CEFET/RJ - CENTRO FEDERAL DE EDUCACAÇÃO TECNOLÓGICA CELSO SUCKOW DA FONSECA

Aprendizado de Representações em Grafos por Redes Neurais: uma Análise Comparativa

Augusto José Moreira da Fonseca

Prof. Orientador:

Eduardo Bezerra da Silva, D.Sc.

Rio de Janeiro,

Dezembro de 2020

CEFET/RJ - CENTRO FEDERAL DE EDUCACAÇÃO TECNOLÓGICA CELSO SUCKOW DA FONSECA

Aprendizado de Representações em Grafos por Redes Neurais: uma Análise Comparativa

Augusto José Moreira da Fonseca

Projeto final apresentado em cumprimento às normas do Departamento de Educação Superior do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, CEFET/RJ, como parte dos requisitos para obtenção do título de Tecnólogo em Sistemas para Internet.

Prof. Orientador:
Eduardo Bezerra da Silva, D.Sc.

Rio de Janeiro,
Dezembro de 2020

Ficha catalográfica elaborada pela Biblioteca Central do CEFET/RJ

F676 Fonseca, Augusto José Moreira da
Aprendizado de representações em grafos por redes neurais: uma análise comparativa / Augusto José Moreira da Fonseca – 2020.
x, 31f. : il. color. ; enc.

Projeto Final (Graduação) Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, 2020.
Bibliografia: f. 29-31
Orientador: Eduardo Bezerra da Silva

1. Computação. 2. Redes Neurais. 3. Teroria dos grafos.
I. Silva, Eduardo Bezerra da (Orient.). II. Título.

CDD 004

Elaborada pelo bibliotecário Leandro Mota de Menezes – CRB/7 nº 5281

DEDICATÓRIA

A Deus pela força e sabedoria. A minha querida esposa e filhos pelo apoio, amor e compreensão. Aos meus professores pelas orientações e ensinamentos.

AGRADECIMENTOS

Agradece-se ao professor Eduardo Bezerra pelos ensinamentos e orientação que tornaram possível a conclusão deste trabalho; aos professores da Escola de Informática & Computação do CEFET/RJ pela minha formação profissional e acadêmica; à Escola de Informática & Computação do CEFET/RJ por ceder os computadores que processaram os experimentos deste trabalho.

RESUMO

Muitos conjuntos de dados reais podem ser representados por uma estrutura em grafo. Pesquisas recentes têm buscado aplicar redes neurais artificiais a dados estruturados em grafos com o objetivo de treinar modelos capazes de inferir padrões desconhecidos. Apesar das *Graph Neural Networks* serem atualmente a abordagem comum para o treinamento destes modelos, sua desvantagem encontra-se no custo computacional em termos de tempo de treinamento e consumo de memória. Essa desvantagem se agrava em função da natureza de alguns conjuntos de dados, que podem alcançar proporções na ordem de bilhões de objetos e relações. O custo computacional das *Graph Neural Networks* tipicamente está relacionado ao aprendizado de representações dos objetos, que incorpora não só as informações do próprio objeto mas também as informações dos objetos relacionados. Em geral, *Graph Neural Networks* realizam esse aprendizado por meio de processamento recursivo além de precisar armazenar em memória várias instâncias de representações para um mesmo objeto. *Autoencoder Neural Networks* são redes que podem ser treinadas para aprender representações com baixo custo computacional, porém com a desvantagem de não incorporar as relações entre os objetos. Este trabalho realizou uma análise experimental comparativa utilizando as duas arquiteturas de redes neurais mencionadas para geração de representações em grafos em tarefas de classificação. Avaliamos essas arquiteturas em termos de tempo de treinamento e inferência, consumo de memória e poder preditivo, utilizando os conjuntos de dados Cora, Pubmed e Reddit.

Palavras-chaves: inferência em grafos; redes neurais; redes de convolução em grafos

ABSTRACT

Many actual data sets can be represented by a graphical structure. Recent research has sought to apply artificial neural networks to data structured in graphs in order to train models capable of inferring unknown patterns. Although Graph Neural Networks are currently the common approach for training these models, their drawbacks lies in the computational cost in terms of training time and memory consumption. This drawback is aggravated by the nature of some data sets, which can reach proportions in the order of billions of objects and relationships. The computational cost of Graph Neural Networks is typically related to the learning object embeddings, which incorporates not only the information of the object itself but also the information of related objects. In general, Graph Neural Networks do this learning through recursive processing in addition to having to store in memory several instances of embeddings for the same object. Autoencoder Neural Networks are networks that can be trained to learn embeddings with low computational cost, but with the drawback of not incorporating the relationships between objects. This work performed a comparative experimental analysis using the two mentioned neural network architectures for generating graph representations in classification tasks. We evaluated these architectures in terms of training time and inference, memory consumption and predictive power, using the Cora, Pubmed and Reddit data sets.

Keywords: graph inference; neural networks; graph convolution networks

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

AENN	*Autoencoder Neural Network*	1, 2, 3, 11, 12, 13, 16, 17, 18, 19, 23, 25, 26, 27, 28
CNN	*Convolutional Neural Network*	13, 14
CONVGNN	*Convolutional Graph Neural Network*	13, 14, 15, 16, 17
FFNN	*Feed Forward Neural Network*	10
GAE	*Graph Autoencoder*	14
GNN	*Graph Neural Network*	1, 2, 3, 27
MLP	*Multilayer Perceptron*	9, 10, 11
MSE	*Mean Squared Error*	19
RECGNN	*Recurrent Graph Neural Network*	14
RELU	*Rectified Linear Unit*	8, 14, 18, 19
RNA	Rede Neural Artificial	7
STGNN	*Spatial-Temporal Graph Neural Network*	14

Capítulo 1

Introdução

1.1 Contextualização

Grafo é uma estrutura matemática capaz de representar objetos e as relações entre estes. Diversos conjuntos de dados podem ser representados por grafos como por exemplo o mapa rodoviário de uma cidade, indivíduos em uma rede social ou citações entre artigos. Tal estrutura dispõe de teorias e algoritmos [McGregor, 2014; Herman et al., 2000] já formulados na literatura que podem ser usados para sua análise [Lehman and Leighton, 2004].

O aprendizado profundo [Goodfellow et al., 2016] é um subcampo do aprendizado de máquina que investiga técnicas para simular o comportamento do cérebro biológico. Recentemente produziu avanços em tarefas como reconhecimento visual, reconhecimento de voz e processamento de linguagem natural. Dentre os modelos empregados no aprendizado profundo, as redes neurais artificiais tem sido utilizadas para resolver uma variedade de tarefas.

Diante destes dois contextos, observa-se uma recente onda de interesse em pesquisas sobre a aplicação de redes neurais artificiais a dados estruturados em grafos [Kipf and Welling, 2016; Ying et al., 2018; Zhuang and Ma, 2018]. A abordagem comum para realizar o aprendizado de modelos neurais sobre grafos é por meio das *Graph Neural Network* (GNN) [Krizhevsky et al., 2012]. Nesta abordagem, as GNNs são aplicadas nos dados do grafo de entrada de forma a aprender as representações dos objetos a partir de suas características e relações, as quais são utilizadas posteriormente, por exemplo, em tarefas de predição ou de classificação.

O aprendizado de representações não é algo exclusivo das GNNs. Por exemplo, uma *Autoencoder Neural Network* (AENN) também pode ser utilizada nessa tarefa. Como vantagem, AENNs possuem arquitetura mais simples e menor consumo de memória em comparação as GNNs. Por outro lado, sua desvantagem reside no fato de que o aprendizado somente leva em consideração as características dos objetos, descartando as informações oriundas das relações entre eles. Por meio das GNNs, é possível treinar modelos para gerar representações de baixa dimensionalidade dos vértices de um grafo, no qual as informações sobre os atributos e vizinhança de cada vértice podem ser representadas como um vetor denso. A importância dessa

pesquisa apareceu em várias aplicações, como previsão de propriedades moleculares [Gilmer et al., 2017], análise de citações [Kipf and Welling, 2016], sistemas de recomendação [Ying et al., 2018], modelagem de movimentos humanos e interações entre objetos físicos [Jain et al., 2016], descoberta de drogas [Feinberg et al., 2018] , extração molecular de impressões digitais [Duvenaud et al., 2015] e verificação de programas computacionais [Li et al., 2015].

1.2 Motivação

Em geral, a arquitetura de uma GNN é composta por várias camadas. Cada camada é responsável por gerar a representação de cada vértice do grafo em níveis de abstração que aumentam conforme o nível de profundidade da rede. As representações geradas no final da rede podem ser passadas como entrada para métodos de aprendizado de máquina para realizar, por exemplo, tarefas de predição ou de classificação.

Tal arquitetura conduz a altos custos computacionais em termos de consumo de memória e tempo de treinamento. O consumo de memória torna-se elemento limitador devido ao fato de que as representações dos vértices devem ser armazenadas distintamente para cada camada da rede. O tempo de aprendizado é prejudicado devido à característica recursiva das GNNs, em que a representação de um vértice depende do processamento de representações dos vértices vizinhos na camada anterior e assim sucessivamente.

Dependendo da natureza do conjunto de dados, sua representação em grafo pode alcançar proporções na ordem de bilhões de objetos e relações. Tal proporção agrava significativamente os problemas anteriormente citados. Apesar de Chiang et al. [2019] ter minimizado tal problema por meio da utilização de amostras no processo de treinamento, a acurácia desejada somente foi obtida com o aumento do número de camadas da rede o que naturalmente conduz a mais custo computacional para convergência.

Uma vez que AENNs somente processam as informações dos vértices e, portanto, não levam em consideração o contexto topológico do vértice no grafo, é de se esperar que o poder preditivo utilizando uma GNN seja maior que com uma AENN. A questão é: quanto maior e a que custo computacional? Avaliar essas duas arquiteturas em contextos de aprendizado comuns pode gerar resultados que sirvam para apoiar na decisão de quando uma arquitetura torna-se mais adequada que a outra.

