ConvGNNs) generalizam a operação de convolução de dados matriciais para dados de grafos. A principal ideia é gerar a representação de um nó v agregando suas próprias características x_v e as características de seus vizinhos x_u, onde $u \in N(v)$. Diferentemente das RecGNNs, as ConvGNNs empilham várias camadas convolucionais de grafos para extrair representações alto nível dos nós. As ConvGNNs desempenham um papel importante na construção de muitos outros modelos complexos de GNNs. A Figura 1 mostra uma ConvGNN para classificação de nó. A Figura 2 mostra uma ConvGNN para classificação de grafo. Uma seção está reservada para discutir melhor esse modelo, uma vez que será o foco deste trabalho.

Grafos auto-codificadores (GAEs) são *frameworks* de aprendizado não supervisionado que codificam nós ou grafos em um vetor latente de espaço e reconstrói dados de grafos a

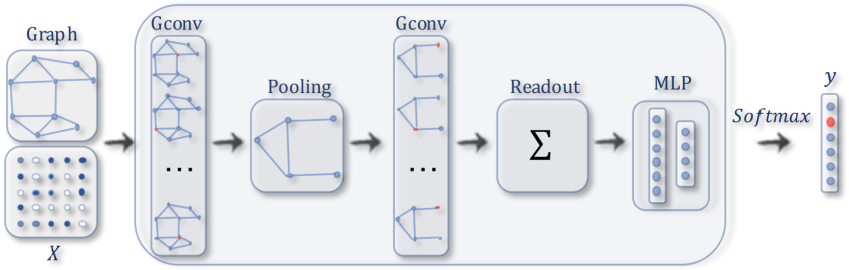
partir da informação codificada. GAEs são utilizadas para aprender *network embeddings* e distribuições generativas de grafos. Para *network embeddings*, GAEs aprendem representações latentes de nós através da reconstrução das informações estruturais dos grafos, tal como uma matriz de adjacência. Para a geração de grafos, alguns métodos geram passo-a-passo nós e arestas de um grafo enquanto que outros métodos geram um grafo inteiro de uma só vez. A Figura 3 mostra uma GAE.

GNNs espaço-temporais (STGNNs) visam aprender padrões ocultos de grafos espaço-temporais, o que tem aumentado a importância em uma variedade de aplicações tal como a previsão da velocidade de tráfego (LI *et al.*, 2018b), antecipação de manobras de um motorista (JAIN *et al.*, 2016) e reconhecimento de ações humanas (YAN; XIONG; LIN, 2018). A ideia chave das STGNNs é considerar a dependência espacial e a dependência temporal ao mesmo tempo. Muitas abordagens atuais integram convoluções de grafos para capturar dependências espaciais com RNNs ou CNNs para modelar as dependências temporais. Um exemplo de STGNN é mostrado na Figura 4.



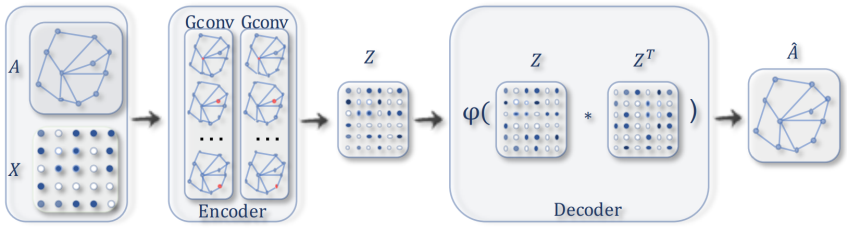
Fonte: Wu *et al.* (2019).

Figura 2 – Uma ConvGNN com múltiplas camadas convolucionais de grafos. Uma camada convolucional de grafo encapsula cada representação *hidden* do nó agregando informações de seus vizinhos. Depois, uma transformação não linear é aplicada aos resultados para produzir uma saída.



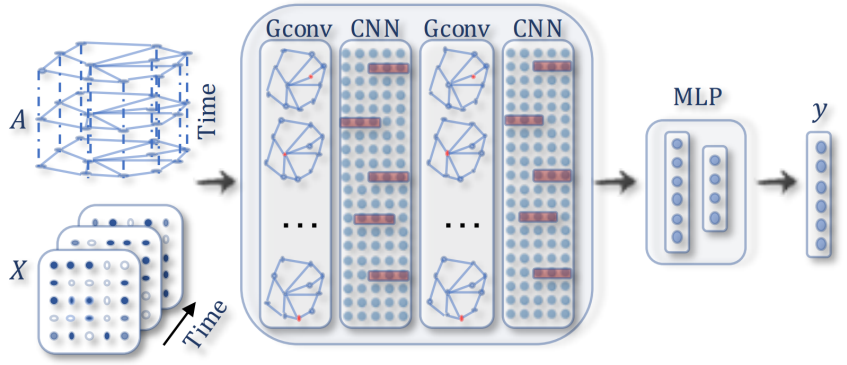
Fonte: Wu *et al.* (2019).

Figura 3 – Uma ConvGNN com *pooling* e camadas *readout* para classificação de grafos. Uma camada convolucional de grafo é acompanhada por uma camada de *pooling* para minimizar um grafo em subgrafos. Os nós desses subgrafos possuem representações de alto nível do grafo original. A camada *readout* resume a representação final do grafo somando as todas representações *hidden* dos subgrafos.



Fonte: Wu *et al.* (2019).

Figura 4 – Uma GAE para *network embedding*. O codificador utiliza as camadas convolucionais de grafo para conseguir uma *network embedding* para cada nó. O decodificador calcula a distância emparelhada dada a *network embedding*. Depois de aplicar uma função de ativação não linear, o decodificador reconstrói a matriz de adjacência do grafo. A rede é treinada para minimizar a discrepância entre a matriz de adjacência real e a matriz de adjacência reconstruída.



Fonte: Wu *et al.* (2019).

Figura 5 – Uma STGNN para previsão de grafos espaço-temporais. Uma camada convolucional de grafo é acompanhada por uma camada 1D-CNN. A camada convolucional de grafo opera em $\vec{A}$ e $\vec{X}^{(t)}$ para capturar a dependência espacial, enquanto que a camada 1D-CNN "desliza" em $\vec{X}$ e ao longo do eixo do tempo para capturar a dependência temporal. A camada de saída é uma transformação linear, gerando uma predição para cada nó, tal como o valor futuro no próximo passo.

2.5.2 Tipos de problemas

Com a estrutura de grafos e informações de nós como entrada, as saídas das GNNs podem focar em diferentes tarefas analíticas de grafos com um dos seguintes mecanismos:

Node-level (nível de nó) produz uma saída relacionada a regressão de nó e tarefas de classificação. RecGNNs e ConvGNNs podem extrair representações de alto nível de nós por propagação de informação ou convolução de grafo. Com um Multi-Perceptron ou uma *softmax* como uma camada de saída, GNNs são capazes de realizar tarefas a nível de nó de maneiras *end-to-end*.

Edge-level (nível de arestas) produz uma saída relacionada a classificação de arestas e predição de ligações. Com duas representações ocultas de nós das GNNs como entrada, uma função similar ou uma rede neural pode ser utilizada para prever a força de conexão ou o rótulo de uma aresta.

Graph-level (nível de grafo) produz uma saída relacionada a classificação de grafos. Para obter a representação compacta a nível de grafo, GNNs são frequentemente combinadas com operações de *pooling* e *readout*.

2.5.2.1 Treinamento de GNNs

Muitas GNNs (e.g., ConvGNNs) podem ser treinadas de maneiras semi-supervisionadas ou até mesmo puramente não-supervisionadas dentro de *frameworks* de aprendizado *end-to-end*, dependendo da tarefa e das informações disponíveis em mãos.

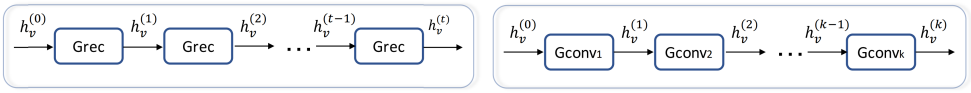
a. Aprendizado semi-supervisionado para classificação a nível de nó: dado uma única rede com nós parcialmente rotulados e outros ainda não rotulados, ConvGNNs podem aprendem um modelo robusto que identifica eficientemente os nós ainda não rotulados. Para isso, um *framework end-to-end* pode ser construído empilhando algumas camadas convolucionais de grafos seguido de camadas *softmax* para classificação multi-classe.

b. Aprendizado supervisionado para classificação a nível de grafo: classificação a nível de grafo visa prever rótulos para um grafo inteiro (ZHANG *et al.*, 2018). O aprendizado *end-to-end* para essa tarefa pode ser realizado com a combinação de camadas convolucionais de grafos, camadas de *graph pooling* e/ou camadas *readout*. Enquanto as camadas convolucionais de grafos são responsáveis por extrair representações alto nível de nós, camadas *graph pooling* desempenham o papel de *down-sampling*, que transforma cada grafo, um por vez, em uma sub-estrutura. Uma camada *readout* recolhe as representações de nós de cada grafo para uma representação em grafo. Aplicando multi-*layer* perceptron e uma camada *softmax* em representações de grafos, podemos construir um *framework end-to-end* para classificação de grafos.

c. Aprendizado não supervisionado de grafos *embedding*: Quando rótulos não estão disponíveis em um grafo, podemos aprender grafos *embedding* de uma maneira puramente não supervisionada em *frameworks end-to-end*. Esses algoritmos exploram informações a nível de arestas de duas maneiras. Uma maneira simples é adotar um *framework* auto-codificador onde o codificador emprega camadas convolucionais de grafos para incorporar o grafo em representações latentes sobre qual o decodificador é usado para reconstruir a estrutura do grafo (KIPF; WELLING, 2016b). A outra maneira pode ser vista com mais detalhes em Hamilton, Ying e Leskovec (2017).

