Redes Neurais de Grafos Cooperativos

Marcéli Melchiors *
Escola de Matemática Aplicada
Fundação Getúlio Vargas
Rio de Janeiro, RJ

Nicole dos Santos de Souza†
Escola de Matemática Aplicada
Fundação Getúlio Vargas
Rio de Janeiro, RJ

Abstract

Este trabalho é baseado no artigo Cooperative Graph Neural Networks [2], de Ben Finkelshtein, Xingyue Huang, Michael Bronstein e Ismail Ilkan Ceylan. Nele, propõe-se uma nova arquitetura para GNNs, denominada CO-GNNs, em que cada nó pode escolher entre 'ouvir', 'transmitir', 'ouvir e transmitir' ou 'isolar'. Essa nova abordagem é mais flexível e dinâmica, permitindo que cada nó escolha sua estratégia com base em seu estado, explorando a topologia do grafo enquanto aprende. Como as redes neurais de grafos (GNNs) são baseadas em "passagem de mensagens", ou seja, em cada camada, o estado de cada nó é atualizado com base em mensagens de seus vizinhos, essa nova arquitetura possibilita essa passagem de maneiras mais complexas que as proporcionadas pelos GNNs tradicionais. Apresentamos uma análise teórica e prática.

1 Motivação

Os dados em muitos domínios do mundo real são naturalmente representados como grafos. Redes Neurais de Grafos (GNNs) são redes neurais projetadas para capturar a estrutura do grafo e a dependência entre os nós, permitindo que o modelo aprenda representações poderosas dessas entidades e suas relações. A maioria das GNNs adotam um método denominado "passagem de mensagens", onde cada nó recebe mensagens (informações) de seus nós vizinhos e, em seguida, atualiza sua própria representação com base nessas mensagens e em sua própria representação atual. Esse processo é iterativo e pode ocorrer em várias camadas da GNN. Embora eficaz, esse método enfrenta limitações como a dificuldade em lidar com dependências de longo alcance, exigindo mais camadas para receber informações de vizinhos distantes, o que pode levar à perda de informações devido ao over-squashing [1].

Considere como exemplo o grafo da Figura 1. Suponha que o nó preto (w) precisa de informações apenas dos vizinhos que têm um vizinho amarelo (u), focando assim em um subgrafo específico (em cinza). Os GNNs tradicionais não são capazes de fazer isso, pois não podem condicionar a passagem de mensagens com base em dois saltos de distância.

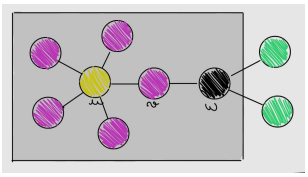

Figure 1: O nó w escuta v e iterativamente recebe informações do subgrafo em cinza.

*marcelimelchiors12@gmail.com
†nicoledossantosouza@gmail.com

Preprint. Under review.

Há, ainda, a possibilidade de os nós escolherem diferentes estratégias em cada camada, tornando o processo ainda mais complexo. Diante disso, torna-se relevante a construção de um método em que a "passagem de mensagens" seja mais flexível, permitindo que cada nó decida como enviar e receber informações de seus vizinhos.

2 Proposta

O intuito do artigo estudado é propor uma nova classe de arquiteturas, denominadas redes neurais de grafos cooperativos (CO-GNNs), que podem lidar com aspectos não capturados pelas GNNs tradicionais. Nas CO-GNNs, cada nó do grafo é visto como um jogador que pode realizar uma das seguintes ações em cada camada: Padrão (STANDARD), Ouvir (LISTEN), Transmitir (BROADCAST) e Isolar (ISOLATE). Com a ação "Padrão" o nó transmite para vizinhos que ouvem e ouve vizinhos que transmitem. A ação "Ouvir" apenas ouve os vizinhos que transmitem, enquanto "Transmitir" faz o contrário. Já a ação "Isolar", nem ouve nem transmite. As CO-GNNs compreendem duas redes neurais de passagem de mensagens cooperativas e treinadas em conjunto: uma rede de ambiente η (para resolver a tarefa dada) e uma rede de ação π (para escolher as melhores ações).