1.3 Objetivos

Este trabalho tem como proposta realizar uma análise experimental comparativa utilizando as arquiteturas GNN e AENN no aprendizado de representações, em tarefas de classificação. A análise comparativa será realizada em termos de tempo de treinamento e inferência, consumo de memória e poder preditivo. Por meio dessa análise, busca-se explorar o *trade-off* entre custo computacional e poder preditivo entre as arquiteturas citadas. Uma vez que a arquitetura GNN explora a relação entre os objetos, os conjuntos de dados empregados deverão permitir sua representação em grafos simples.

1.4 Metodologia

Para realizar a análise comparativa entre as arquiteturas GNN e AENN, definimos um contexto de aprendizado comum para ambas. A ideia é que tanto os dados de entrada quanto o método de aprendizado para a tarefa de classificação fossem os mesmos, alterando-se apenas a arquitetura de aprendizado de representação. Dessa forma, foi possível analisar os impactos no custo computacional e poder preditivo quando uma ou outra arquitetura era utilizada.

Para estender a análise comparativa, foram utilizados 3 conjuntos de dados que permitem realizar tarefas de classificação. Estes foram selecionados dentre os mais utilizados na literatura em GNNs. Um pré-processamento foi realizado para que os conjuntos de dados pudessem servir de entrada para as duas arquiteturas em análise sem qualquer modificação.

Como arquitetura GNN foi selecionada uma dentre as mais relevantes na literatura, o ClusterGCN [Chiang et al., 2019]. Além da sua relevância, a justificativa desta escolha baseou-se na disponibilidade de *download* do código-fonte e na abrangência em tarefas de classificação. Como forma de viabilizar o contexto de aprendizado comum, a arquitetura AENN foi implementada. Foi aplicado na AENN o mesmo método de aprendizado para tarefa de classificação utilizado pelo ClusterGCN.

Com o ambiente devidamente preparado, foram realizados os experimentos. Cada conjunto de dados foi utilizado duas vezes na tarefa de classificação, cada vez utilizando uma arquitetura proposta distinta. A comparação do desempenho das duas arquiteturas foi avaliada em termos de tempo de treinamento e inferência, consumo de memória e poder preditivo.

1.5 Organização dos Capítulos

Além desta introdução, o trabalho se divide em mais quatro outros Capítulos. O Capítulo 2 apresenta a fundamentação teórica e os trabalhos relacionados. A modelagem da análise proposta é detalhada no Capítulo 3. O Capítulo 4 exibe os resultados alcançados na Avaliação Experimental. O Capítulo 5 aborda as conclusões obtidas e as oportunidades de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este Capítulo aborda a fundamentação teórica e os trabalhos relacionados. Na Seção 2.1 são apresentados conceitos relevantes acerca de grafos. A Seção 2.2 aborda as redes neurais que foram utilizadas neste trabalho, bem como conceitos sobre aprendizado de representações em grafos. Por fim a Seção 2.3 apresenta os trabalhos relacionados.

2.1 Grafos

Fundamentos sobre grafos podem ser encontrados em vasta literatura disponível. As definições apresentadas a seguir têm como fonte o e-book *Mathematics for Computer Science* Lehman and Leighton [2004]. Um grafo G é constituído de um par de conjuntos (V,E), formalmente definido por $G = (V,E)$ onde:

- V é um conjunto não vazio cujos elementos são chamados de **nós** ou **vértices**.
- E é uma coleção de subconjuntos de dois elementos $v \in V$ chamados de **arcos** ou **arestas**.

A Figura 1 mostra um exemplo de grafo. Os vértices são representados pelos pontos pretos e as arestas pelas linhas. Dessa forma, os conjuntos de vértices e arestas do grafo da Figura 1 são denotados por:

- $V = \{A,B,C,D,E,F,G,H,I\}$
- $E = \{\{A,B\},\{A,C\},\{B,D\},\{C,D\},\{C,E\},\{E,F\},\{E,G\},\{H,I\}\}$

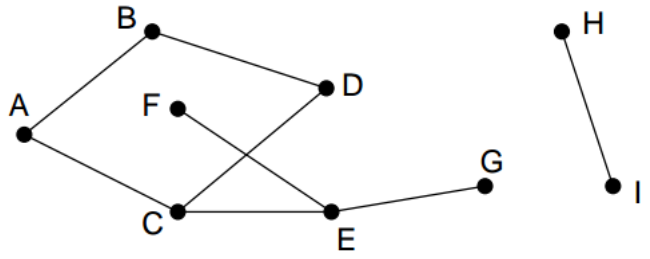

Figura 1: Exemplo de grafo. Pontos pretos representam os vértices. Linhas representam as arestas. Lehman and Leighton [2004]

A notação $\{A,B\}$ é então utilizada para denotar uma aresta entre os vértices A e B. $\{A,B\}$ e $\{B,A\}$ denotam a mesma aresta. Dois vértices em um grafo são ditos **adjacentes** quando existe uma aresta entre eles. O número de arestas incidentes em um vértice é chamado **grau**. Na Figura 1, A é adjacente a B e B é adjacente a D. A aresta $\{A,C\}$ é incidente aos vértices A e C. O vértice H tem grau 1, D tem grau 2 e E tem grau 3.

Um **subgrafo** $G' = (V',E')$ é dito subgrafo de G se $(V' \subseteq V)$ e $(E' \subseteq E)$. Uma vez que um subgrafo é também um grafo, os extremos de uma aresta $e \in E'$ devem ser vértices $v \in V'$.

Existem variações de grafos tais como **multigrafos** (grafos que podem possuir mais de uma aresta para um mesmo par de vértices) e **grafos direcionados** (grafos onde a navegabilidade entre dois vértices é definida pela aresta). Também como característica de alguns grafos, um **laço** é uma aresta que conecta um vértice a ele próprio. Grafos ditos **simples** são grafos que não possuem as variações mencionadas. A partir deste ponto, o termo grafo deve ser entendido como grafo simples, exceto quando explicitado o contrário.

Grafos podem representar todo tipo de objetos e relacionamentos entre estes. Seus vértices e/ou arestas podem possuir atributos que denotem determinados aspectos do domínio que se quer estruturar. Por exemplo, um grafo que representa as conexões aéreas de um país pode ter seus vértices atribuídos de um índice único que denote um determinado aeroporto e suas arestas podem ter um valor atribuído que represente o tempo de voo entre os aeroportos. Esse recurso é largamente utilizado em grafos e por meio dele vários problemas do mundo real podem ser computacionalmente solucionados. Comumente, tais atributos em vértices e/ou arestas de um grafo são chamados **rótulos**. A Figura 2 mostra dois grafos com vértices rotulados.

Figura 2: Rótulos em grafos. Lehman and Leighton [2004]

Uma das formas de se representar um grafo é por sua **matriz de adjacência**. A matriz de adjacência A de um grafo com n vértices possui dimensões $n \times n$. Uma entrada $a_{i,j}$ (linha i, coluna j) é igual a 1 se existe uma aresta entre os vértices v_i e v_j, 0 caso contrário. A diagonal principal da matriz é toda entrada $a_{i,j}$, com $i = j$. Em grafos simples, a diagonal principal é sempre igual a 0. Um grafo simples é representado por uma matriz de adjacência que é

simétrica: $a_{i,j} = a_{j,i}\ \forall i, j$. A Figura 3 mostra um exemplo de matriz de adjacência.

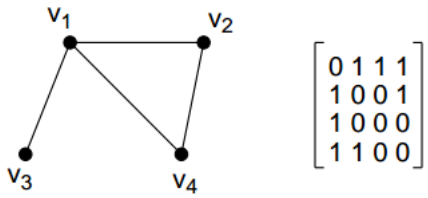

Figura 3: Matriz de Adjacência. Lehman and Leighton [2004]

A **lista de arestas** de um grafo simples G consiste de uma lista $E = \{e_1, e_2, \ldots, e_m\}$ onde m representa o número de arestas do grafo G e cada elemento e_i consiste de um par não ordenado de vértices $v \in V$ aos quais a aresta i é incidente. O grafo da Figura 3 possui a seguinte lista de arestas: $\{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_4\}\}$.

2.2 Redes Neurais Artificiais

Rede Neural Artificial (RNA) é uma arquitetura computacional baseada no cérebro biológico que tem por objetivo processar informação. O surgimento das RNAs pode ser atribuído à complexidade de implementação de sistemas para resolução de problemas complexos baseados em regras heurísticas. Como forma de transpor tal dificuldade, as RNAs são implementadas de forma a receber como entrada dados de exemplo e por meio de sucessivas observações aprender determinados padrões sobre eles. Tal etapa é comumente chamada de *treinamento* e baseia-se no aprendizado humano que evolui por meio das experiências. Os padrões aprendidos geram um modelo capaz de inferir sobre dados não utilizados durante o treinamento, mas que possuam padrões similares. A seguir, são descritos os componentes e o funcionamento das RNAs que foram exploradas neste trabalho, que têm como fonte o tutorial de aprendizado profundo da Universidade de Stanford [Stanford, 2019].

2.2.1 Perceptron

A unidade básica de processamento de uma rede neural artificial é chamada de *perceptron*. O *perceptron* foi proposto por Rosenblatt [1958] como aprimoramento do trabalho anterior de McCulloch and Pitts [1943] e sua estrutura foi inspirada no neurônio biológico. A Figura 4 mostra um exemplo de *perceptron*.