2.6 ConvGNNs

GNNs convolucionais (ConvGNNs) estão intimamente relacionadas a GNNs recorrentes. Em vez de estados de nós iterativos com restrições contrativas, ConvGNNs abordam as dependências mútuas e cíclicas utilizando um número fixo de camadas com diferentes pesos em cada camada. Essa importante distinção é ilustrada na Figura 5. Como as convoluções de grafos são mais eficientes e convenientes para hibridizar com outras redes neurais, a popu-



(a) RecGNNs usam a mesma camada recorrente de grafo (Grec) na atualização das representações.

(b) ConvGNNs usam uma camada convolucional de grafo diferente a cada atualização das representações.

Fonte: Wu *et al.* (2019).

Figura 6 – RecGNNs VS. ConvGNNs

laridade das ConvGNNs tem crescido rapidamente nos últimos anos. ConvGNNs dividiram-se em duas categorias: os modelos espectrais e os espaciais. A abordagem do modelo espectral define convoluções de grafos introduzindo filtros a partir da perspectiva de processamento de sinais de grafos, onde a operação de convolução de grafos é interpretada como uma remoção de ruídos dos sinais de grafos. A abordagem do modelo espacial herda ideias das RecGNNs, ou seja, a definição de convolução de grafos aqui se dá por agregação de informações. Desde que a GCN preencheu a lacuna entre a abordagem espectral e a abordagem espacial, os métodos que utilizam modelos espaciais têm-se desenvolvido rapidamente, graças a sua atraente eficiência, flexibilidade e generalidade.

2.6.1 ConvGNNs espectrais

Métodos espectrais tem uma sólida fundamentação matemática em processamento de sinais de grafos. Eles assumem que os grafos sejam não-direcionados. Uma robusta representação matemática de um grafo não-direcionado é a matriz Laplaciana normalizada, definida como $\vec{L} = \vec{I}_n - \vec{D}^{\frac{-1}{2}} \vec{A} \vec{D}^{\frac{-1}{2}}$, onde $\vec{D}$ é a matriz diagonal dos graus dos nós. A matriz Laplaciana normalizada é real, simétrica e semi-definida positiva. Dado essas propriedades, a matriz Laplaciana normalizada pode ser fatorada como $\vec{L} = \vec{U} \Lambda \vec{U}^T$, onde $\vec{U} = [u_0, u_1, ..., u_{n-1}]$ é a matriz de autovetores ordenado pelos autovalores e Λ é a matriz diagonal de autovalores (espectro), $\Lambda_{ii} = \lambda_i$. Os autovetores da matriz Laplaciana normalizada formam um espaço ortonormal, matematicamente falando, $\vec{U}^T \vec{U} = \vec{I}$. Em processamento de sinais de grafos, um sinal de grafo $\vec{X} \in \mathbb{R}^n$ é um vetor de características de todos os nós de um grafo onde x_i é o valor do i-ésimo nó. A transformada de Fourier em grafos para um sinal $\vec{X}$ é definida como $F(\vec{X}) = \vec{U}^T \vec{X}$ e a transformada de Fourier inversa em grafos é definida como $F^{-1}(\vec{\hat{X}}) = U\vec{\hat{X}}$, onde $\vec{\hat{x}}$ representa o sinal resultante da transformada de Fourier em grafos. A transformada de Fourier em grafos projeta o sinal de entrada no espaço ortonormal, onde a base é formada pelos autovetores da matriz Laplaciana normalizada. Os elementos do sinal transformado $\vec{\hat{X}}$ são as coordenadas do sinal de grafo no novo espaço, então o sinal de entrada pode ser representado como $\vec{X} = \sum_i \hat{x}_i \vec{u}_i$, que é exatamente a transformada de Fourier inversa em grafos. Agora, a convolução em grafos do sinal de entrada $\vec{X}$ com um filtro $\vec{g} \in \mathbb{R}^n$ é definida como

$$\vec{X} *_G \vec{g}_\theta = F^{-1}(F(\vec{X}) \cdot F(\vec{g}))$$

onde · é o produto de Hadamard.Se assumirmos um filtro como $\vec{g_\theta} = diag(\vec{U^T}g)$, então a convolução de grafos espectrais é simplificada como

$$\vec{X} *_G \vec{g}_\theta = U g_\theta \vec{U^T} X$$

Todas as ConvGNNs baseadas em espectro seguem essa definição. A diferença está na escolha do filtro $\vec{g_\theta}$.

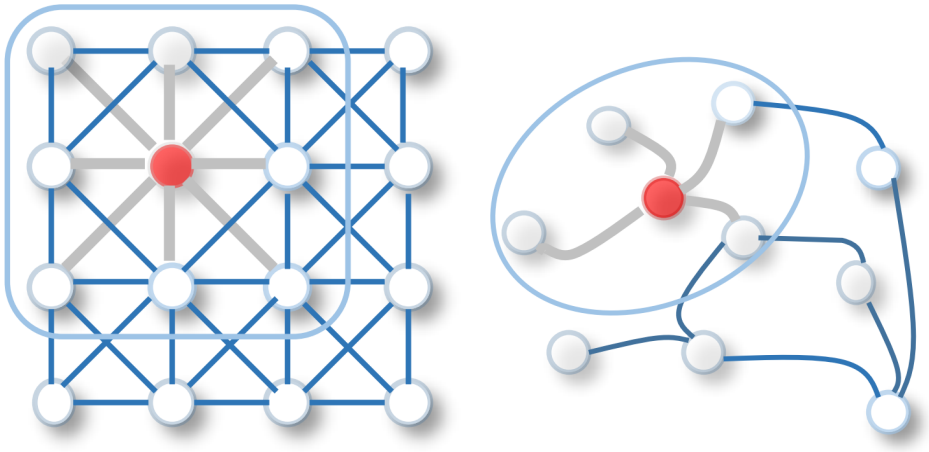
Redes neurais de convolução espectral (do inglês *spectral convolutional neural network*, ou ainda Spectral CNN) assumem o filtro $g_\theta = \vec{\Theta}_{i,j}^{(k)}$ como um conjunto de parâmetros aprendíveis e considera um sinal de grafo com múltiplos canais. A camada de convolução de grafos de uma Spectral CNN é definida como

$$\vec{H}_{:,j}^{(k)} = \sigma(\sum_{i=1}^{f_{k-1}} U\vec{\Theta}_{i,j}^{(k-1)}\vec{U}^T \vec{H}_{:,i}^{(k-1)})(j = 1, 2, ..., f_k)$$

onde k é o índice da camada, $\vec{H}^{(k-1)} \in \mathbb{R}^{n \times f_{k-1}}$ é o sinal de entrada do grafo, $\vec{H}^{(0)} = \vec{X}$, f_{k-1} é o número de canais de entrada e f_k é o número de canais de saída, $\vec{\Theta}_{i,j}^{(k-1)}$ é a matriz diagonal preenchida com parâmetros aprendíveis. Graças a autodecomposição da matriz Laplaciana, as Spectral CNNs encaram três limitações. Primeiro, qualquer perturbação no grafo resulta em uma mudança de autobase (autovalores e autovetores). Segundo, os filtros aprendíveis são um domínio dependente, ou seja, eles não podem ser aplicados a grafos com diferentes estruturas. Terceiro, a autodecomposição requer uma complexidade computacional $O(n^3)$. Os modelos ChebNet e GCN, apesar de espectrais, reduziram a complexidade computacional a $O(m)$, realizando várias aproximações e simplificações. O modelo GCN pode também ser estudado do ponto de vista dos modelos espaciais, uma vez que ele utiliza os conceitos de agregação de informação. Por ser o foco deste trabalho, uma seção mais a frente está reservada para discuti-lo.

2.6.2 ConvGNNs espaciais

Motivado pela operação de convolução da convencional CNN em uma imagem, os modelos espaciais definem convolução de grafo baseado nas relações espaciais entre nós. Imagens podem ser consideradas uma forma especial de grafo com cada pixel representando um



(a) Convolução 2D. Análogas a grafos, cada pixel em uma imagem é tomado como um nó, onde seus vizinhos são determinados pelo tamanho do filtro. A convolução 2D leva a média ponderada dos valores de pixel do nó vermelho junto com seus vizinhos. Os vizinhos de um nó são ordenados e possuem um tamanho fixo.

(b) Convolução de grafo. Para conseguir uma representação *hidden* do nó vermelho, uma simples solução de operação convolucional em grafo é pegar o valor médio das características do nó vermelho e de seus vizinhos. Diferentemente dos dados de imagem, os vizinhos de um nó são desordenados e possuem tamanhos variáveis.

Fonte: Wu *et al.* (2019).

Figura 7 – Convolução 2D VS. Convolução em grafos

nó. Cada pixel está diretamente conectado aos pixels mais próximos, como ilustrado na Figura 6a. Com uma janela 3×3, a vizinhança de cada nó é formada por 8 pixels. Um filtro então é aplicado a este grafo 3×3 tomando a média ponderada dos valores do pixel do nó central e seus vizinhos em cada canal. Semelhantemente, a convolução de grafos baseado em métodos espaciais convolui a representação do nó central com a representação de seus vizinhos para obter uma representação atualizada do nó central, como ilustrado na Figura 6b. Olhando por outro ponto de vista, ConvGNNs espaciais compartilham com as RecGNNs a mesma ideia de propagação de informação. A operação de convolução espacial em grafos essencialmente propaga as informações de cada nó ao longo das arestas.