3 Contexto

3.1 Redes Neurais de Grafos

Gilmer et al. [3] mostraram que a grande maioria das GNNs pode ser implementada através de message-passing, onde a ideia fundamental é atualizar a representação de cada nó com base em um um processo de agregação de informações que fluem dos vizinhos desse nó.

Seja $X \in \mathbb{R}^{|V| \times d}$ a matriz de características dos nós, onde $x_v \in \mathbb{R}^d$ representa a característica de um nó $v \in V$. Um grafo simples, não direcionado e com atributos é representados por $G = (V, E, X)$. A representação inicial de cada nó v é dada por $h_v^{(0)} = x_v$. As redes neurais de passagem de mensagens (MPNNs) atualizam essa representação por meio de $0 \leq \ell \leq L-1$ iterações, com base em seu próprio estado e nos estados dos seus vizinhos N_v:

$$h_v^{(\ell+1)} = \phi^{(\ell)} \left(h_v^{(\ell)}, \psi^{(\ell)} \left(h_v^{(\ell)}, \{\{h_u^{(\ell)} | u \in N_v\}\} \right) \right)$$

com $\phi^{(\ell)}$ e $\psi^{(\ell)}$ funções diferenciáveis de atualização e agregação, respectivamente. A dimensão dos embeddings dos nós na iteração ℓ é $d^{(\ell)}$. As representações finais $h_v^{(L)}$ de cada nó podem ser usadas para prever propriedades do nó ou combinadas para formar um vetor de embeddings do grafo $z_G^{(L)}$ para prever propriedades dele. O vetor de embeddings pode ser calculado através da média, soma ou máximo dos elementos.

As MPNNs ditas básicas são da forma:

$$h_v^{(\ell+1)} = \sigma \left(W_s^{(\ell)} h_v^{(\ell)} + W_n^{(\ell)} \psi \left(\{\{h_u^{(\ell)} | u \in N_v\}\} \right) \right)$$

em que $W_s^{(\ell)}$ atua na auto-representação do nó e $W_n^{(\ell)}$ na representação agregada de seus vizinhos. São ambas matrizes de parâmetros ajustáveis de dimensão $d^{(\ell)} \times d^{(\ell+1)}$. Já σ é uma função de ativação não-linear, e ψ é uma função de agregação.

3.2 Estimador Straight-through Gumbel-softmax

Na abordagem proposta, depende-se de uma rede de ação para prever ações categóricas para os nós no grafo, o que não é diferenciável e representa um desafio para a otimização baseada em gradiente. Uma solução para isso é o Gumbel-softmax [4] [5], que resulta em uma aproximação contínua e diferenciável.

Seja um conjunto finito Ω de ações e seja $a \in \Omega$, uma ação qualquer. O Gumbel-softmax é um truque de reparametrização para estimar $p \in \mathbb{R}^{|\Omega|}$, um vetor de probabilidade onde cada elemento

armazena a probabilidade de uma ação específica, usando um vetor Gumbel $g \in \mathbb{R}^{|\Omega|}$, que contém uma amostra independente e identicamente distribuída $g(a) \sim \text{GUMBEL}(0, 1)$ para cada a. Sendo τ um parâmetro de temperatura e p uma distribuição categórica, as pontuações Gumbel-softmax são:

$$\text{Gumbel-softmax}(p; \tau) = \frac{\exp((\log(p) + g)/\tau)}{\sum_{a \in \Omega} \exp((\log(p(a)) + g(a))/\tau)}$$

Quando τ diminui, o resultado se aproxima de um vetor one-hot, um vetor que possui um elemento com o valor 1 e todos os outros elementos com o valor 0. O estimador Straight-through Gumbel-softmax utiliza o Gumbel-softmax na passagem para trás para permitir atualizações diferenciáveis, enquanto na passagem para frente usa amostragem comum.

4 Redes Neurais de Grafos Cooperativos

Em CO-GNNs, os nós dos grafos colaboram entre si para alcançar um objetivo comum. Essas redes são capazes de aprender representações de dados que levam em conta não apenas as características individuais de cada nó, mas também como esses nós interagem e se influenciam mutuamente.