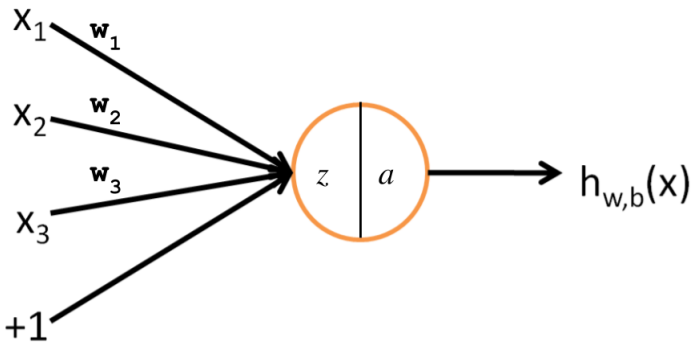

Figura 4: Perceptron. Stanford [2019]

Considerando o perceptron da Figura 4, os valores x_1, x_2 e x_3 são passados como entrada. Cada um desses valores é multiplicado por seus respectivos pesos w_1, w_2 e w_3 e a soma destes produtos com o termo de interceptação b (+1) gera o valor agregado z, que sofre a transformação por meio da função $f(\cdot)$ gerando o valor a. Esse processamento é resumido pela Equação 2.1

$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b), \tag{2.1}$$

onde $f : \Re \mapsto \Re$ é chamada de *função de ativação*. As funções de ativação $f(\cdot)$ comumente utilizadas são a *Sigmoid* (Equação 2.2) que tem como resultado valores entre [0,1], a *hyperbolic tangent* (Equação 2.3) que tem como resultado valores entre [-1,1] e a *Rectified Linear Unit* (ReLU) (Equação 2.4) que retorna o maior valor entre $(0, z)$.

$$f(z) = \frac{1}{1+\exp(-z)}, \tag{2.2}$$

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{2.3}$$

$$f(z) = \max(0, z), \tag{2.4}$$

Denotamos $h_{W,b}(x)$ a hipótese que o *perceptron* calcula tendo como entrada informações sobre x (x_1, x_2 e x_3), utilizando-se dos parâmetros W e b que devem ser aprendidos. Denotamos $x^{(i)}$ como o i-ésimo exemplo de um conjunto de dados e $y^{(i)}$ o seu respectivo valor alvo, o qual chamamos de *rótulo*. Em uma tarefa de classificação podemos definir $y^{(i)} = 1$ se $x^{(i)}$ pertence a uma determinada classe arbitrária, 0 caso contrário. Como unidade única de processamento, o *perceptron* somente é capaz de aprender funções lineares. Para o aprendizado de funções

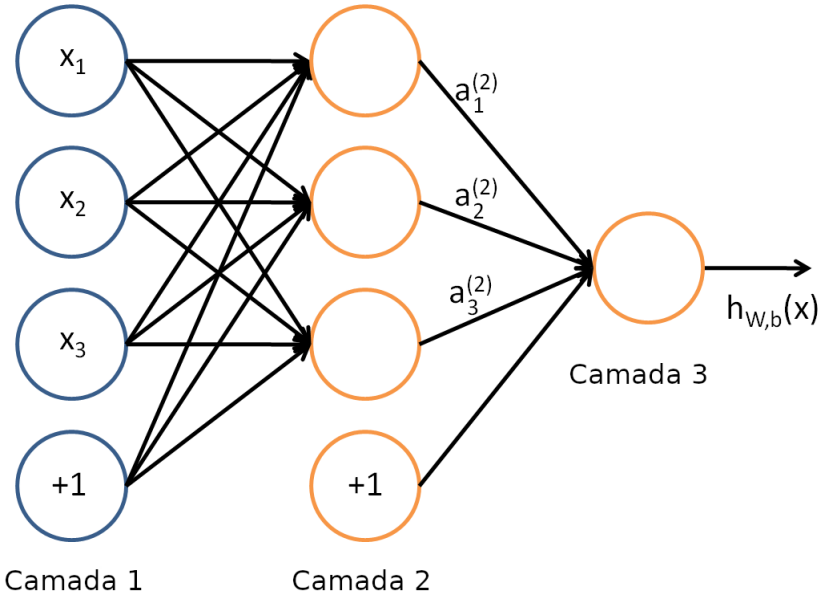

Figura 5: Multilayer Perceptron. Stanford [2019]

não lineares, as unidades *perceptrons* precisam ser agrupadas em camadas de processamento formando o que chamamos de *Multilayer Perceptron* (MLP).

2.2.2 Multilayer Perceptron

Para o aprendizado de funções não lineares os *perceptrons* são agrupados em camadas. Tal arquitetura é organizada de forma que a saída de um neurônio seja a entrada de outro neurônio da camada seguinte. A Figura 5 mostra um exemplo de MLP.

Neste exemplo, denotamos $W_{ij}^{(l)}$ como o peso associado à conexão entre a unidade j na camada l e a unidade i na camada $l+1$, com $l = \{1,2,...,L\}$. Denotamos o termo de interceptação associado à unidade i na camada $l+1$ por $b_i^{(l)}$ o qual, no contexto das MLPs, é chamado *bias*. Denotamos $a_i^{(l)}$ como a ativação da unidade i na camada l. Para $l = 1$, usamos $a_i^{(1)} = x_i$ para denotar a i-ésima entrada. A computação realizada pela MLP é então representada pelas Equações 2.5, 2.6, 2.7 e 2.8.

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \tag{2.5}$$

$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \tag{2.6}$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \tag{2.7}$$

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)}) \tag{2.8}$$

Denotamos $z_i^{(l)}$ como a soma ponderada da unidade i na camada l, incluindo o *bias*. Por exemplo, $z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(1)} x_j + b_i^{(1)}$, de modo que $a_i^{(l)} = f(z_i^{(l)})$. Considerando a função de ativação $f(\cdot)$ aplicada a vetores na forma elemento a elemento (*i.e.*, $f([z_1^{(l+1)}, z_2^{(l+1)}, z_3^{(l+1)}]) = [f(z_1^{(l+1)}), f(z_2^{(l+1)}), f(z_3^{(l+1)})]$), chamamos de *propagação adiante* a computação realizada pela MLP que pode ser então simplificada pelas Equações 2.9 e 2.10.

$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)} \tag{2.9}$$

$$a^{(l+1)} = f(z^{(l+1)}) \tag{2.10}$$

Essa classe de rede é conhecida como *Feed Forward Neural Network* (FFNN). Apesar do exemplo simples, FFNNs podem variar em número de camadas, número de unidades por camada e até mesmo número de unidades de saída. Em uma arquitetura onde queremos que a rede calcule a probabilidade de um exemplo $x^{(i)}$ pertencer a um número n de classes distintas, podemos ter $y^{(i)} \in \Re^n$.

O treinamento de uma FFNN consiste então em aprender os valores de (W, b) em cada camada, de modo que uma entrada $x^{(i)}$ seja mapeada pela rede para o rótulo $y^{(i)}$. Vale ressaltar que para cada camada (à exceção da camada L), existe um par $(W^{(l)}, b^{(l)})$ na forma de matrizes. O gradiente descendente é utilizado nessa etapa do treinamento chamada *retropropagação*, a qual atualiza os valores de (W, b) conforme $h_{W,b}(x^{(i)})$ se aproxima ou se afasta de $y^{(i)}$. A distância entre esses dois valores é chamada de *custo* e é denotada pela Equação 2.11.

$$J(W, b; x, y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2 \tag{2.11}$$

Considerando um conjunto de dados com m exemplos, o custo total da rede pode ser calculado pelas Equações 2.12 e 2.13.

$$J(W, b) = \frac{1}{m} \sum_{i=1}^{m} J(W, b; x^{(i)}, y^{(i)}) \tag{2.12}$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2 \right) \tag{2.13}$$

O objetivo do treinamento é então minimizar $J(W, b)$. O processo de treinamento inicia pela atribuição de valores aleatórios próximos de zero para cada parâmetro $W_{ij}^{(l)}$ e $b_i^{(l)}$. Após

executar a propagação adiante com os m exemplos do conjunto de dados, $J(W,b)$ é calculado. A partir deste ponto, inicia-se a retropropagação que nada mais é do que a etapa de atualização dos parâmetros. A atualização é dada pelas Equações 2.14 e 2.15

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) \tag{2.14}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b), \tag{2.15}$$

onde α é chamado de *taxa de aprendizado* e é considerado um hiperparâmetro que deve ser ajustado por meio de alguma heurística e a derivada da função de custo $J(W,b)$ é calculada pelas Equações 2.16 e 2.17.

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)}) \tag{2.16}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J(W,b;x^{(i)},y^{(i)}) \tag{2.17}$$

Excluindo a etapa inicial de atribuição aleatória dos parâmetros, a rede repete a propagação adiante e a retropropagação até que haja a convergência. Entende-se por convergência o valor de $J(W,b)$ chegar a um valor mínimo tal que novas iterações modifiquem irrisoriamente o valor de $J(W,b)$.

2.2.3 Autoencoder Neural Network

Um conceito importante acerca do treinamento de uma rede neural artificial é que quando utilizamos conjuntos de dados com exemplos rotulados (ou seja, cada exemplo $x^{(i)}$ possui um rótulo associado $y^{(i)}$) e fazemos uso dos rótulos no processo de otimização da rede, dizemos que o treinamento ocorre de forma *supervisionada*. Quando o conjunto de dados não possui rótulos para os exemplos, chamamos esse treinamento de *não supervisionado*. Baseado na arquitetura MLP, AENN é uma classe de rede neural que pode ser treinada para aprender representações dos exemplos de um conjunto de dados de forma não supervisionada. Em suma, o treinamento de uma AENN visa aplicar retropropagação, utilizando como rótulos os próprios valores de entrada, ou seja, $y^{(i)} = x^{(i)}$. A Figura 6 mostra um exemplo de AENN.