2.6.3 Comparação entre modelos espectrais e espaciais

Modelos espectrais tem uma fundamentação teórica em processamento de sinais de grafos. Projetando novos filtros de sinais de grafos, pode-se teoricamente construir novas ConvGNNs. Entretanto, os modelos espaciais são melhores do que os modelos espectrais, graças a sua eficiência, generalidade e flexibilidade. Primeiro, modelos espectrais são menos eficientes que os modelos espaciais. Modelos espectrais ou calculam autovetores ou lidam com grafos inteiros ao mesmo tempo. Modelos espaciais são mais escaláveis para grafos maiores por realizarem a convolução diretamente no domínio do grafo via agregação de informação. A computação pode ser realizada em lotes de nós ao invés do grafo inteiro. Segundo, modelos

espectrais são dependentes da base de Fourier e são ineficientes para generalizar novos grafos. Eles assumem um grafo fixo. Qualquer perturbação no grafo resultaria em uma mudança de autovalores e autovetores. Modelos espaciais, por outro lado, realizam convolução de grafos localmente em cada grafo onde pesos podem ser facilmente compartilhados entre diferentes localizações e estruturas. Terceiro, modelos espectrais são limitados a operações em grafos não-direcionados. Modelos espaciais são mais flexíveis para lidar com várias entradas de grafos, tais como entradas de arestas, grafos direcionados, e grafos heterogêneos, graças ao fato dessas entradas serem incorporadas facilmente por uma função de agregação.

2.7 GCN

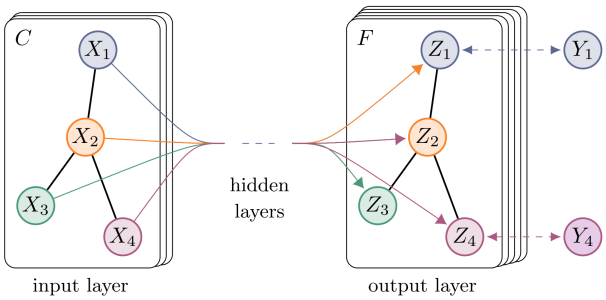
A GCN (do inglês *graph convolutional network*) foi elaborada para resolver problemas de classificação de nós (tal como documentos) em um grafo (tal como uma rede de citações), onde somente um pequeno subconjunto de nós esteja rotulado (KIPF; WELLING, 2016a). Este problema pode ser enquadrado como aprendizado semi-supervisionado baseado em grafos, onde as informações dos rótulos são absorvidas utilizando regularização de grafos, como por exemplo a regularização de grafos Laplacianos como uma função de perda:

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{reg}, \text{ com } \mathcal{L}_{reg} = \sum_{i,j} \vec{A}_{ij} ||f(\vec{X}_i) - f(\vec{X}_j)||^2 = f(\vec{X})^T \Delta f(\vec{X})$$

Aqui, $\mathcal{L}_0$ é a perda supervisionada da parte rotulada do grafo, $f(\cdot)$ pode ser uma rede neural qualquer, como por exemplo uma função diferenciável, λ é o fator de ponderação e $\vec{X}$ é a matriz de características do nós, tendo cada linha um vetor de características $\vec{X}_i$. $\Delta = \vec{D} - \vec{A}$ é o grafo Laplaciano não normalizado de um grafo não-direcionado $G = (V, E)$ com n nós $v_i \in V$, arestas $(v_i, v_j) \in E$. $\vec{A} \in \mathbb{R}^{n \times n}$ é a matriz de adjacência (binária ou com peso) e D_{ii} a matriz de graus. A formulação acima baseia-se na suposição de que os nós conectados no grafo provavelmente compartilham o mesmo rótulo. Esta suposição, entretanto, pode restringir a capacidade de modelagem, uma vez que as arestas do grafo pode não necessariamente codificar similaridade entre os nós, mas pode conter informações adicionais.

Para construir a GCN, a estrutura do grafo foi codificada utilizando um modelo $f(X, \vec{A})$, treinado em alvos supervisionados $\mathcal{L}_0$ para todos os nós rotulados, evitando assim regularização explícita em grafos na função de perda. Condicionando $f(\cdot)$ com a matriz de adjacência do grafo, o modelo distribuirá gradiente de informação da perda supervisionada $\mathcal{L}_0$, aprendendo assim representações tanto de nós não rotulados quanto de nós rotulados.

O artigo original, ao qual se baseia estas constatações (KIPF; WELLING, 2016a) sobre a construção da GCN, demonstra matematicamente a simplificação dos filtros utilizados em modelos espectrais, chegando a conclusão de que esta arquitetura possui complexidade linear



Fonte: Wu *et al.* (2019).

Figura 8 – Representação esquemática da GCN de múltiplas camadas para aprendizado semi-supervisionado com canais de entrada C e mapas de recursos F na camada de saída

nas arestas, ou seja, $O(m)$. Além disso, dado o condicionamento de $f(X, A)$, espera-se que este modelo seja especialmente poderoso em cenários onde a matriz de adjacência $\vec{A}$ contém informações não presentes no vetor de características $\vec{X}$, tal como citações entre documentos em uma rede de citações ou relações em um grafo de conhecimento. Para este trabalho, assumiremos isso como verdade.

O modelo geral, uma GCN multi-camadas para aprendizado semi-supervisionado, é esquematicamente retratado na Figura 7.

2.7.1 Limitações da GCN

Embora a GCN seja um método eficaz na classificação de problemas supervisionados, a mesma possui limitações que devem ser levadas em conta na hora de optar pelo seu uso. Dentre elas podemos citar:

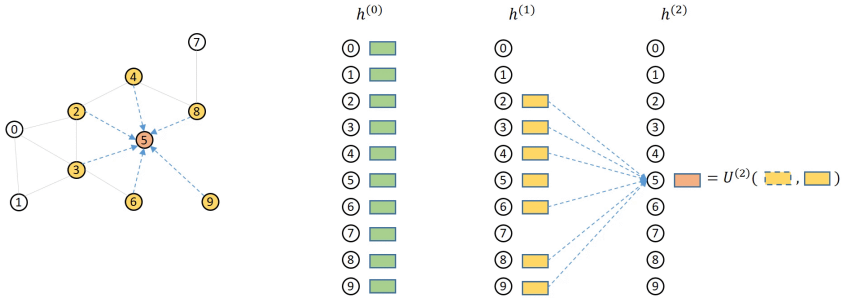
a. Requerimento de memória: a quantidade de memória requerida pela GCN cresce linearmente com o tamanho do *dataset*. Para grafos grandes que não podem ser treinados na memória da GPU, treinar na CPU ainda pode ser uma opção viável. Utilizar gradiente descendente estocástico (*mini-batch*) pode aliviar esse problema. O procedimento de geração de *mini-batches*, entretanto, deve levar em conta o número de camadas no modelo GCN, dado o fato de que a vizinhança de ordem K de uma GCN com K camadas deve ser armazenada na memória para um procedimento exato. Para grafos muito grande e densamente conectados, algumas aproximações podem ser necessárias. Reservo a próxima subseção para discutir sobre uma solução para esse problema.

b. Arestas direcionadas e características de arestas: a GCN não suporta características de arestas e é limitada a grafos não direcionados (com peso ou sem peso). Entretanto, Kipf provou que é possível lidar com grafos direcionados e características de arestas representando o grafo direcionado original como um grafo não direcionado bipartido, com nós adicionais que representam as arestas do grafo original.

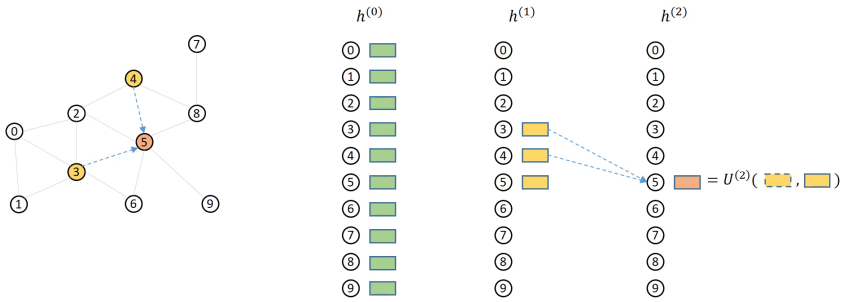
2.7.2 *Neighborhood Sampler*

Calcular representações para um pequeno número de nós frequentemente requer selecionar características como entradas de um grande número de nós. Utilizar todos os vizinhos para agregar mensagens pode ser muito caro uma vez que selecionar os nós necessários para utilizar na entrada pode facilmente cobrir uma vasta porção do grafo, especialmente em grafos do mundo real.

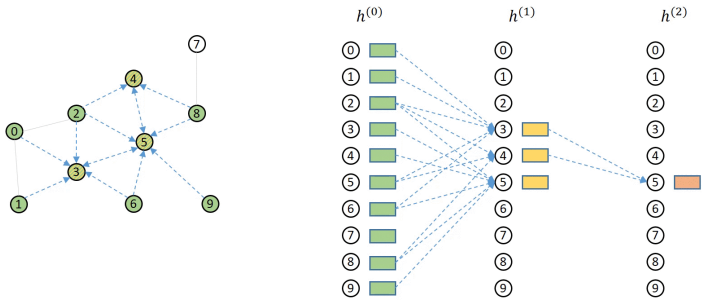
Neighbor Sampling resolve este problema selecionando um *subset* dos vizinhos para realizar a agregação. Veja na Figura 9 um exemplo:



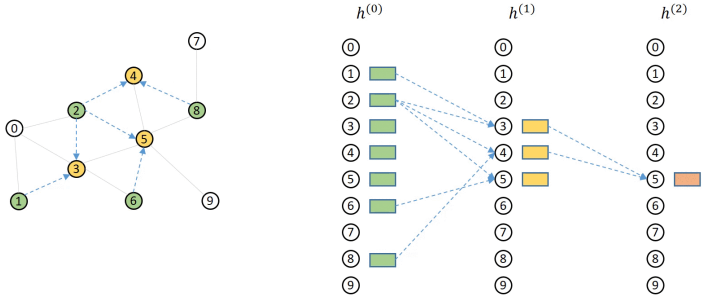
(a) Buscamos uma representação do nó 5 para a terceira camada da rede neural. A maneira tradicional de fazer isso seria agregar as informações de todos os seus vizinhos, o que pode ser muito caro em situações onde um grafo é muito grande e é densamente conectado.