O estado de cada nó é representado pela condição atual dele. Para a atualização do estado de cada nó, primeiramente, cada nó, considerando tanto seu estado atual quanto o estado dos nós ao seu redor, seleciona uma ação a partir de um conjunto predefinido de ações, que conforme citado, são: Padrão (STANDARD), Ouvir (LISTEN), Transmitir (BROADCAST) e Isolar (ISOLATE).

Se todos os nós escolhem a ação "Isolar", todas as conexões do grafo são removidas, resultando em previsões isoladas por nó. Quando todos os nós escolhem a ação "Padrão", temos MPNNs, seguindo o esquema tradicional. A flexibilidade de mudar essas ações dinamicamente torna o modelo mais robusto, permitindo a separação entre o grafo de entrada e o grafo computacional, e adiciona direcionalidade à passagem de mensagens: um nó só ouve os vizinhos que estão transmitindo e vice-versa.

4.1 Exemplo ilustrativo

Na figura abaixo, vemos dois exemplos: o nó u escuta todos os vizinhos na primeira camada, apenas v na segunda, e os nós s e r na terceira camada. Por outro lado, o nó v escuta o nó w nas duas primeiras camadas e o nó u na última camada. Isso mostra um fluxo de informações dinâmico e não restrito a um subgrafo fixo.

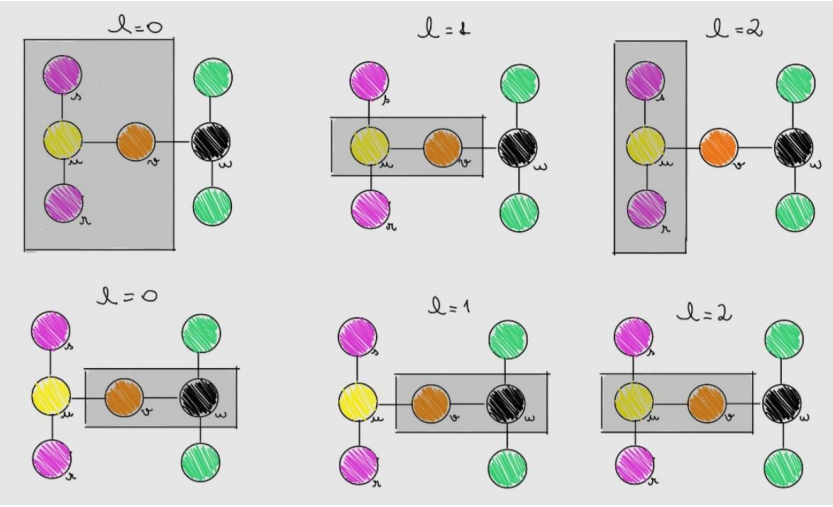

Figure 2: Exemplo de fluxo de informações para os nós u, v.

Essa situação não poderia ser solucionada com os MPNNs. Os CO-GNNs, por sua vez, podem resolver o problema do seguinte modo: Pode-se fazer com que o nó u escolha as ações (L, L, S), os nós v e w escolham (S, S, L), e os nós s e r escolham (S, I, S).

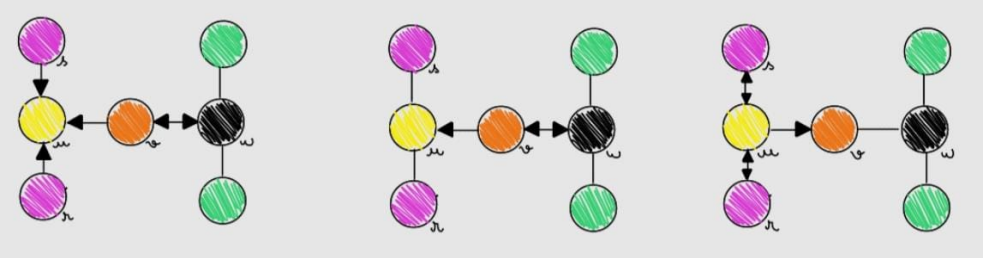

Figure 3: Exemplo de fluxo de informações para os nós u, v.

4.2 Detalhamento matemático

O nó ajusta seu estado com base no estado atual e nos estados de alguns de seus vizinhos, de acordo com o estado definido na etapa anterior. Assim, cada nó pode decidir como transmitir e receber informações dos nós vizinhos. Uma CO-GNN (π, η) consiste em duas GNNs que cooperam: (i) uma rede de ação π, que decide as melhores ações a serem tomadas, e (ii) uma rede de ambiente η, que atualiza as representações de cada nó.