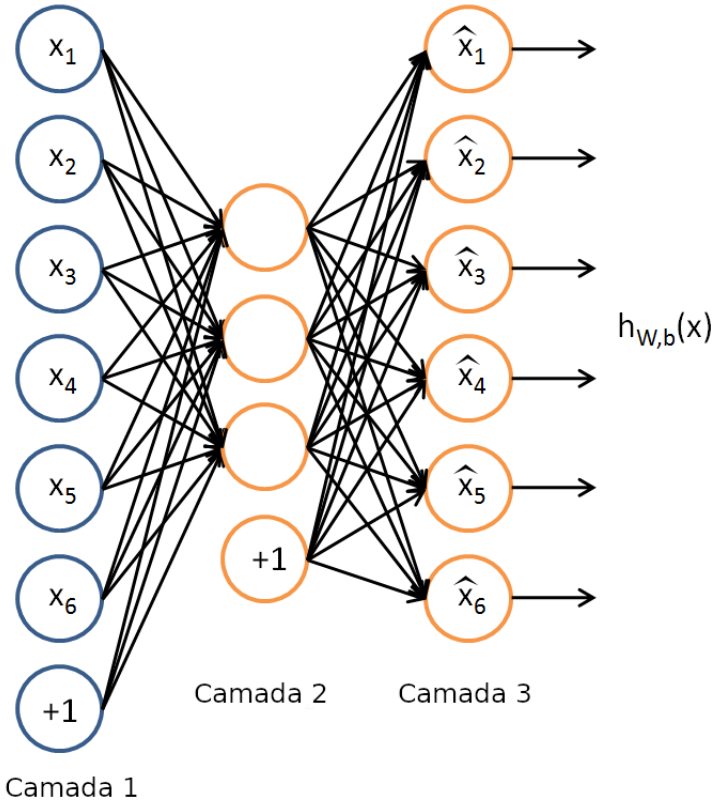

Figura 6: Autoencoder Neural Network. Stanford [2019]

Uma vez que a rede é treinada para apresentar na saída uma aproximação dos valores de entrada, ou seja $h_{W,b}(x) \approx x$, a função que a rede aprende por ser entendida como uma aproximação da função identidade. Em uma rede com 3 camadas, quando utilizamos $W^{(2)} = W^{(1)T}$, dizemos que a rede utiliza o mecanismo de *pesos atados*, o que resulta em menos parâmetros a serem otimizados [Bezerra, 2016] além de prevenir soluções degeneradas. A característica importante das AENNs reside na sua camada intermediária já que o vetor $a^{(2)}$ é considerado como o vetor de características que representa o valor de entrada x, considerando uma rede com 3 camadas. Quando a camada intermediária possui uma quantidade de unidades menor que a camada de entrada, dizemos que o vetor $a^{(2)}$ é uma *representação de baixa dimensionalidade* de x.

Essas representações podem servir, por exemplo, de entrada para métodos de aprendizado de máquina em tarefas de aprendizado supervisionado. A capacidade de uma AENN gerar boas representações de baixa dimensionalidade está diretamente ligada à natureza dos dados, uma vez que a rede aprende as correlações entre os dados de entrada e por isso é capaz de "comprimi-los". Apesar do apresentado até agora, AENNs também podem ter arquiteturas na qual o vetor $a^{(2)}$ tem dimensão maior que o vetor de entrada. Um exemplo disso é quando o

objetivo da AENN reside no aprendizado de representações esparsas. Para estes casos, técnicas distintas precisam ser aplicadas no treinamento.

2.2.4 Aprendizado de Representações em Grafos

Apesar de existirem distintas técnicas para aprendizado de representações em grafos [Hamilton et al., 2017b], a técnica de convolução é uma das mais comuns na literatura recente. Inspirada na técnica de convolução utilizada nas *Convolutional Neural Network* (CNN) [LeCun et al., 1995], a convolução em grafos consiste em gerar representações para um vértice por meio da agregação das informações dos vértices vizinhos locais. Esse é um processo tipicamente recursivo, uma vez que a cada iteração utiliza-se a informação agregada da iteração anterior.

No primeiro passo da iteração, cada vértice possui suas próprias informações, em geral um vetor de características acerca do domínio. Por exemplo, em um grafo que representa uma rede social e cada vértice representa um indivíduo, o vetor de características pode conter dados sobre o perfil do indivíduo na rede social (*e.g*, sexo, idade, gênero musical, etc). A cada iteração seguinte, o vetor de características dos vértices vizinhos são agregados por meio de uma função de agregação arbitrária. Ao final de cada iteração, cada vértice recebe um novo vetor que é uma combinação do seu atual vetor de características com o vetor de vizinhança agregado. Um detalhe importante nesse processo é que a cada nova iteração, cada vértice passa a agregar características de vizinhos cada vez mais distantes [Hamilton et al., 2017b].

O aprendizado de representações em grafos é portanto um processo que toma por vantagem a informação acerca das relações entre os objetos. Aplicando a convolução em grafos, é possível aprender representações que incorporam não só as informações de um vértice mas também as informações no contexto de sua vizinhança. Assim como nas AENNs, tais representações podem ser utilizadas em tarefas de análise em grafos tais como predição de conexão, agrupamento ou classificação.

Uma *Convolutional Graph Neural Network* (ConvGNN) é uma classe de rede neural comumente utilizada nessa tarefa. Sua arquitetura explora a estrutura dos dados organizados em rede, de forma a adaptar o que as CNNs executam em dados estruturados. Seja $G = (V, E)$ um grafo no qual $x^{(i)}$ é um vetor de características de dimensão d para o i-ésimo vértice. Seja $n = |V|$. O treinamento de uma ConvGNN envolve duas entradas:

- Uma matriz de características $X \in \Re^{n \times d}$ composta por vetores de características de cada

um dos n vértices em G.

- G, tipicamente na forma de uma matriz de adjacência.

A ConvGNN pode então ser treinada para gerar como saída uma matriz $\hat{Y} \in \Re^{n \times f}$, onde f seja o número de características, $f < d$. A maioria das ConvGNNs seguem uma arquitetura geral. Seja uma ConvGNN com L camadas. Seja $H(i)$ a i-ésima camada. A computação realizada na camada $l+1$ pode ser escrita como a Equação 2.18

$$H(l+1) = \phi(H(l), A), \tag{2.18}$$

onde ϕ é uma função de ativação para a camada $l+1$ (*e.g.*, ReLU, *Sigmoid*, etc). Na expressão recursiva acima, existem dois casos base:

- $H(1) = X$
- $H(L) = \hat{Y}$

Dada a arquitetura geral acima, específicas ConvGNNs diferem em como ϕ é escolhida e parametrizada.

2.3 Trabalhos Relacionados

Os primeiros trabalhos de aplicação de redes neurais em dados estruturados em grafos foram observados em Sperduti and Starita [1997], Gori et al. [2005] e Scarselli et al. [2008]. Desde então muitas abordagens novas surgiram inspiradas principalmente nos sucessos alcançados pelas CNNs. Em seu recente trabalho, Wu et al. [2019] propôs uma nova taxonomia de redes neurais em grafos, categorizando-as em 4 grupos: *Recurrent Graph Neural Network* (RecGNN), ConvGNN, *Graph Autoencoder* (GAE) e *Spatial-Temporal Graph Neural Network* (STGNN). Dentre os grupos acima citados, as ConvGNN foram as que mais tiveram avanços [Wu et al., 2019]. A ideia de adaptar as CNNs para aprender representações de alto nível em grafos conduziu a vários trabalhos os quais são divididos em duas categorias: as *Spectral-based* ConvGNN e as *Spatial-based* ConvGNN. Devido às limitações e custo computacional da primeira, o maior número de trabalhos e contribuições ocorreram na segunda. O interesse nas *Spatial-based* ConvGNN reside em suas vantagens em termos de flexibilidade, eficiência e capacidade de generalização comparadas às outras abordagens.

Micheli [2009] realizou o primeiro trabalho no contexto das *Spatial-based* ConvGNN no qual o processo de convolução consiste de somas sucessivas dos vetores de representações dos vértices vizinhos. Atwood and Towsley [2016] propôs um modelo que por meio de uma matriz de probabilidade de transição, os vértices vizinhos mais distantes do vértice convolvido contribuem com menos informações. Gilmer et al. [2017] descreve um *framework* genérico no qual as informações são passadas diretamente pelas arestas no processo de convolução.

A representação em grafo de certos conjuntos de dados pode alcançar largas proporções em termos de quantidade de vértices e arestas. Treinar *Spatial-based* ConvGNN conduz a altos custos em termos de consumo de memória e tempo de treinamento. Para resolver o problema de consumo de memória Hamilton et al. [2017a] propôs o *framework GraphSAGE* que adotou uma amostragem de vértices vizinhos ao vértice convolvido. A amostra foi definida com um número fixo de vértices amostrados uniformemente ao acaso. Nos casos onde um vértice não possui vizinhos suficientes, a amostragem é realizada com repetição. Ainda como forma de resolver a questão do tempo de treinamento do modelo, Hamilton et al. [2017a] também apresentou uma adaptação de forma que o processo de convolução ocorresse por *batchs*.

Apesar de trabalhos posteriores tentarem aprimorar as técnicas de amostragem dos vértices [Chen et al., 2018a,b], foi no trabalho de Chiang et al. [2019] que notou-se avanço significativo. Por meio do seu *framework* ClusterGCN, Chiang et al. [2019] utilizou algoritmos de agrupamento para particionar os vértices do grafo. Por meio desse pré-processamento, o treinamento foi então realizado por *batchs* que são partições de agrupamentos de vértices aleatoriamente selecionados. Essa técnica permitiu uma significativa diminuição do consumo de memória e tempo de treinamento, além de viabilizar a manipulação de grafos com mais de 2 milhões de vértices.