(b) Podemos selecionar somente dois vizinhos ao invés de todos. Isto é responsabilidade do *sampler* e para cada época do treinamento um *subset* diferente é escolhido.



(c) Percebe-se o quão caro pode ser um processo de agregação de mensagem. A quantidade de arestas e mensagens a serem passadas subiu bastante para somente 3 nós. Aplicamos também a ideia de amostragem aqui para obtermos a próxima figura.



(d) Este é o resultado final do *sampling*. Ao final do treinamento o *sampler* terá se encarregado de selecionar *subsets* diferentes o suficiente para gerar boas representações dos nós.

Fonte: adaptado de Introduction... (2018).

Figura 9 – Fluxo de agregação de mensagem realizado pelo Sampler

3 MATERIAIS E MÉTODOS

Descreve-se neste Capítulo os materiais e métodos utilizados para atingir os resultados deste trabalho.

3.1 *Frameworks* de programação

Os *frameworks* utilizados neste trabalho foram o DGL (WANG *et al.*, 2019) e o RayTune (LIAW *et al.*, 2018). O primeiro contém todas as classes necessárias para montar o modelo GCN, bem como o *sampler* e o *loader* do *dataset*. O segundo contém o GridSearch, método de busca exaustivo de hiperparâmetros utilizado neste trabalho, explicado brevemente na Subseção 3.2. O RayTune contém também o *Scheduler* ASHA (LI *et al.*, 2018a), utilizado neste trabalho apenas para realizar treinamento em paralelo.

3.2 GridSearch

O GridSearch é um método de busca exaustivo utilizado na seleção de modelos de Machine Learning. Ele seleciona o melhor modelo dentre todos os modelos possíveis no espaço de busca passado como parâmetro. A escolha dos hiperparâmetros buscados no GridSearch é melhor explicado na Subseção 4.1.1.

3.3 Base de dados

A base de dados utilizada neste trabalho é a Cora Citation Network, que consiste em um grafo com 2708 nós (publicações científicas) e 5429 arestas (citações entre as publicações). Cada nó possui um vetor de 1433 índices preenchidos com 0 ou 1, representando palavras chaves contidas no texto da publicação.

Cada uma das publicações está classificada em uma de 7 classes, como ilustra a Tabela 1.

Classes	Quantidade na base
Neural Networks	818
Probabilistic Methods	426
Genethic Algorithms	418
Theory	351
Case Based	298
Reinforcement Learning	217
Rule Learning	180

Tabela 1 – Classes e suas quantidades no dataset

Os nós foram divididos seguindo a mesma metodologia utilizada no trabalho de Kipf e Welling (2016a), explicada detalhadamente na Subseção 3.3.1. Ele se baseia na metodologia

de Yang, Cohen e Salakhutdinov (2016) que seleciona aleatoriamente 20 nós de cada classe para treinamento e 1000 nós rotulados para teste, e ainda adiciona uma base de validação de 500 nós para otimização de hiperparâmetros. O restante dos nós são considerados como não rotulados. A Tabela 2 ilustra essa divisão.

Train	140
Validation	500
Test	1000

Tabela 2 – Cora Dataset Split

Vale lembrar que, apesar dessa rede conter 5429 arestas, após um pré-processamento ela terá aproximadamente o dobro. Isso porque este grafo é não direcional, e a melhor maneira de representar isso computacionalmente sem realizar muitas comparações e facilitando os cálculos, é criando duas arestas por conexão, cada uma representando um sentido diferente. Assim, cada aresta entre dois nós diferentes no grafo original será representado aqui por duas arestas com sentidos contrários. Depois disso remove-se os laços (ou *self-loops*) que existirem, evitando que na propagação de mensagens, um nó receba informações de si mesmo, resultando em 10556 arestas. Este pré-processamento já é realizado pelo *framework* DGL adotado neste trabalho, sendo necessário somente a importação do *dataset*.

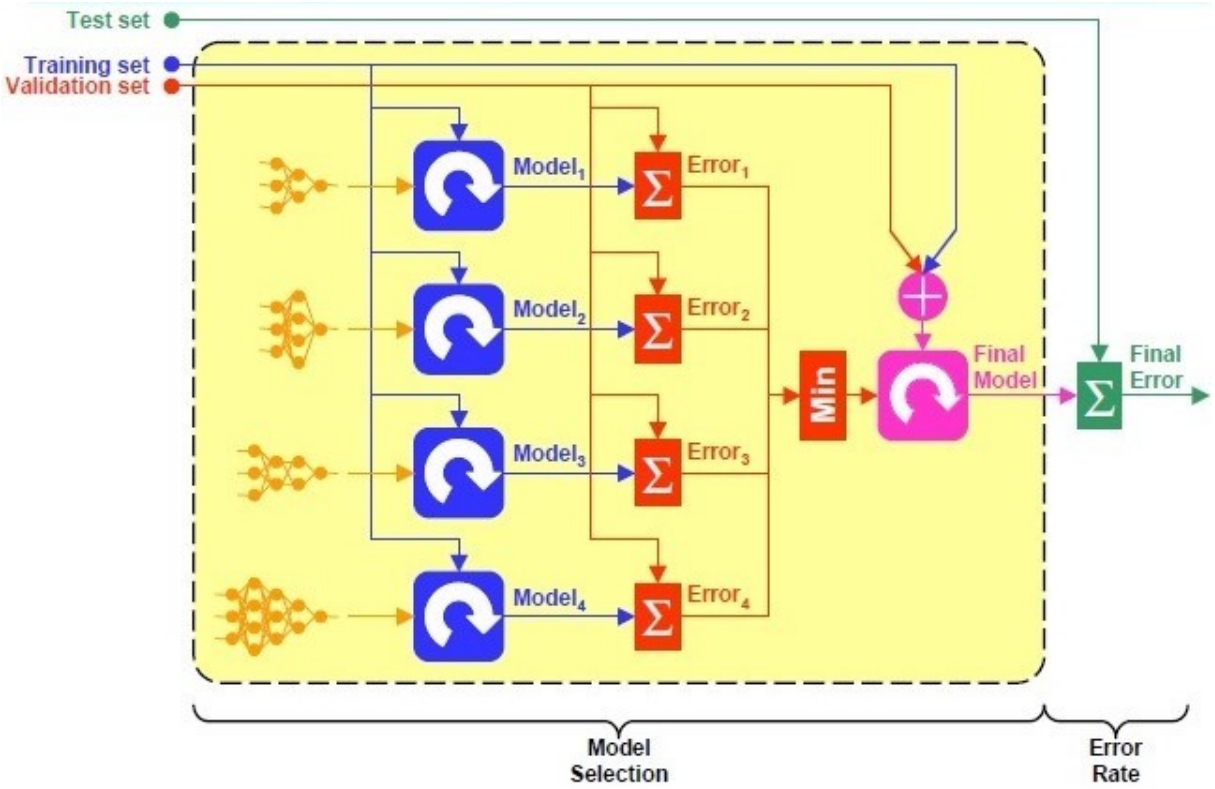
3.3.1 Seleção e validação do modelo

Para a seleção e validação do modelo foi adotado o método *3-way holdout*. Segundo Jaokar (2019) este método consiste em separar a base de dados em treinamento, validação e teste e é motivado por dois problemas fundamentais em problemas de aprendizado: seleção de modelo e avaliação de desempenho.

- **Seleção de modelo:** envolve a seleção de hiperparâmetros ou modelos otimizados. Técnicas de reconhecimento de padrões têm um ou mais hiperparâmetros livres. Por exemplo, o número de vizinhos e número de neurônios em um classificador kNN (*k-Nearest Neighbors*) ou camadas e método de otimização em MLPs (*Multi-Layer Perceptron*). A seleção correta desses hiperparâmetros determina a eficiência da solução.
- **Validação do modelo:** Para que o melhor modelo seja selecionado corretamente é preciso avaliar seu desempenho. Uma vez que o objetivo do aprendizado é classificar dados desconhecidos, espera-se uma boa generalização na etapa de treinamento. Para averiguar se o erro obtido é real ou proveniente do acaso é necessário apresentar dados diferentes daqueles que foram utilizados para treinar.

Dito isto, a metodologia de avaliação de desempenho nesse trabalho segue os seguintes passos, conforme a Figura 10:

1. Dividir os dados disponíveis em treinamento, validação e teste
2. Selecionar a arquitetura e hiper-parâmetros de treinamento
3. Treinar o modelo utilizando a base de treinamento
4. Avaliar o modelo utilizando a base de validação
5. Repetir os passos 2 ao 4 utilizando diferentes arquiteturas e parâmetros de treinamento
6. Selecionar o melhor modelo e treiná-lo utilizando os dados da base de treinamento e validação
7. Avaliar o modelo final utilizando a base de teste



Fonte: Jaokar (2019).
Figura 10 – Método de seleção e validação de modelo de Deep Learning

Para a otimização dos hiperparâmetros do modelo foi utilizado o método GridSearch, explicado anteriormente na Seção 3.2.