Para atualizar as representações $h_v^{(\ell)}$ de um nó v, a rede CO-GNN realiza: (i) Para cada nó v, a rede de ação π prediz uma distribuição de probabilidade $p_v^{(\ell)} \in \mathbb{R}^4$ para as ações possíveis $\{P, O, T, I\}$, baseando-se no estado do nó e no estado dos seus vizinhos N_v:

$$p_v^{(\ell)} = \pi\left(h_v^{(\ell)}, \{\{h_u^{(\ell)} | u \in N_v\}\}\right).$$

Com o estimador Straight-through Gumbel-softmax, para cada nó v uma ação $a_v^{(\ell)}$ é selecionada a partir da distribuição $p_v^{(\ell)}$. (ii) Em seguida, a rede de ambiente η , com base nas ações amostradas, ajusta o estado de cada nó. Então, temos:

$$h_v^{(\ell+1)} = \begin{cases} \eta^{(\ell)}\left(h_v^{(\ell)}, \{\}\right), & a_v^{(\ell)} = I \text{ ou } T \\ \eta^{(\ell)}\left(h_v^{(\ell)}, \{h_u^{(\ell)} | u \in N_v, a_u^{(\ell)} = P \text{ ou } T\}\right), & a_v^{(\ell)} = O \text{ ou } P. \end{cases}$$

Tomando como exemplo a equação acima, se um vértice v opta por Isolar ou Transmitir, então a atualização é feita baseada apenas no seu estado anterior. Se um vértice v escolhe Ouvir ou Padrão, sua atualização ocorre levando em conta tanto seu estado anterior quanto o dos vizinhos que escolhem Transmitir ou Padrão.

Com essa operação, é possível obter todas as representações h_v^L para cada nó v, sendo L o número de camadas. De forma geral, CO-GNN (π, η) pode usar qualquer GNN como a rede de ação π e a rede de ambiente η.

5 Propriedades de CO-GNNs

CO-GNNs chegam em grafos mais adequados para a tarefa alvo, pois se a tarefa necessita informações apenas de vizinhos que possuem um determinado grau, então a rede de ação pode aprender a se concentrar somente nesses nós.

Com os nós escolhendo suas ações, é como se o grafo estivesse reconfigurando seu direcionamento. Se um nó v Transmite e seu vizinho w Ouve, então há uma aresta direcionado de v para w. A atualização do conjunto de arestas direcionadas $E^{(\ell)}$ do grafo $G^{(\ell)} = (V, E^{(\ell)})$ é dado por:

$$h_v^{(\ell+1)} = \eta^{(\ell)} \left(h_v^{(\ell)}, \{\{h_u^{(\ell)} | (u,v) \in E^{(\ell)}\}\} \right).$$

Desse modo, CO-GNNs podem ser implementadas apenas com a matriz de adjacência dos grafos induzidos em cada camada.

Outras propriedades conceituais interessantes são: (i) Dinâmico, dado que CO-GNNs operam em um grafo que é dinâmico entre as camadas devido ao aprendizado que eles obtêm. (ii) Assíncrono, permitindo maior flexibilidade e eficiência na propagação de mensagens e atualizações de estados dos nós.

5.1 Isomorfismo

Proposição 5.1. Seja $G_1 = (V_1, E_1, X_1)$ e $G_2 = (V_2, E_2, X_2)$ dois grafos não isomorfos. Então, para qualquer limiar $0 < \delta < 1$, existe uma parametrização de uma arquitetura CO-GNN usando um número suficiente de camadas L, satisfazendo $\mathbb{P}(z_{G_1}^{(L)} \neq z_{G_2}^{(L)}) \geq 1 - \delta$.

As CO-GNNs aprendem distribuições de probabilidade para ações em cada nó v, que são iguais para nós isomorfos. No entanto, a aleatoriedade na seleção de ações pode diferir, tornando o modelo invariante em média e a variância ajuda a diferenciar nós isomorfos que são indistinguíveis por 1-WL. Desse modo, conseguimos distinguir grafos com uma função de agrupamento injetiva.