Capítulo 3

Desenvolvimento

Este Capítulo detalha a modelagem e os elementos que foram empregados neste trabalho. A Seção 3.1 apresenta uma visão geral da proposta. A implementação do ClusterGCN e da AENN são detalhadas nas Seções 3.2 e 3.3. A Seção 3.4 apresenta os hiperparâmetros testados, a forma de treinamento das redes e as métricas utilizadas para medição do custo computacional e poder preditivo. Por fim a Seção 3.5 detalha os conjuntos de dados utilizados no experimento.

3.1 Visão Geral

De modo geral as ConvGNNs geram em suas camadas intermediárias representações de baixa dimensionalidade para os vértices, que são passadas para camadas finais que tem, por exemplo, a tarefa de executar predição ou classificação. Tal arquitetura conduz a altos custos computacionais em termos de consumo de memória e tempo de treinamento. O consumo de memória reside no fato de que as representações dos vértices devem ser armazenadas para cada camada. Por exemplo, se considerarmos que uma ConvGNN possui L camadas de convolução, que cada iteração da rede processa n vértices e, por simplificação, que a representação de um vértice tem o mesmo tamanho d em todas as camadas, então o custo de memória somente para a tarefa de convolução será de $L \times n \times d$. Já o tempo de treinamento está diretamente relacionado ao processamento que deve ser executado bem como o tempo para convergência do modelo. Estes dois parâmetros também crescem conforme aumenta L, n e d. Apesar de Chiang et al. [2019] ter minimizado tal problema por meio da utilização de amostras no processo de treinamento, a acurácia desejada somente foi obtida com o aumento do número de camadas de convolução o que naturalmente conduz a mais processamento e tempo para convergência.

Visando explorar o *trade-off* entre custo computacional e poder preditivo, este trabalho busca comparar a utilização de representações geradas por meio das ConvGNN em relação as geradas por clássicas AENNs, em tarefas de classificação. A ideia então consiste em comparar o desempenho das duas arquiteturas em termos de tempo de treinamento e inferência, consumo de memória e poder preditivo. A Figura 7 ilustra uma visão geral desta proposta.

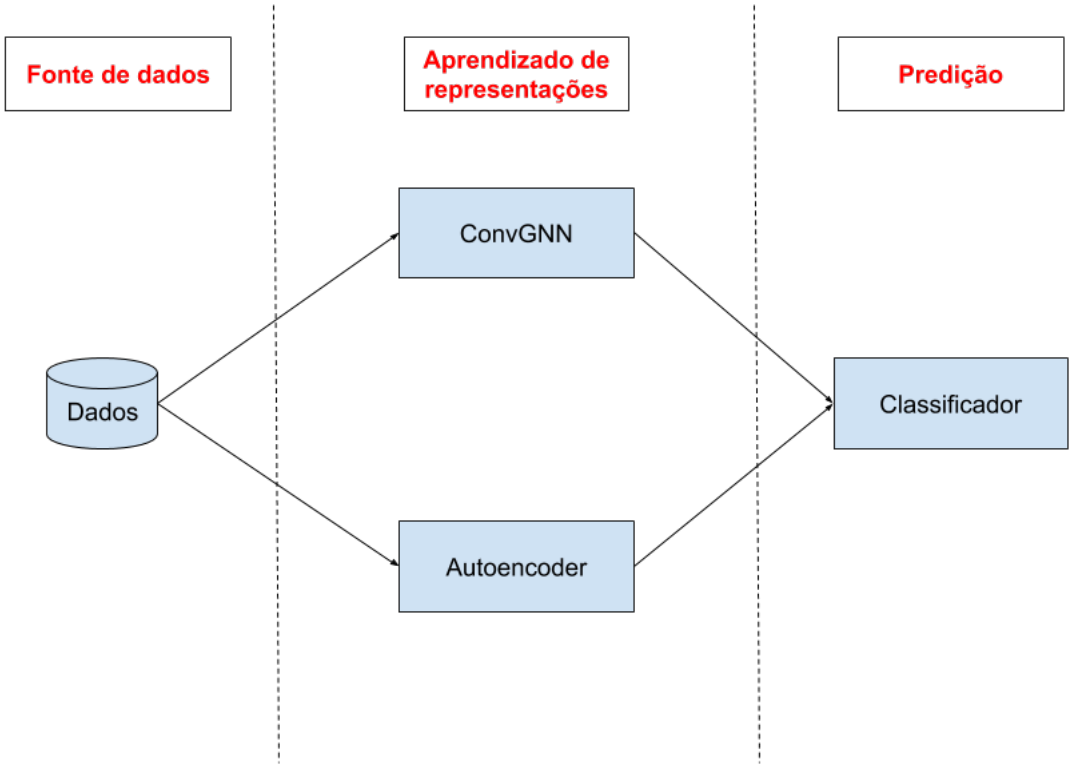

Figura 7: Visão geral da proposta. A mesma fonte de dados servirá de entrada para as duas arquiteturas de aprendizado de representações. As representações geradas serão passadas para o mesmo método de aprendizado para tarefas de classificação. O custo computacional e poder preditivo será comparado entre as duas arquiteturas.

Uma vez que AENNs somente processam as características dos vértices e portanto não levam em consideração o contexto topológico do vértice no grafo, é de se esperar que o poder preditivo de uma ConvGNN seja maior que a de uma AENN. A questão é: quanto maior e a que custo computacional? Para obter tais respostas, este trabalho visa realizar experimentos utilizando-se das duas arquiteturas porém com as mesmas fontes de dados.

3.2 ClusterGCN

O *framework* ClusterGCN foi empregado na arquitetura ConvGNN. A escolha por esse *framework* justifica-se pelas seguintes vantagens em relação aos trabalhos anteriores: *i)* obteve melhor acurácia em conjuntos de dados utilizados em trabalhos anteriores; *ii)* baixa complexidade de tempo e memória comparado aos trabalhos anteriores; *iii)* única arquitetura que conseguiu trabalhar com ConvGNN profunda e conjunto de dados com mais de 2 milhões de vértices e; *iv)* possui código fonte [Google, 2020] publicado no Github.

A implementação do ClusterGCN permite a parametrização de alguns hiperparâmetros de

treinamento. Dentre os relevantes para este trabalho pode-se citar: *i)* o número de épocas de treinamento; *ii)* o número de partições de agrupamentos de vértices para os conjuntos de treino, validação e teste; *iii)* a taxa de aprendizado da rede; *iv)* o número de partições de agrupamentos de vértices por *batch* de treinamento e; *v)* o número de camadas intermediárias da rede e a quantidade de neurônios para elas. Todos os hiperparâmetros configuráveis possuem valores padrão permitindo que o *framework* seja executado sem definições arbitrárias.

Além das camadas intermediárias que o ClusterGCN permite parametrizar, arbitrariamente a quantidade de neurônios nas camadas de entrada e saída possuem respectivamente as dimensões d (quantidade de características associadas a cada vértice) e $|C|$ (quantidade de classes possíveis as quais um vértice pode pertencer). A função de ativação empregada nas camadas intermediárias é a ReLU. Na camada $H(L)$ é aplicada a função de ativação *Softmax* que gera uma distribuição de probabilidade com valores entre $[0,1]$.

Uma vez que a camada $H(L)$ possui a mesma quantidade de neurônios que o número de classes possíveis, ou seja $|H(L)| = |C|$, e sendo $|C| < d$ então o vetor de características $x_L^{(i)}$ pode ser entendido como a representação de baixa dimensionalidade para um vértice $v^{(i)}$ de entrada da rede. Dessa forma, observa-se que o método de aprendizado para classificação implementado no ClusterGCN reduz-se à aplicação da função de ativação *Softmax* na camada $H(L)$, onde o índice do maior valor da distribuição de probabilidade gerada denota a classe predita pela rede para o vértice $v^{(i)}$ de entrada. A função de custo empregada na rede é a *Softmax Cross Entropy*.

3.3 Autoencoder Neural Network

A implementação da arquitetura AENN foi realizada de forma a ajustar-se às fontes de dados utilizadas pelo ClusterGCN, bem como empregar o mesmo método de aprendizado para a tarefa de classificação. Para tal, a rede foi implementada com 3 camadas (entrada, intermediária e saída) e de forma que a propagação adiante do sinal de entrada ocorresse em dois passos. No primeiro passo o sinal de entrada é propagado com o objetivo de gerar na saída a aproximação do sinal de entrada. No segundo passo, a rede aplica a função de ativação *Softmax* na camada intermediária que tem como objetivo realizar a tarefa de classificação.

As quantidades de neurônios por camada são de d para as camadas de entrada e saída e $|C|$ para a camada intermediária. Dessa forma, a camada intermediária torna-se análoga à camada $H(L)$ da arquitetura ClusterGCN, sendo portanto seu vetor $x_2^{(i)}$ considerado como a representação de baixa dimensionalidade para um vértice $v^{(i)}$ de entrada da rede. No primeiro passo

as funções de ativação para as camadas intermediária e saída são respectivamente a ReLU e a *Linear*. A função de custo empregada nesse passo é a *Mean Squared Error* (MSE).

No segundo passo é aplicado ao vetor $x_2^{(i)}$ a função de ativação *Softmax*. A ideia neste segundo passo é realizar a tarefa de classificação por meio da representação gerada para o vetor de entrada. Da mesma forma que na arquitetura ClusterGCN, a função de custo aqui empregada é a *Softmax Cross Entropy*.

Conforme detalhado acima, a propagação adiante da AENN implementada ocorre portanto em dois passos. Para que a rede possa então ajustar os parâmetros W, b foi necessário somar os custos calculados pelas duas funções de custo. Esse somatório é então passado para que o gradiente descendente ajuste os parâmetros da rede durante a retropropagação.