Após a seleção, validação e re-treinamento do melhor modelo, será calculado um intervalo de confiança para a acurácia do modelo na base de teste. Os resultados podem ser vistos no Capítulo 4.

4 RESULTADOS E DISCUSSÕES

Neste Capítulo são apresentados os resultados e discussões acerca dos mesmos.

4.1 Construção e seleção do modelo

Utilizando como base o modelo proposto por Kipf e Welling (2016a), construiu-se um modelo de classificação com duas camadas convolucionais. Para seleção dos hiperparâmetros deste modelo utilizou-se o método GridSearch, um buscador exaustivo de hiperparâmetros de Machine Learning. Ao final desta busca exaustiva, selecionou-se dentre todos os modelos possíveis aquele que obteve a melhor acurácia na base de treinamento para ser retreinado utilizando a base de testes, conforme metodologia ilustrada na Figura 10. As subseções a seguir ilustram os resultados deste processo.

4.1.1 Seleção do modelo

Ao construir uma arquitetura de rede neural, os seguintes hiperparâmetros precisam ser ajustados para otimizar o resultado:

- Número de camadas
- *hidden layer size*
- Função de Ativação
- *learning rate*
- *batch size*
- Otimizador

Por tratar-se de muitos hiperparâmetros, a busca realizada pelo GridSearch pode levar muito tempo, uma vez que ela é exaustiva, ou seja, testa todas as possibilidades dentro do seu espaço de busca. Por isso é desejável fixar um valor para alguns hiperparâmetros. Sendo assim, a seleção dos hiperparâmetros foi realizada em duas etapas, explicadas nas subseções a seguir.

4.1.1.1 Etapa 1

Na primeira etapa foi realizada uma análise de desempenho nos hiperparâmetros número de camadas e otimizador, mantendo um valor fixo para todos os outros hiperparâmetros. Esses foram os hiperparâmetros escolhidos pois:

- Implementar um modelo que varia em número de camadas torna-o diferente de qualquer outro. Portanto testar este hiperparâmetro antes da seleção com o GridSearch simplifica a implementação.

- O Adam é conhecidamente um dos melhores otimizadores na literatura de Machine Learning. Portanto, deseja-se excluir a necessidade de testar qualquer outro durante a seleção do GridSearch.

É possível observar na Figura 11 que conforme cresce o número de camadas, cresce também o ruído no aprendizado do modelo, além de demorar mais para atingir a acurácia final. Além disso, aumentar o número de camadas aumenta também a quantidade de neurônios que precisam ser treinados e ajustados, tornando o processo de treinamento mais caro. Sendo assim, optou-se por fixar o hiperparâmetro número de camadas em dois, uma vez que as demais opções afetam negativamente a performance da rede neural.

Com relação ao otimizador, é possível observar primeiramente na Figura 12 que utilizar o SGD está fora de questão, uma vez que o modelo com este otimizador atingiu uma acurácia de 0.16%. Comparando agora o Adam com o RMSprop, percebe-se um pequeno ruído no aprendizado utilizando o RMSprop, além de este modelo não ter alcançado uma acurácia maior que a do Adam. Por essa razão, optou-se por utilizar o otimizador Adam no treinamento do modelo final deste trabalho.

Hiperparâmetros fixados durante a variação de número de camadas:

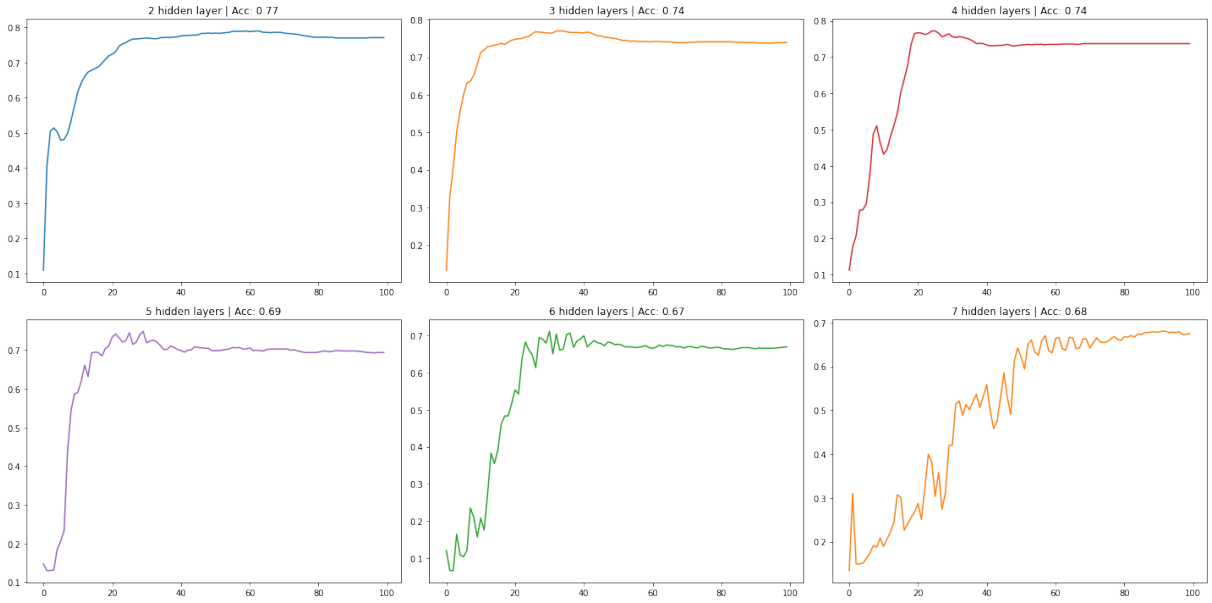
- Otimizador: Adam

- Learning Rate: 0.01

- Função de ativação: ReLu

- Tamanho do batch de entrada durante o treinamento: 1

- Tamanho de cada camada: 16

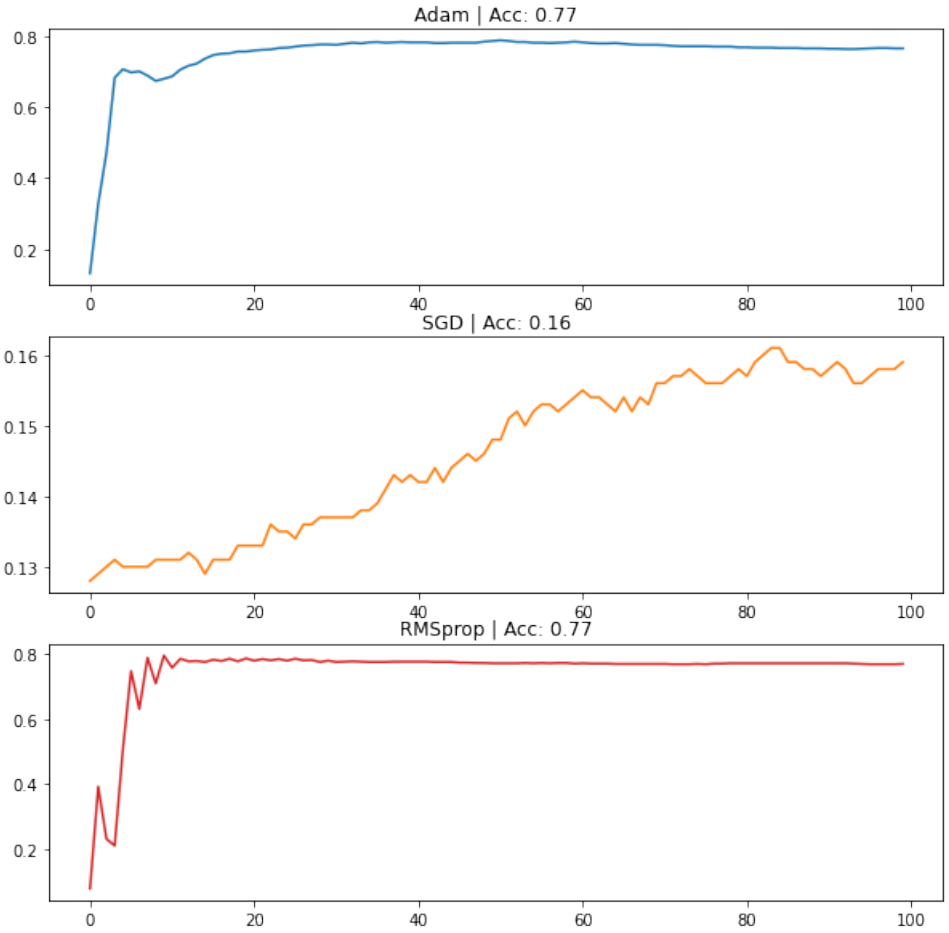


Fonte: autoria própria.

Figura 11 – Diferentes valores para número de camadas

Hiperparâmetros fixados durante a variação de otimizadores:

- Número de camadas: 2
- Learning Rate: 0.01
- Função de ativação: ReLu
- Tamanho do batch de entrada durante o treinamento: 1
- Tamanho de cada camada: 16



Fonte: autoria própria.

Figura 12 – Diferentes valores para otimizador

4.1.1.2 Etapa 2

A segunda etapa de seleção dos hiperparâmetros é realizada então com o número de camadas fixado em dois, o otimizador fixado no Adam e é realizado uma busca nos demais hiperparâmetros utilizando o GridSearch. A Tabela 3 ilustra este espaço de busca.