A capacidade de distinguir grafos isomorfos é relevante para a generalização e robustez do modelo em aplicações reais. Em muitos cenários práticos, é importante que o modelo seja capaz de reconhecer e tratar grafos que são estruturalmente diferentes de maneiras distintas. Por exemplo, em química computacional, onde a estrutura de uma molécula (representada como um grafo) tem isomorfismos que não devem ser tratados como equivalentes devido a diferentes propriedades químicas ou físicas.

5.2 Tarefas de Longo Alcance

Tarefas de longo alcance em redes exigem a propagação eficiente de informações entre nós distantes. Um paradigma dinâmico de passagem de mensagens se mostra eficaz ao focar apenas nas informações relevantes para a tarefa específica. Esse método filtra dados irrelevantes, utilizando o caminho mais curto entre os nós de origem e destino, maximizando o fluxo de informações. Essa abordagem pode ser generalizada para múltiplos nós distantes.

O teorema abaixo mostra que se uma propriedade de um nó v é uma função de k nós distantes, as CO-GNNs podem aproximar essa função. Isso é baseado nas características de k nós que podem ser transmitidas ao nó de destino sem perda de informação em CO-GNNs; e na camada final de uma CO-GNN, como uma MLP, que pode aproximar qualquer função diferenciável dessas características.

Teorema 5.2. Seja $G = (V, E, X)$ um grafo conectado com características de nós. Para algum $k > 0$, para qualquer nó de destino $v \in V$, para quaisquer k nós de origem $u_1, ..., u_k \in V$, e para qualquer função compacta e diferenciável $f : \mathbb{R}^{d(0)} \times ... \times \mathbb{R}^{d(0)} \to \mathbb{R}^d$, existe uma CO-GNN de L camadas computando representações finais de nó de forma que, para qualquer $\epsilon, \delta > 0$, vale que $P(|h_v^{(L)} - f(x_{u1}, ..., x_{uk})| < \epsilon) \geq 1 - \delta$

6 Experimentos

6.1 Implementação

Consideremos para essa implementação o dataset Roman Empire, que é baseado no extenso artigo da Wikipédia sobre o Império Romano. Cada nó no grafo representa uma palavra do texto, e duas palavras são conectadas se são sequenciais no texto ou conectadas na árvore de dependência da sentença. O grafo resultante é um grafo em cadeia com arestas adicionais para dependências sintáticas. Os nós são classificados por seus papéis sintáticos, com 17 classes principais e uma 18ª para os papéis menos frequentes, e possuem embeddings de palavras como características. Analisar esse dataset como um grafo é interessante para categorizar palavras conforme seus papéis sintáticos, aproveitando a estrutura sintática e semântica do texto.