3.4 Hiperparâmetros, Treinamento e Métricas

Como primeiro passo para o treinamento, foram definidos os hiperparâmetros a serem testados para cada uma das arquiteturas. Dado que as arquiteturas possuem distintas estratégias de aprendizado, os hiperparâmetros também foram definidos de forma distinta. O número de épocas de treinamento foi fixado em 200 para todos os experimentos uma vez que testes preliminares mostraram ser um número suficiente para a convergência dos modelos. Da mesma forma, foi definido para todos os experimentos um valor fixo q de *Early Stopping* [Prechelt, 1998] que interrompe o treinamento caso o valor médio do erro de validação das q iterações de treinamento anteriores ultrapasse o valor da iteração corrente. A implementação desta técnica contribuiu para os objetivos deste trabalho uma vez que o tempo de treinamento e inferência é um dos elementos analisados. No caso dos experimentos com a arquitetura ClusterGCN, o número de partições de agrupamento de vértices para o conjunto de treino foi definido proporcionalmente ao número total de vértices de cada conjunto de dados de entrada. Para o conjunto de dados Reddit, foram fixados os hiperparâmetros apresentados no artigo os quais alcançaram o atual estado da arte em relação à precisão *F1 Score*. As Tabelas 1 e 2 apresentam os hiperparâmetros definidos.

Tabela 1: Hiperparâmetros do ClusterGCN

	Conjuntos de Dados		
Hiperparâmetro	**Cora**	**Pubmed**	**Reddit**
Partições de treino	[10, 15, 20]	[80, 130, 180]	1500
Partições de validação	2	2	20
Partições de teste	1	1	1
Partições por batch de treinamento	[1, 2, 3]	[1, 2, 3]	20
Num. de camadas intermediárias	[3, 4, 5]	[3, 4, 5]	4
Num. neurônios nas camadas intermediárias	128	128	128
Taxa de aprendizado	[0.001, 0.005, 0.01, 0.05]	[0.001, 0.005, 0.01, 0.05]	0.01

Tabela 2: Hiperparâmetros do AENN

Hiperparâmetro	**Valores**
Taxa de aprendizado	[0.001, 0.005, 0.01, 0.05]
Vértices por batch de treinamento	[32, 64, 128]

Após a definição dos hiperparâmetros, o método *busca em grade* foi aplicado para obtenção das melhores redes. Foram então selecionadas 6 redes, uma para cada par (arquitetura, conjunto de dados). Todos os experimentos foram realizados em um computador do laboratório de pesquisas da EIC/CEFET-RJ. O computador possui sistema operacional Ubuntu 18.04, processador *Intel Xeon CPU E3-1220 v5 3.00GHz* e memória *RAM* de 32 GB.

Para alcançar os objetivos deste trabalho, as métricas de custo computacional e poder preditivo foram aplicadas às 6 melhores redes e definidas da seguinte forma: *i)* para medir o consumo de memória, foi utilizada a biblioteca *memory profiler* [Pedregosa, 2020] que permite monitorar a quantidade de memória alocada ao longo do tempo de execução de um processo; *ii)* para medir o tempo de treinamento e inferência bastou cronometrar a execução do processo por meio do módulo nativo *time* do Python. Tanto para o consumo de memória quanto para o tempo de treinamento e inferência, o monitoramento ocorreu desde o início até a finalização de cada processo e; *iii)* o poder preditivo foi medido por meio da métrica *F1 score* aplicada ao conjunto de teste.

3.5 Conjuntos de Dados

Neste trabalho foram utilizados 3 conjuntos de dados disponíveis para *download* e comumente utilizados na literatura. São eles o *Pubmed* [Sen et al., 2008], o *Cora* [Sen et al., 2008] e

o *Reddit* [Hamilton et al., 2017a] . Os 3 conjuntos de dados selecionados foram utilizados nos experimentos originais do ClusterGCN. As propriedades dos conjuntos de dados estão descritas na Tabela 3 e a distribuição de classes é apresentada nas Figuras 8, 9 e 10.

Tabela 3: Propriedades dos conjuntos de dados

Categoria	Nome	Vértices	Arestas	Características	Classes
Citações	Cora	2708	5429	1433	7
Citações	Pubmed	19717	44338	500	3
Redes Sociais	Reddit	232965	11606919	602	41

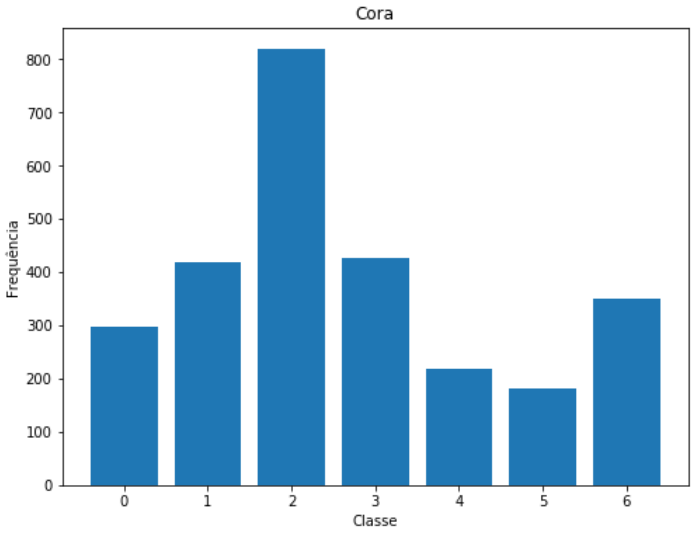

Figura 8: Distribuição de classes do conjunto de dados Cora

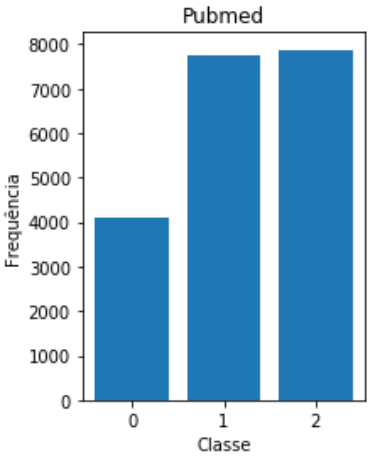

Figura 9: Distribuição de classes do conjunto de dados Pubmed

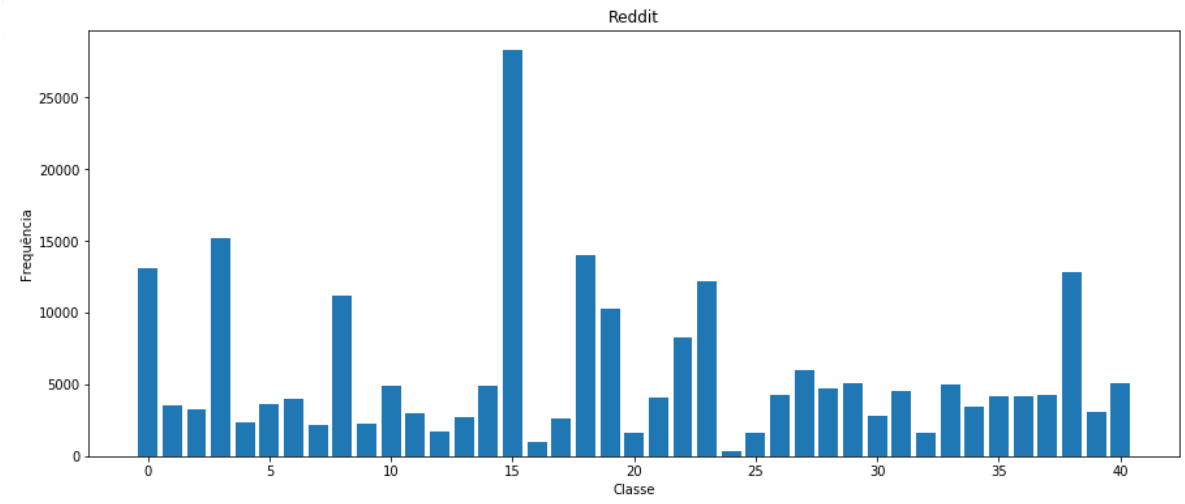

Figura 10: Distribuição de classes do conjunto de dados Reddit

No conjunto de dados *Cora* um vértice representa um artigo científico no campo do apren-

dizado de máquina. Uma aresta representa uma citação entre os artigos. Uma classe representa um subcampo do aprendizado de máquina (*e.g.*, Algoritmos Genéticos, Aprendizado por Reforço, etc). Cada uma das características associadas a um vértice representa a presença ou ausência de uma palavra no artigo, na forma de um *bit*.

No conjunto de dados *Pubmed* um vértice representa um artigo científico no campo da medicina relacionado à diabetes. Uma aresta representa uma citação entre os artigos. Uma classe representa o tipo da diabetes (Pré-diabete, Tipo 1, Tipo 2). Cada uma das características associadas a um vértice representa a frequência de uma palavra no artigo, calculada pelo método estatístico *Term Frequency Inverse Document Frequency* (TFIDF) que obtém a frequência de uma palavra em um documento, inversamente ponderada pela frequência que essa palavra ocorre em documentos distintos.

Reddit é um fórum on-line de discussão onde usuários podem realizar postagens e comentários sobre determinado tópico. O conjunto de dados *Reddit* possui postagens realizadas no mês de Setembro de 2014 amostradas de 50 grandes comunidades. Um vértice representa uma postagem. Uma aresta denota que duas postagens foram comentadas pelo mesmo usuário. Uma classe representa a comunidade à qual a postagem pertence. Um vetor de características associado a um vértice é formado pela concatenação de: *i)* um *GloVe Common Crawl word vector* [Pennington et al., 2014] de 300 posições; *ii)* a representação média do título da postagem; *iii)* a representação média de todos os comentários da postagem; *iv)* a pontuação da postagem e; *v)* o número de comentários realizados na postagem.