Hiperparâmetro	Espaço de busca
Learning rate	0.1, 0.01, 0.001
Batch size	16, 32, 64
Hidden Layer Size	16, 32, 64
Activation Function	Silu, Relu, Elu, Celu

Tabela 3 – Espaço de busca do GridSearch

4.1.1.3 Construção e treinamento do modelo

Antes de realizar a seleção do melhor modelo, nessa seção é apresentado os detalhes de implementação do mesmo, bem como os detalhes relativos ao GridSearch.

Classe GCN utilizada no treinamento com GridSearch é apresentada na Listagem 4.1.1.3.

Listing 4.1 – Classe GCN

```
1  from dgl.nn import GraphConv
2
3  class GCN(nn.Module):
4    def __init__(self, in_feats, h_feats, num_classes, config):
5      super(GCN, self).__init__()
6      self.conv1 = GraphConv(in_feats, h_feats)
7      self.conv2 = GraphConv(h_feats, num_classes)
8
9      self.config = config
10
11   def forward(self, bipartites, x):
12
13     h_dst = x[:bipartites[0].num_dst_nodes()]
14     h = self.conv1(bipartites[0], (x, h_dst))
15     h = self.config["activation"](h)
16     h_dst = h[:bipartites[1].num_dst_nodes()]
17     h = self.conv2(bipartites[1], (h, h_dst))
18
19     return h
```

Esta é a classe GCN construída para este trabalho. Na linha 4 é definida a função de inicialização, recebendo os seguintes parâmetros:

- in_feats: quantidade de características de entrada do modelo. Para o *dataset* da Cora são as 1433 palavras representadas em um vetor, apresentado anteriormente na subseção 3.3.
- h_feats: quantidade de neurônios que a *hidden layer* possui.
- num_classes: número de classes do *dataset*, também apresentado na subseção 3.3.
- config: o arquivo de configuração para o GridSearch, definido na listagem 4.1.1.3.

Na linha 11 é definida a função de propagação deste modelo.
Algoritmo de treinamento, validação e teste, apresentado na Listagem 4.1.1.3.

Listing 4.2 – Algoritmo de treinamento e teste

```
import tqdm
import sklearn.metrics

def train_hypertuning(config):

  epochs = 100

  g_cora, dataset_cora, train_nids, test_nids, val_nids = load_data()

  sets = {
    "train": train_nids,
    "test": test_nids,
    "val": val_nids
  }

  sampler = dgl.dataloading.neighbor.MultiLayerFullNeighborSampler(2)

  model_cora = GCN(g_cora.ndata['feat'].shape[1], config["hidden_layer"], dataset_cora.

  trainset = sets["train"]
  testset = sets["test"]
  valset = sets["val"]

  optimizer = torch.optim.Adam(model_cora.parameters(), lr=config["lr"])

  trainloader = dgl.dataloading.NodeDataLoader(
      g_cora, trainset, sampler,
      batch_size=config["batch_size"]
  )

  valloader = dgl.dataloading.NodeDataLoader(
      g_cora, valset, sampler,
      batch_size=config["batch_size"]
```

```
34    )
35
36    for epoch in range(epochs):
37      model_cora.train()
38
39      with tqdm.tqdm(trainloader) as tq:
40        for step, (input_nodes, output_nodes, bipartites) in enumerate(tq):
41          inputs = bipartites[0].srcdata['feat']
42          labels = bipartites[-1].dstdata['label']
43
44          predictions = model_cora(bipartites, inputs)
45
46          loss = F.cross_entropy(predictions, labels)
47          optimizer.zero_grad()
48          loss.backward()
49          optimizer.step()
50
51          accuracy = sklearn.metrics.accuracy_score(labels, predictions.argmax(1))
52
53          tq.set_postfix({'loss': '%.03f' % loss.item(),
54                          'acc': '%.03f' % accuracy}
55                         , refresh=True)
56
57      model_cora.eval()
58
59      predictions = []
60      labels = []
61      best_accuracy = 0
62      with tqdm.tqdm(valloader) as tq, torch.no_grad():
63        for input_nodes, output_nodes, bipartites in tq:
64          inputs = bipartites[0].srcdata['feat']
65          labels.append(bipartites[-1].dstdata['label'])
66          predictions.append(model_cora(bipartites, inputs).argmax(1))
67        predictions = np.concatenate(predictions)
```

```
68          labels = np.concatenate(labels)
69          accuracy = sklearn.metrics.accuracy_score(labels, predictions)
70          print('Epoch {} Validation Accuracy {}'.format(epoch, accuracy))
71          if best_accuracy < accuracy:
72            best_accuracy = accuracy
73
74      tune.report(accuracy=best_accuracy)
```

Esta é a função de treinamento utilizada para encontrar o melhor modelo proposto pelo GridSearch.

Até a linha 31 são somente criações de objetos utilizado no treinamento:

- linha 8: carregamento dos dados
- linha 16: criação do Sampler
- linha 18: criação do modelo GCN
- linha 24: definição do otimizador
- linhas 26 e 31: definição dos carregadores de nós

O restante é responsável por treinar, calcular a acurácia e armazenar o melhor valor encontrado.

Configuração dos hiperparâmetros para o GridSearch, apresentado na Listagem 4.1.1.3

Listing 4.3 – Configuração de hiperparâmetros para o GridSearch

```
1 config ={
2     "lr": tune.grid_search([1e-1, 1e-2, 1e-3]),
3     "batch_size": tune.grid_search([16, 32, 64]),
4     "hidden_layer": tune.grid_search([16, 32, 64]),
5     "activation": tune.grid_search([F.silu, F.relu, F.elu, F.celu])
6 }
```

O GridSearch navega neste arquivo para montar o seu espaço de busca.

Configuração do *Scheduler*, *Reporter* e comando de inicialização do GridSearch, apresentado na Listagem 4.1.1.3.

Listing 4.4 – Configuração do *Scheduler*, *Reporter* e comando de inicialização do GridSearch

```
1 from ray import tune
2 from ray.tune import CLIReporter
3 from ray.tune.schedulers import ASHAScheduler
```

```
4
5  scheduler = ASHAScheduler(
6          metric="accuracy",
7          mode="max",
8          reduction_factor=2)
9
10 reporter = CLIReporter(
11         metric_columns=['accuracy']
12         )
13
14 result = tune.run(
15     partial(train_hypertuning),
16     config=config,
17     scheduler=scheduler,
18     progress_reporter=reporter
19 )
```

Na linha 5 é criado o *Scheduler* ASHA (LI *et al.*, 2018a), utilizado aqui somente para treinamento em paralelo, como citado anteriormente na Seção 3.1. É um parâmetro obrigatório para iniciar o GridSearch.

Na linha 10 é definido o *Reporter*. É um objeto que imprime na tela os resultados dos modelos conforme eles vão finalizando para fins de acompanhamento do treinamento.

Os dez melhores modelos encontrados pelo GridSearch podem ser observados na Tabela 4.

Accuracy	Learning Rate	Batch Size	Hidden Layer Size	Activation Function
0.766	**0.10**	**16**	**16**	**Elu**
0.764	0.10	16	16	Celu
0.764	0.01	16	32	Celu
0.764	0.01	16	64	Elu
0.762	0.01	16	32	Elu
0.762	0.01	16	64	Celu
0.760	0.10	32	16	Elu
0.756	0.10	16	16	Silu
0.756	0.10	16	64	Relu
0.754	0.01	16	32	Silu

Tabela 4 – Melhores modelos encontrado pelo GridSearch

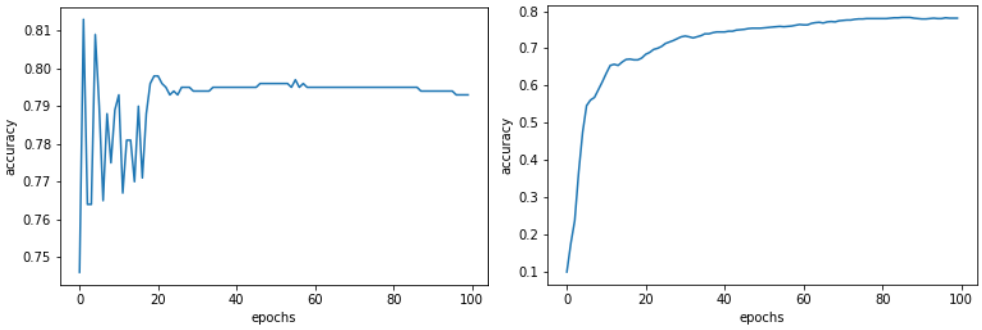
4.1.2 Avaliação do modelo

Conforme a Tabela 4, o melhor modelo foi selecionado (i.e. linha em negrito na Tabela 4) para ser re-treinado utilizando-se a base de treino mais a base de validação. A avaliação

Author	Accuracy
Wang e Leskovec (2020)	88.5% +- 1.5%
Sun, Zhu e Lin (2021)	85.46% +- 0.25%
Método proposto	79.3 % +- 3.7%
Tran (2018)	78.3%
Abu-El-Haija *et al.* (2018)	67.9%

Tabela 5 – Resultado comparado com outros trabalhos que tentaram classificar os artigos da Cora

final é realizado sobre a base de testes (durante o treinamento o modelo não enxerga a base de testes, como explicado na Subseção 3.3.1). O resultado final é uma acurácia de 0.793 e o aprendizado pode ser visto na Figura 13(a).



(a) Aprendizado estocástico do modelo final

(b) Aprendizado não estocástico (sem o uso do sampler)

Fonte: autoria própria.

