GNN Shortest Path Solver: Mathematical Documentation

Anas AIT ALI

2025

1 Problem Formulation

1.1 Shortest Path Problem

Soit un graphe orienté $G = (V, E)$ avec :

- V : ensemble des sommets (nœuds)
- E : ensemble des arêtes
- $w : E \to \mathbb{R}^+$: fonction de poids associant un poids non négatif à chaque arête

Le problème du plus court chemin consiste à trouver un chemin $P = (v_1, v_2, \ldots, v_k)$ entre un nœud source s et un nœud cible t qui minimise :

$$\min \sum_{e \in P} w(e)$$

1.2 Approche par réseau de neurones graphes (GNN)

Notre modèle GNN apprend à approximer la fonction du plus court chemin :

$$f : (G, s, t) \to (P, d)$$

où :

- P : chemin prédit
- d : distance prédite

2 Architecture du modèle

2.1 Représentation du graphe

Pour un graphe G avec n nœuds, on représente :

- Les caractéristiques des nœuds : $X \in \mathbb{R}^{n \times d_{in}}$
- Les caractéristiques des arêtes : $E \in \mathbb{R}^{m \times d_{edge}}$
- La matrice d'adjacence : $A \in \{0, 1\}^{n \times n}$

2.2 Couches GNN

Le modèle utilise plusieurs couches de convolution graphes (GCN) avec la règle de propagation suivante :

Pour la couche l :

$$H^{(l+1)} = \sigma\left(D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

avec :

- $\tilde{A} = A + I$ (matrice d'adjacence avec boucles sur les nœuds)
- D : matrice des degrés
- $H^{(l)}$: caractéristiques des nœuds à la couche l
- $W^{(l)}$: matrice de poids apprenables
- σ : fonction d'activation (ReLU)

2.3 Encodage de position

On améliore les caractéristiques des nœuds par concaténation avec un encodage de position :

$$X_{\text{enhanced}} = [X \,||\, P]$$

où :

- $P \in \mathbb{R}^{n\times 2}$: matrice d'encodage de position
- $||$: opérateur de concaténation

2.4 Connexions résiduelles

Pour atténuer le problème de gradients évanescents, on utilise des connexions résiduelles :

$$H^{(l+1)} = H^{(l+1)} + W_{\text{skip}}H^{(l)}$$

où W_{skip} est une matrice de transformation apprenable.

3 Processus d'entraînement

3.1 Fonctions de perte

Le modèle optimise deux objectifs :

(1) Perte sur la prédiction de la distance (Huber Loss) :

$$L_{\text{dist}} = \frac{1}{n}\sum \text{huber}(d_{\text{pred}} - d_{\text{true}})$$

(2) Perte sur la prédiction du chemin (Binary Cross-Entropy) :

$$L_{\text{path}} = -\frac{1}{n}\sum \left[y_{\text{true}} \log y_{\text{pred}} + (1 - y_{\text{true}}) \log(1 - y_{\text{pred}})\right]$$

La perte totale s'écrit :

$$L_{\text{total}} = L_{\text{dist}} + \lambda L_{\text{path}}$$

où λ est un paramètre de pondération.

3.2 Apprentissage par curriculum

Trois niveaux de difficulté sont utilisés :

- **Facile** : graphes denses avec poids uniformes
  - Densité d'arêtes : 0.3 - 0.8
  - Poids : $[0.5, 5.0]$
- **Moyen** : graphes équilibrés
  - Densité d'arêtes : 0.3
  - Poids : $[0.1, 10.0]$
- **Difficile** : graphes clairsemés avec poids variés
  - Densité d'arêtes : 0.1 - 0.3
  - Poids : $[0.1, 20.0]$

4 Composants du modèle

4.1 Couches GCN

Le modèle utilise 4 couches GCN avec :

- Dimension cachée : 512
- Normalisation par batch
- Taux de dropout : 0.2
- Activation ReLU

4.2 Têtes de prédiction

- **Prédiction du chemin** :

$$P_{\text{path}} = \sigma(\text{MLP}(H^{(L)}))$$

où MLP est un réseau de neurones à 3 couches et $H^{(L)}$ la sortie finale.

- **Prédiction de la distance** :

$$P_{\text{dist}} = \text{MLP}(H^{(L)})$$

5 Métriques de performance

5.1 Précision sur la distance

Une prédiction est correcte si :

$$\frac{|d_{\text{pred}} - d_{\text{true}}|}{d_{\text{true}}} < 0.1$$

5.2 Précision sur le chemin

La prédiction du chemin est correcte si :

$$P_{\text{pred}} \cap P_{\text{true}} = P_{\text{true}}$$

avec P_{pred} et P_{true} les chemins prédit et réel.

6 Détails d'implémentation

6.1 Génération des données

Pour chaque graphe :

- Nombre de nœuds : $n \in [20, 100]$
- Densité d'arêtes : $p \in [0.1, 0.8]$
- Distribution des poids : $w \sim \mathcal{U}[a, b]$

6.2 Configuration d'entraînement

- Taille de batch : 256
- Taux d'apprentissage : 0.001
- Optimiseur : AdamW
- Décroissance du poids : 0.01
- Scheduler : CosineAnnealingWarmRestarts

6.3 Utilisation GPU

- Fraction mémoire CUDA : 0.8
- Normalisation batch
- Accumulation de gradients

7 Comparaison avec algorithmes classiques

Algorithme	Complexité temporelle	Complexité spatiale
Dijkstra	$O(\|E\| + \|V\| \log \|V\|)$	$O(\|V\|)$
Bellman-Ford	$O(\|V\|\|E\|)$	$O(\|V\|)$
Modèle GNN	$O(\text{batch\_size} \times \text{num\_epochs} \times \|V\|)$ (train) $O(\|V\|)$ (inférence)	$O(\|V\| \times \text{hidden\_dim})$

8 Améliorations futures

- Mécanismes d'attention :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Pooling de graphes :

$$H_{\text{pooled}} = \max_{\text{pool}}(H)$$

- Apprentissage multi-tâches :

$$L_{\text{total}} = \sum_i \lambda_i L_i$$

9 Visualisation des performances

9.1 Métriques d'entraînement

Nous visualisons :

- Courbes de perte :
  - Perte totale L_{total}
  - Perte distance L_{dist}
  - Perte chemin L_{path}
- Précisions :
  - Précision sur distance en fonction des époques
  - Précision sur chemin en fonction des époques
  - Distribution de l'erreur relative

9.2 Visualisation des graphes

Pour chaque prédiction, on visualise :

- Graphe d'entrée : positions des nœuds (layout force-directed), épaisseur des arêtes selon poids, source et cible mises en évidence
- Chemin prédit : chemin réel en vert, chemin prédit en bleu, erreurs en rouge
- Poids d'attention : scores d'attention sur nœuds et arêtes, heatmap de probabilité sur le chemin

9.3 Analyse des performances

- Matrices de confusion : précision chemin et distance
- Analyse des erreurs : par taille de graphe, densité, et poids
- Comparaison : GNN vs Dijkstra, GNN vs Bellman-Ford, temps d'entraînement

9.4 Outils de visualisation

Nous utilisons :

- NetworkX pour la visualisation de graphes
- Matplotlib pour les courbes de métriques
- Seaborn pour les visualisations statistiques
- Plotly pour les graphiques interactifs

Références

1. Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks.
2. Veličković, P., et al. (2017). Graph attention networks.
3. Hamilton, W. L., et al. (2017). Inductive representation learning on large graphs.

Licence