Capítulo 4

Avaliação Experimental

As Tabelas 4 e 5 exibem os melhores hiperparâmetros obtidos pela busca em grade para cada arquitetura/conjunto de dados. Cada uma das 6 redes obtidas pela busca em grande foi novamente treinada por 3 vezes, de forma a monitorar o tempo de treinamento e inferência e o consumo de memória, obtendo os valores médios e máximos respectivamente. A acurácia obtida pelos modelos é apresentada na Tabela 6, na qual vale destacar os valores muito próximos mensurados no conjunto de dados Pubmed (87,47% do ClusterGCN contra 85,19% da AENN).

Tabela 4: Melhores hiperparâmetros ClusterGCN

	Conjuntos de Dados		
Hiperparâmetro	**Cora**	**Pubmed**	**Reddit**
Partições de treino	15	80	1500
Partições de validação	2	2	20
Partições de teste	1	1	1
Partições por batch de treinamento	3	3	20
Num. de camadas intermediárias	3	4	4
Num. neurônios nas camadas intermediárias	128	128	128
Taxa de aprendizado	0.001	0.05	0.01

Tabela 5: Melhores hiperparâmetros AENN

	Conjuntos de Dados		
Hiperparâmetro	**Cora**	**Pubmed**	**Reddit**
Taxa de aprendizado	0.01	0.01	0.001
Vértices por batch de treinamento	64	128	128

Tabela 6: Acurácia dos modelos (F1 Score)

	ClusterGCN	**AENN**
Cora	86,90	14,02
Pubmed	**87,47**	**85,19**
Reddit	95,73	61,95

A Figura 11 apresenta o consumo de memória durante todo o tempo de treinamento e inferência das melhores redes obtidas. Cada gráfico apresenta as 3 execuções da mesma rede

para um determinado conjunto de dados. Como é possível observar, as execuções mostram-se consideravelmente regulares em ambas as dimensões.

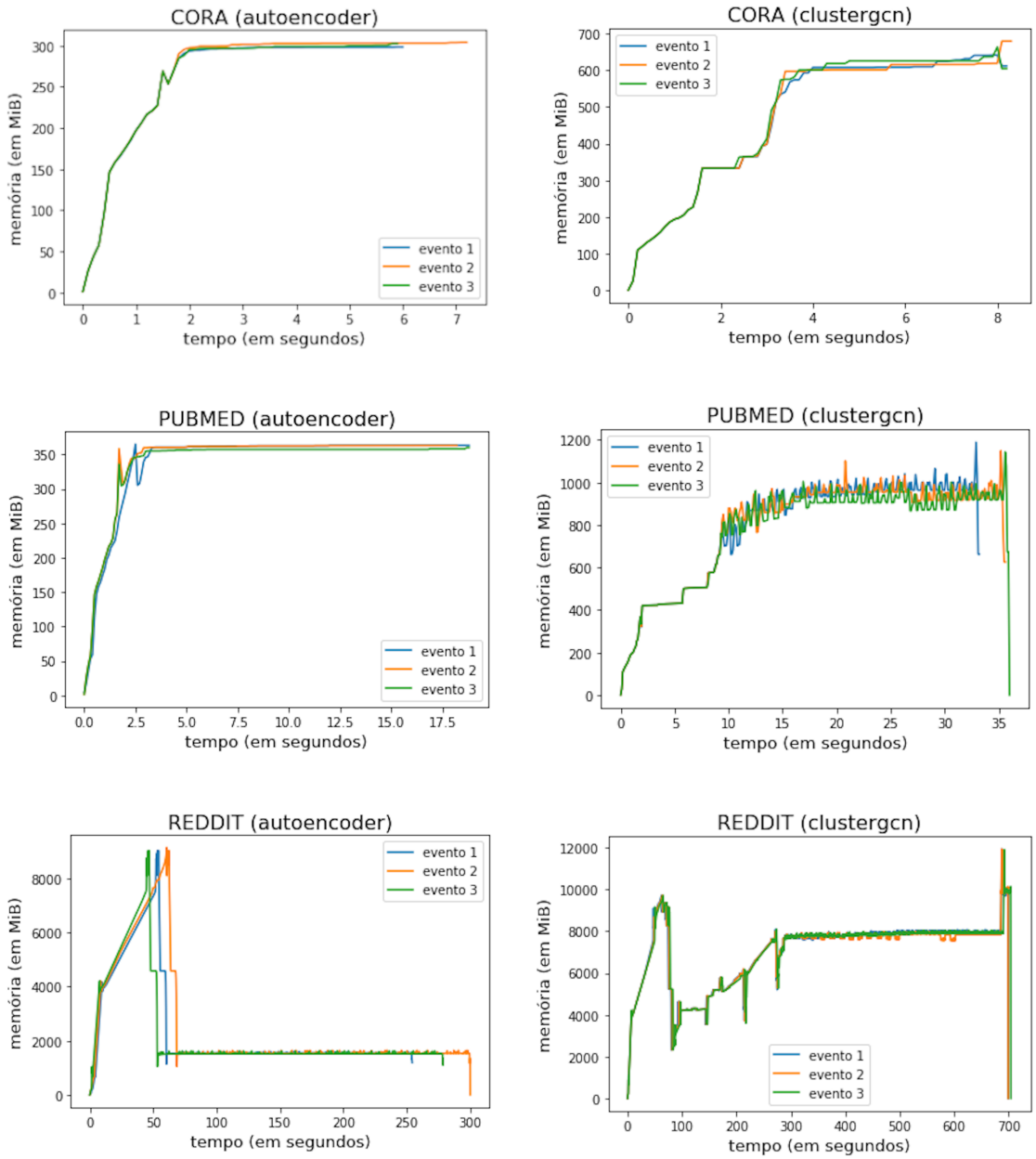

Figura 11: Consumo de memória das melhores redes

As Figuras 12, 13 e 14 apresentam os gráficos comparativos entre as arquiteturas. Em cada figura, o gráfico à esquerda apresenta uma execução de cada arquitetura de forma a exibir um comparativo entre elas. Nos dois gráficos mais à direita são apresentados o tempo médio de treinamento e inferência e o consumo máximo de memória. Vale ressaltar que as redes interrompem o treinamento quando alcançada a condição definida pela regra de *Early Stopping*.

Na Figura 12 são apresentados os valores mensurados para o conjunto de dados Cora.

Observa-se que a arquitetura AENN obteve um tempo médio de treinamento e inferência 22,63% menor em comparação ao ClusterGCN. Já o consumo máximo de memória para a arquitetura ClusterGCN ultrapassa o dobro em comparação à AENN. Uma vez que o conjunto de dados foi pré-processado para servir de entrada para as duas arquiteturas sem sofrer modificações, percebe-se um consumo de memória similar nos primeiros segundos do monitoramento. Após este carregamento inicial, as arquiteturas tendem à um consumo de memória relativamente constante que pode ser interpretado como o transcurso das épocas de treinamento.

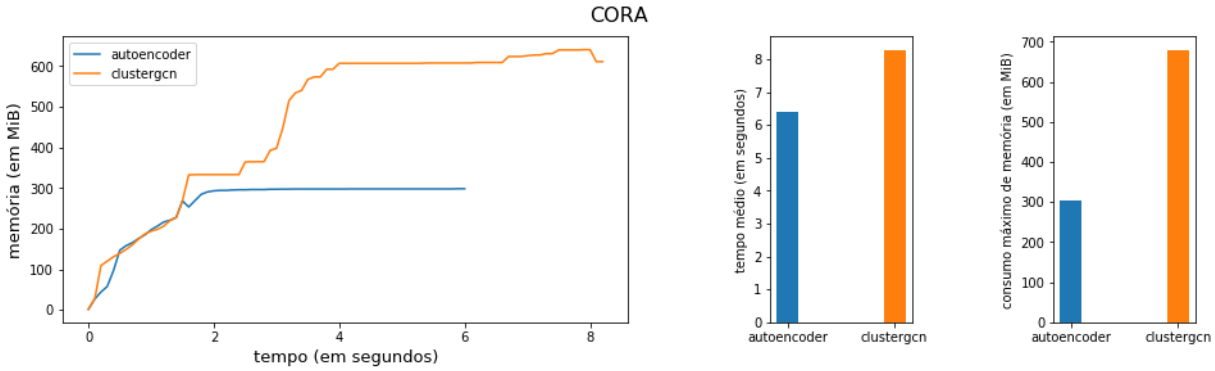

Figura 12: Comparação entre AENN e ClusterGCN utilizando o conjunto de dados Cora

Na Figura 13 são apresentados os valores mensurados para o conjunto de dados Pubmed. A arquitetura AENN obteve um tempo médio de treinamento e inferência 46,81% menor em comparação ao ClusterGCN. Já o consumo máximo de memória para a arquitetura ClusterGCN chega ao triplo em comparação à AENN. Mais uma vez o consumo de memória é similar nos primeiros segundos e tende a permanecer constante durante o transcurso das épocas de treinamento.