Figura 13 – Comparação entre um aprendizado estocástico e um não estocástico

Na Figura 13 observa-se um comportamento oscilatório devido ao uso do *sampler*. Como no começo do treinamento o modelo ainda não recebeu mensagens suficientes para generalizar a classificação, ele tenta classificar com as poucas informações que já viu. Ao longo do aprendizado o *sampler* já forneceu diversas mensagens para o classificador, o que possibilita uma convergência ao final do treinamento. A Figura 13(b) ilustra um aprendizado não estocástico, ou seja, sem o uso do *sampler*.

Ainda que o aprendizado com o *sampler* possua ruídos, é preferível utilizá-lo pois seu uso economiza muito espaço em memória e em tempo de processamento, uma vez que ele cria representações satisfatórias para um nó utilizando o método de amostragem descrito na Subseção 2.7.2.

A Tabela 5 compara o resultado atingido nesse trabalho com outros que também tentaram classificar os artigos da Cora.

4.1.2.1 Intervalo de Confiança

Após a obtenção do modelo final, calcula-se um intervalo de confiança para este resultado utilizando a fórmula $CI = Acc \pm z\sqrt{\frac{Acc(1-Acc)}{n}}$, onde CI é o intervalo de confiança, Acc

a acurácia obtida pelo modelo final, z o coeficiente de distribuição normal para uma confiança de 99% e n a quantidade de amostras na base de teste utilizada para avaliar o modelo final.

O intervalo obtido foi $[0.76, 0.83]$, ou seja, para qualquer outro subconjunto desta base de dados, ao utilizar esse modelo para classificá-lo, tem-se 99% de certeza que a acurácia estará entre 0.76 e 0.83.

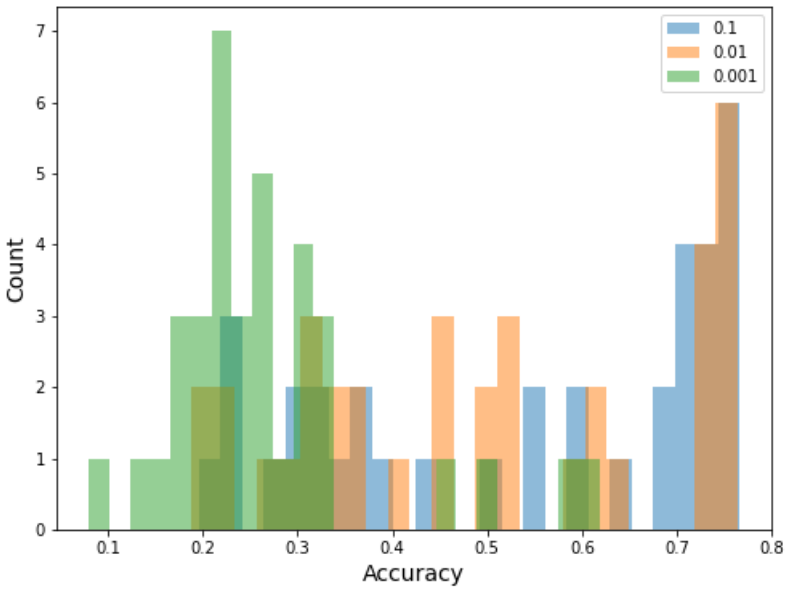
4.1.2.2 Análise da influência dos hiperparâmetros

A fim de entender os impactos dos hiperparâmetros em um processo de treinamento de ML e propor valores mais assertivos para trabalhos futuros e relacionados, segue nesta subseção uma análise sobre como os hiperparâmetros influenciaram os 108 modelos treinados (espaço de busca total do GridSearch: 3 possíveis valores de *learning rate* x 3 possíveis valores de *batch size* x 3 possíveis valores para *hidden layer size* x 4 possíveis valores para função de ativação = 108 modelos).

Os hiperparâmetros escolhidos para otimizar, como visto anteriormente na Subseção 4.1.1, foram:

- *learning rate*
- *batch size*
- *hidden layer size*
- função de ativação

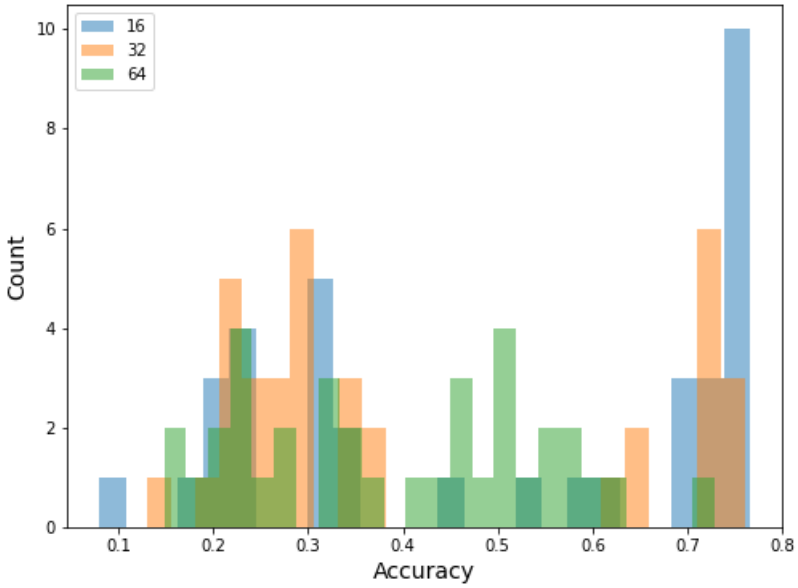
As Figuras 14, 15, 16 e 17 ilustram a influência que cada uma delas tiveram no treinamento, seguidos de uma análise.



Fonte: autoria própria.

Figura 14 – Distribuição dos modelos por *learning rate*

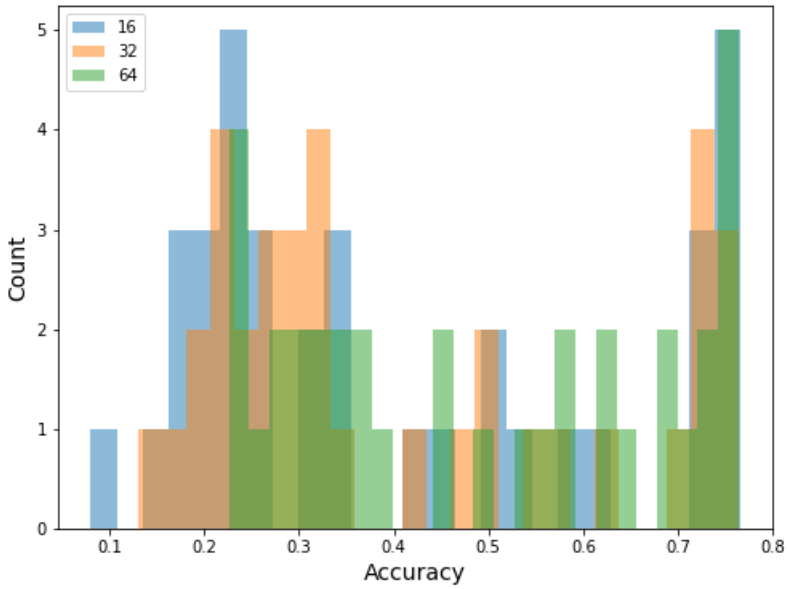
É bastante claro como um *learning rate* de 0.001 influencia negativamente o resultado. A grande maioria dos modelos com esse valor ficaram abaixo de 0.45 de acurácia, enquanto que os outros dois valores mostraram-se eficientes, tendo vários modelos acima de 0.60 de acurácia.



Fonte: autoria própria.

Figura 15 – Distribuição dos modelos por *batch size*

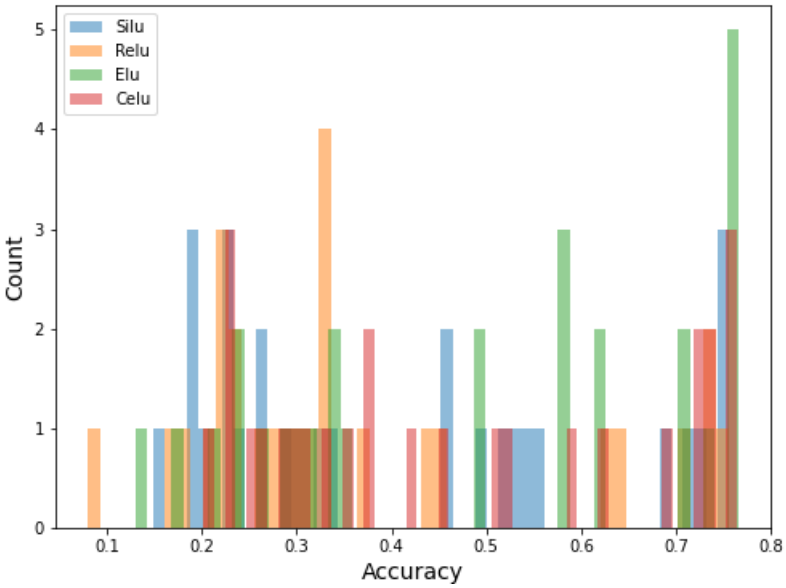
Um *batch size* de 16 é notoriamente a melhor opção, possuindo quase 10 modelos com mais de 0.70. De fato, se voltarmos a Tabela 4 veremos que somente uma arquitetura não possui *batch size* 16.



Fonte: autoria própria.

Figura 16 – Distribuição dos modelos por *hidden layer size*

Uma distribuição que não chama a atenção para nenhum valor específico. Conclui-se que este não é um parâmetro que influencia significativamente nos resultados.



Fonte: autoria própria.