Construímos, então, uma implementação para a CO-GNN para classificação dos nós desse dataset. A parte principal do código está disposta abaixo e o restante pode ser encontrado em <https://github.com/nicolessouza/CO-GNNs>.

```
class ActionNetwork(MessagePassing):
    def __init__(self, in_channels, hidden_dim, num_hidden_layers, out_channels=4):
        """
        Inicializa a rede de ação para CO-GNN.
        """
        super(ActionNetwork, self).__init__(aggr='mean')  # Usando agregação média por padrão.

        self.layers = ModuleList()

        # Primeira camada oculta que transforma a entrada para a dimensão oculta.
        self.layers.append(Linear(in_channels, hidden_dim))
        self.layers.append(ReLU())

        # Camadas ocultas intermediárias.
        for _ in range(num_hidden_layers - 1):
            self.layers.append(Linear(hidden_dim, hidden_dim))
            self.layers.append(ReLU())

        # Camada de saída que mapeia a última camada oculta para as ações.
        self.layers.append(Linear(hidden_dim, out_channels))
        self.softmax = Softmax(dim=1)

    def forward(self, x, edge_index):
        """
        Propagação para frente que processa os estados dos nós e retorna
        as probabilidades das ações.
        """
        return self.propagate(edge_index, size=(x.size(0), x.size(0)), x=x)

    def message(self, x_j):
        """
        Cria mensagens para serem enviadas aos nós centrais na agregação.
        """
        for layer in self.layers:
            x_j = layer(x_j) if not isinstance(layer, ReLU) else layer(x_j)
        return x_j

    def update(self, aggr_out):
        """
        Atualiza os estados dos nós depois da agregação.
        """
        return self.softmax(aggr_out)


class EnvironmentNetwork(torch.nn.Module):
    def __init__(self, in_channels, hidden_dim, out_channels, num_layers):
        """
        Inicializa a rede de ambiente para a CO-GNN.
        A rede usa camadas GCN seguidas de funções de ativação GELU.
        """
        super(EnvironmentNetwork, self).__init__()
        self.layers = ModuleList()

        self.layers.append(GCNConv(in_channels, hidden_dim))

        # Camadas GCN ocultas intermediárias
```

```
        for _ in range(num_layers - 2):
            self.layers.append(GCNConv(hidden_dim, hidden_dim))

        # Camada GCN final para transformar para a dimensão de saída
        if num_layers > 1:
            self.layers.append(GCNConv(hidden_dim, out_channels))
        else:
            # Se houver apenas uma camada, mapeie diretamente da entrada para a saída
            self.layers.append(GCNConv(in_channels, out_channels))

        # Função de ativação GELU após cada camada GCN, exceto a última
        self.activation = GELU()

    def forward(self, x, edge_index):
        """
        Propagação para frente da rede de ambiente.

        """
        for layer in self.layers[:-1]:
            x = self.activation(layer(x, edge_index))

        x = self.layers[-1](x, edge_index)
        return x


def gumbel_softmax(logits, tau=1.0, hard=False, dim=-1):
    """
    Amostragem Gumbel-Softmax para ações discretas.
    """
    gumbels = -torch.empty_like(logits).exponential_().log()  # Amostras Gumbel
    gumbels = (logits + gumbels) / tau  # Aplica a temperatura tau
    y_soft = gumbels.softmax(dim)

    if hard:
        # Straight-through Gumbel-Softmax
        index = y_soft.max(dim, keepdim=True)[1]
        y_hard = torch.zeros_like(logits).scatter_(dim, index, 1.0)
        ret = y_hard - y_soft.detach() + y_soft
    else:
        ret = y_soft
    return ret

class CO_GNN(torch.nn.Module):
    def __init__(self, action_network, environment_network, tau=1.0):
        super(CO_GNN, self).__init__()
        self.action_network = action_network
        self.environment_network = environment_network
        self.tau = tau  # Temperatura Gumbel-Softmax

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        action_logits = self.action_network(x, edge_index)
        actions = gumbel_softmax(action_logits, tau=self.tau, hard=True)

        # Filtrar arestas com base nas ações (Padrão ou Transmissão)
        actions_argmax = actions.argmax(dim=1)
        mask = (actions_argmax[edge_index[0]] == 0) | (actions_argmax[edge_index[0]] == 2)
        new_edge_index = edge_index[:, mask]

        updated_x = self.environment_network(x, new_edge_index)
```

```
        return updated_x
```

O resultado obtido foi uma acurácia no teste de 46,06%, de validação de 46,64% e de treinamento de 50,05% após 5000 épocas. Não é um resultado bom, mas considerando a complexidade do problema, é um começo promissor. Pode-se ter essa noção visualizando o gráfico de *perdas vs. épocas* disposto abaixo.

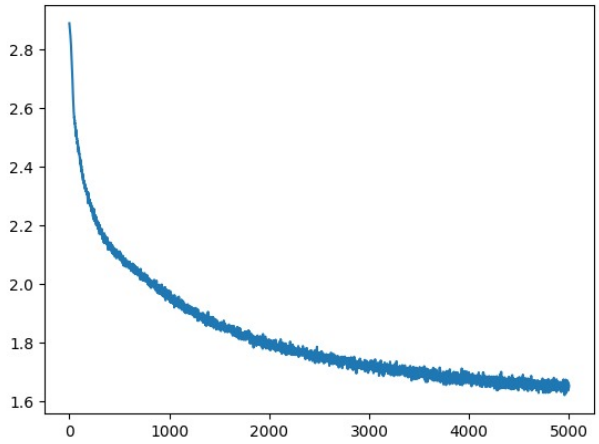

Figure 4: losses vs. epochs

É importante destacar que a implementação desenvolvida é uma versão simplificada e pode ser melhorada com ajustes adicionais, como aprimoramento da arquitetura da rede, ajuste de hiperparâmetros, treinamento por mais épocas e uso de técnicas de regularização.