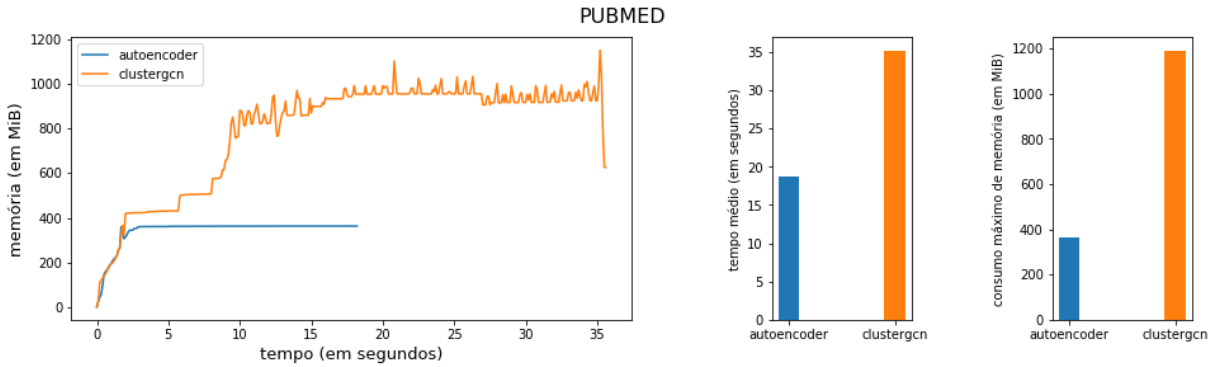

Figura 13: Comparação entre AENN e ClusterGCN utilizando o conjunto de dados Pubmed

Na Figura 14 são apresentados os valores mensurados para o conjunto de dados Reddit. A arquitetura AENN obteve um tempo médio de treinamento e inferência 60,35% menor em

comparação ao ClusterGCN. O consumo máximo de memória foi apenas 23,36% menor para a AENN. Apesar desta diferença reduzida do consumo máximo de memória em relação aos conjuntos de dados anteriores, observa-se que durante as épocas de treinamento a diferença chega a ser 4 vezes maior para o ClusterGCN, o que demonstra que o gargalo nesse caso está relacionado ao carregamento inicial dos dados.

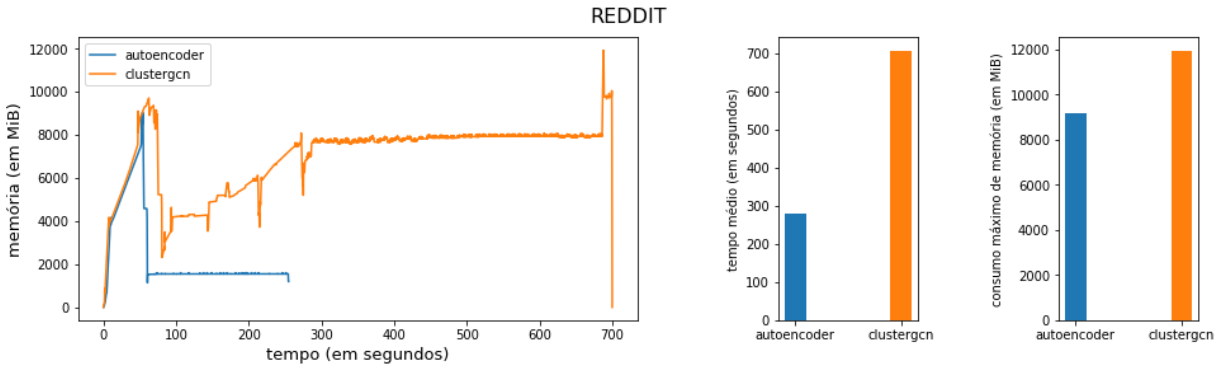

Figura 14: Comparação entre AENN e ClusterGCN utilizando o conjunto de dados Reddit

Capítulo 5

Conclusão

5.1 Análise Retrospectiva

Este trabalho consistiu em realizar uma análise experimental comparativa utilizando as arquiteturas GNN e AENN no aprendizado de representações, dentro de um mesmo contexto de aprendizado em tarefas de classificação. Empregamos o ClusterGCN como arquitetura GNN e uma implementação própria de AENN. Os melhores hiperparâmetros para cada rede foram obtidos por meio de uma busca em grade. As melhores redes foram então avaliadas em termos de custo computacional e poder preditivo sobre os conjuntos de dados Cora, Pubmed e Reddit.

A arquitetura proposta pelo ClusterGCN foi capaz de gerar modelos satisfatórios, em termos de poder preditivo, para os três conjuntos de dados utilizados neste experimento. Sua técnica em utilizar partições do grafo como amostras no processo de aprendizado reduziu o uso dos recursos computacionais, o que para os padrões de hardware atuais não justifica, em um primeiro momento, o uso das AENNs. Por outro lado, a tendência natural é a de que conjuntos de dados cada vez maiores necessitem ser manipulados, o que eventualmente poderá tornar-se novamente um gargalo até mesmo para o ClusterGCN.

O resultado alcançado pela AENN para o conjunto de dados Pubmed, em termos de poder preditivo, foi muito próximo do resultado alcançado pelo ClusterGCN. Um indício para justificar esse resultado pode talvez residir nas características do conjunto de dados. O Pubmed, dentre os três conjuntos de dados empregados, é o que apresenta o menor número de classes (apenas três). Além disso, dentre as três classes possíveis, duas são balanceadas. Já nos conjuntos de dados Cora e Reddit, além da quantidade maior de classes, observa-se também um nível maior de desbalanceamento entre elas.

Outra hipótese levantada está relacionada com a quantidade de exemplos de cada conjunto de dados. A quantidade de exemplos no conjunto de dados Pubmed é cerca de 7,3 vezes maior do que no Cora. Uma hipótese possível seria a de que, para a arquitetura AENN, o Cora possui um número insuficiente de exemplos para o aprendizado de representações.

A hipótese anterior conduz ao seguinte questionamento: se o elemento limitador da AENN

no conjunto de dados Cora foi a quantidade insuficiente de exemplos, por que motivo a acurácia no conjunto de dados Reddit foi consideravelmente menor do que a obtida pelo ClusterGCN? O conjunto de dados Reddit é cerca de 12 vezes maior do que o Pubmed. Uma hipótese possível seria a de que, para um conjunto de dados com essas proporções, seria necessário uma rede AENN com maior número de camadas para o devido aprendizado de representações.

5.2 Trabalhos Futuros

Diante dos resultados e análise realizada, a ideia de que AENNs possam ser satisfatoriamente empregadas no aprendizado de representações não fica descartada. O reduzido uso de tempo e recursos computacionais pela AENN pode, em determinados casos, torná-la mais adequada ou simplesmente suficiente para o problema que se deseja resolver. Apesar disso, obter os elementos indicativos para apoiar essa decisão requerem o estudo das hipóteses levantadas neste trabalho.

Como trabalho futuro, outros conjuntos de dados similares ao Pubmed podem ser experimentados. O balanceamento e o reduzido número de classes podem ser um indício de que o emprego da AENN é adequado. Técnicas de balanceamento de dados também podem ser investigadas.

Uma AENN com maior número de camadas também pode ser testada. No aprendizado de representações, cada camada de uma rede pode ser considerada como um nível maior de abstração do dado de entrada. Além disso, AENNs geram representações de baixa dimensionalidade com base na correlação entre os dados de entrada. Uma configuração com mais camadas pode levar à uma maior capacidade de aprendizado, principalmente para conjuntos de dados com maior número de características associadas aos vértices.

Tanto o Cora quanto o Pubmed são oriundos de artigos (representados pelos vértices) e citações entre estes (representados pelas arestas). Além das diferenças já citadas entre esses dois conjuntos de dados (quantidade de exemplos e classes), outra que pode ter influenciado os resultados obtidos pela AENN é a técnica de representação aplicada às características associadas aos vértices. No Cora, cada característica representa a ausência ou presença de uma palavra na forma de um bit. Já no Pubmed, cada característica representa a frequência de uma palavra calculada pelo método estatístico TFIDF. Como oportunidade de trabalho futuro, conjuntos de dados pré-processados com o método TFIDF podem ser avaliados.

Referências Bibliográficas

Atwood, J. and Towsley, D. (2016). Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001. 15

Bezerra, E. (2016). *Introdução à Aprendizagem Profunda*, pages 57–86. 12

Chen, J., Ma, T., and Xiao, C. (2018a). Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*. 15

Chen, J., Zhu, J., and Song, L. (2018b). Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pages 941–949. 15

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. (2019). Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. *arXiv preprint arXiv:1905.07953*. 2, 3, 15, 16

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232. 2

Feinberg, E. N., Sur, D., Husic, B. E., Mai, D., Li, Y., Yang, J., Ramsundar, B., and Pande, V. S. (2018). Spatial graph convolutions for drug discovery. *arXiv preprint arXiv:1803.04465*. 2

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1263–1272. JMLR.org. 2, 15

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`. 1

Google (2020). google-research/google-research. `https://github.com/google-research/google-research`. Library Catalog: github.com. 17

Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE. 14

Hamilton, W., Ying, Z., and Leskovec, J. (2017a). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034. 15, 21

Hamilton, W. L., Ying, R., and Leskovec, J. (2017b). Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*. 13

Herman, I., Melançon, G., and Marshall, M. S. (2000). Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on visualization and computer graphics*, 6(1):24–43. 1

Jain, A., Zamir, A. R., Savarese, S., and Saxena, A. (2016). Structural-rnn: Deep learning on spatio-temporal graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907. 1, 2

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc. 1

LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995. 13

Lehman, E. and Leighton, T. (2004). Mathematics for computer science. `https://www.cs.princeton.edu/courses/archive/spring10/cos433/mathcs.pdf`. 1, 5, 6, 7

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2015). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*. 2

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133. 7

McGregor, A. (2014). Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20. 1

Micheli, A. (2009). Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511. 14

Pedregosa, F. (2020). pythonprofilers/memory-profiler. `https://github.com/pythonprofilers/memory_profiler`. 20

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543. 22

Prechelt, L. (1998). Early Stopping - But When? In Orr, G. B. and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade*, pages 55–69. Springer Berlin Heidelberg, Berlin, Heidelberg. 19

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386. 7

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80. 14

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. (2008). Collective classification in network data. *AI magazine*, 29(3):93–93. 20

Sperduti, A. and Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735. 14

Stanford (2019). Unsupervised Feature Learning and Deep Learning. `http://ufldl.stanford.edu/tutorial/`. 7, 8, 9, 12

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*. 14

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 974–983, New York, NY, USA. ACM. 1, 2

Zhuang, C. and Ma, Q. (2018). Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 499–508, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee. 1