Figura 17 – Distribuição dos modelos por função de ativação

Observando o gráfico, nota-se um desempenho melhor da função Elu, apesar das outras também estarem presentes nos melhores modelos. Entretanto, a função ReLu obteve mais resultados ruins do que bons. Com exceção desta última, todas as funções parecem ser boas opções para se testar neste problema.

5 CONCLUSÃO

Este trabalho apresentou uma maneira de realizar buscas por artigos científicos dada sua rede de citações. Comparando-se os resultados com outros trabalhos de mesmo objetivo na literatura, verifica-se uma boa taxa de acerto. Apesar de existirem trabalhos com resultados melhores é importante salientar que o modelo aqui empregado de GCN é um modelo bastante básico se comparado com os trabalhos estado da arte. Ainda assim, a acurácia é satisfatória, pois é melhor que muitos outros trabalhos da literatura.

Para obtenção desses resultados a base de dados CORA foi utilizada e uma busca de hiperparâmetros demonstrou que os hiperparâmetros investigados interferem de forma drástica na acurácia, de forma que uma combinação correta dos mesmos é necessária para alcançar uma boa taxa de acerto. Como apresentado na seção de resultados os hiperparâmetros que mais interferem são *learning rate* e *batch size* . O motivo por trás disso não está claro e pesquisas futuras podem ser devotadas a compreender esse comportamento.

De forma geral os resultados são encorajadores à continuação de pesquisas com modelos de grafos convolucionais. Como posto na literatura, existem modelos mais avançados já existentes e que precisam ser melhor compreendidos para sua aplicabilidade.

REFERÊNCIAS

ABU-EL-HAIJA, S. *et al.* Watch your step: Learning node embeddings via graph attention. **Advances in neural information processing systems**, v. 31, 2018.

Begum, A.; Fatima, F.; Sabahath, A. Implementation of deep learning algorithm with perceptron using tenzorflow library. In: **2019 International Conference on Communication and Signal Processing (ICCSP)**. [*S.l.*: *s.n.*], 2019. p. 0172–0175.

BRONSTEIN, M. M. *et al.* Geometric deep learning: Going beyond euclidean data. **IEEE Signal Process. Mag**, v. 34, n. 4, p. 18–42, 2017.

BRUNA, J. *et al.* Spectral networks and locally connected networks on graphs. In: BENGIO, Y.; LECUN, Y. (Ed.). **2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings**. [*s.n.*], 2014. Disponível em: http://arxiv.org/abs/1312.6203.

BUI, T. D.; RAVI, S.; RAMAVAJJALA, V. Neural graph learning: Training neural networks using graphs. In: **Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2018. (WSDM '18), p. 64–71. ISBN 9781450355810. Disponível em: https://doi.org/10.1145/3159652.3159731.

COSTA, L. da F. Convolution! (cdt-14). 08 2019.

DABHI, S.; PARMAR, M. **NodeNet: A Graph Regularised Neural Network for Node Classification**. 2020.

DUVENAUD, D. K. *et al.* Convolutional networks on graphs for learning molecular fingerprints. In: CORTES, C. *et al.* (Ed.). **Advances in Neural Information Processing Systems 28**. Curran Associates, Inc., 2015. p. 2224–2232. Disponível em: http://papers.nips.cc/paper/5954-convolutional-networks-on-graphs-for-learning-molecular-fingerprints.pdf.

Gallicchio, C.; Micheli, A. Graph echo state networks. In: **The 2010 International Joint Conference on Neural Networks (IJCNN)**. [*S.l.*: *s.n.*], 2010. p. 1–8. ISSN 2161-4407.

Gori, M.; Monfardini, G.; Scarselli, F. A new model for learning in graph domains. In: **Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.** [*S.l.*: *s.n.*], 2005. v. 2, p. 729–734 vol. 2. ISSN 2161-4393.

HAMILTON, W. L.; YING, R.; LESKOVEC, J. Inductive representation learning on large graphs. **CoRR**, abs/1706.02216, 2017. Disponível em: http://arxiv.org/abs/1706.02216.

HENAFF, M.; BRUNA, J.; LECUN, Y. Deep convolutional networks on graph-structured data. **CoRR**, abs/1506.05163, 2015. Disponível em: http://arxiv.org/abs/1506.05163.

INTRODUCTION of neighbor sampling for GNN training¶. 2018. Disponível em: https://docs.dgl.ai/en/0.7.x/tutorials/large/L0_neighbor_sampling_overview.html.

IZADI, M. R. *et al.* Optimization of graph neural networks with natural gradient descent. **CoRR**, abs/2008.09624, 2020. Disponível em: https://arxiv.org/abs/2008.09624.

JAIN, A. *et al.* **Structural-RNN: Deep Learning on Spatio-Temporal Graphs**. 2015. Comment: CVPR 2016 (Oral). Disponível em: http://arxiv.org/abs/1511.05298.

JAIN, A. *et al.* Structural-rnn: Deep learning on spatio-temporal graphs. In: **CVPR**. IEEE Computer Society, 2016. p. 5308–5317. ISBN 978-1-4673-8851-1. Disponível em: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7776647.

JAOKAR, A. **Three-way data splits (training, test and validation) for model selection and performance estimation**. 2019. Disponível em: https://www.datasciencecentral.com/profiles/blogs/three-way-data-splits-training-test-and-validation-for-model.

KIPF, T. N.; WELLING, M. Semi-supervised classification with graph convolutional networks. **arXiv preprint arXiv:1609.02907**, 2016.

KIPF, T. N.; WELLING, M. Variational graph auto-encoders. **CoRR**, abs/1611.07308, 2016. Disponível em: http://arxiv.org/abs/1611.07308.

LANDHUIS, E. Scientific literature: Information overload. **Nature**, v. 535, n. 7612, p. 457–458, Jul 2016. ISSN 1476-4687. Disponível em: https://doi.org/10.1038/nj7612-457a.

LI, L. *et al.* Massively parallel hyperparameter tuning. **CoRR**, abs/1810.05934, 2018. Disponível em: http://arxiv.org/abs/1810.05934.

LI, Y. *et al.* Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In: **ICLR**. [*S.l.*]: OpenReview.net, 2018.

LIAW, R. *et al.* Tune: A research platform for distributed model selection and training. **arXiv preprint arXiv:1807.05118**, 2018.

LUAN, S. *et al.* **Is Heterophily A Real Nightmare For Graph Neural Networks To Do Node Classification?** 2021.

Micheli, A. Neural network for graphs: A contextual constructive approach. **IEEE Transactions on Neural Networks**, v. 20, n. 3, p. 498–511, March 2009. ISSN 1045-9227.

PEROZZI, B.; AL-RFOU', R.; SKIENA, S. Deepwalk: Online learning of social representations. **CoRR**, abs/1403.6652, 2014. Disponível em: http://arxiv.org/abs/1403.6652.

RAHMAN, M. Search engines going beyond keyword search: A survey. **International Journal of Computer Applications**, v. 75, p. 1–8, 08 2013.

Scarselli, F. *et al.* The graph neural network model. **IEEE Transactions on Neural Networks**, v. 20, n. 1, p. 61–80, Jan 2009. ISSN 1045-9227.

SHEN, X. *et al.* Discrete network embedding. In: LANG, J. (Ed.). **Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden**. ijcai.org, 2018. p. 3549–3555. ISBN 978-0-9992411-2-7. Disponível em: http://www.ijcai.org/proceedings/2018/.

SHERVASHIDZE, N. *et al.* Weisfeiler-lehman graph kernels. **J. Mach. Learn. Res**, v. 12, p. 2539–2561, 2011. Disponível em: http://dl.acm.org/citation.cfm?id=2078187.

SPERDUTI, A.; STARITA, A. Supervised neural networks for the classification of structures. **IEEE Transactions on Neural Networks**, v. 8, n. 3, p. 714–735, 1997.

SUN, K.; ZHU, Z.; LIN, Z. **AdaGCN: Adaboosting Graph Convolutional Networks into Deep Models**. 2021.

TRAN, P. V. Learning to make predictions on graphs with autoencoders. In: IEEE. **2018 IEEE 5th international conference on data science and advanced analytics (DSAA)**. [*S.l.*], 2018. p. 237–245.

VISHWANATHAN, S. V. N. *et al.* Graph kernels. **CoRR**, abs/0807.0093, 2008. Disponível em: http://arxiv.org/abs/0807.0093.

WANG, F.; ZHANG, C. Label propagation through linear neighborhoods. **IEEE Transactions on Knowledge and Data Engineering**, v. 20, n. 1, p. 55–67, 2008.

WANG, H.; LESKOVEC, J. **Unifying Graph Convolutional Neural Networks and Label Propagation**. 2020.

WANG, M. *et al.* Deep graph library: A graph-centric, highly-performant package for graph neural networks. **arXiv preprint arXiv:1909.01315**, 2019.

WU, Z. *et al.* A comprehensive survey on graph neural networks. **CoRR**, abs/1901.00596, 2019. Disponível em: http://arxiv.org/abs/1901.00596.

YAN, S.; XIONG, Y.; LIN, D. Spatial temporal graph convolutional networks for skeleton-based action recognition. **CoRR**, abs/1801.07455, 2018. Disponível em: http://arxiv.org/abs/1801.07455.

YANG, Z.; COHEN, W. W.; SALAKHUTDINOV, R. Revisiting semi-supervised learning with graph embeddings. **CoRR**, abs/1603.08861, 2016. Disponível em: http://arxiv.org/abs/1603.08861.

ZHANG, M. *et al.* An end-to-end deep learning architecture for graph classification. In: MCILRAITH, S. A.; WEINBERGER, K. Q. (Ed.). **Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018**. [*S.l.*]: AAAI Press, 2018. p. 4438–4445.