Por isso, para comparação, treinamos o modelo implementado no artigo com esse mesmo conjunto de dados.A implementação dos autores é encontrada em <https://github.com/benfinkelshtein/CoGNN>.

Dessa vez, obtivemos uma acurácia de 80% no conjunto de teste e no de validação. Este resultado demonstra uma melhora significativa em relação à implementação desenvolvida, por conta dos aprimoramentos feitos no código e por conta dos ajustes de hiperparâmetros.

Por outro lado, uma acurácia de 99% no conjunto de treino para o CO-GNN, comparada com 80% nos conjuntos de validação e teste, indica um possível overfitting. O modelo está se especializando nos dados de treino. Porém, observando o gráfico abaixo, notamos que há uma ausência de queda significativa na acurácia do conjunto de validação/teste, o que é um indicativo de uma característica interessante do modelo: ele é robusto a overfitting. Ou seja, embora tenha se especializado nos dados de treino, isso não atrapalhou consideravelmente o desempenho nos outros conjuntos.

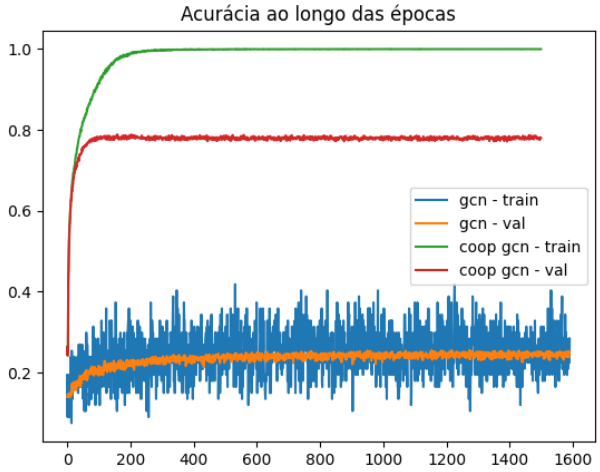

Figure 5: Acurácia versus épocas.

Os resultados indicam que, uma vez aprimorada e ajustada, a CO-GNN é uma solução interessante para o problema proposto, performando bem e capturando satisfatoriamente a complexidade da rede.

7 Conclusão

Esse trabalho explorou a proposta original das CO-GNNs, destacando como essa abordagem supera limitações das GNNs tradicionais, especialmente em relação à passagem de mensagens e à captura de dependências de longo alcance.

As CO-GNNs podem alcançar desempenhos satisfatórios em problemas complexos de classificação de nós, conforme visto experimentalmente.

Cabe acrescentar que, apesar dos benefícios, há desafios a serem enfrentados. Ao dependerem do estimador Straight-through Gumbel-softmax para a seleção de ações, as CO-GNNs introduzem um nível de estocasticidade que pode afetar a estabilidade do treinamento. Embora essa abordagem permita a diferenciação e otimização baseada em gradientes, a natureza não determinística das escolhas de ações pode levar a variações nos resultados finais, exigindo múltiplas execuções para garantir a consistência.

No geral, as CO-GNNs demonstram um potencial significativo para revolucionar a maneira como lidamos com dados estruturados em grafos, oferecendo um framework poderoso e flexível para diversas aplicações no mundo real.

References

[1] Uri Alon and Eran Yahav. “On the Bottleneck of Graph Neural Networks and its Practical Implications”. In: *arXiv preprint arXiv:2006.05205* (2021).

[2] Ben Finkelshtein et al. “Cooperative Graph Neural Networks”. In: *arXiv preprint arXiv:2301.12345* (2023).

[3] Justin Gilmer et al. “Neural message passing for quantum chemistry”. In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[4] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical Reparameterization with Gumbel-Softmax”. In: *arXiv preprint arXiv:1611.01144* (2017).

[5] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. “The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables”. In: *arXiv preprint arXiv:1611.00712* (2017).