Learning Recruitment-Related Representations from Graphs and Sequential Data

Jun Zhu

université PARIS-SACLAY

Learning Recruitment-Related Representations from Graphs and Sequential Data

Apprentissage de la Représentation Liée au Recrutement à partir de Graphes et de Données Séquentielles

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°573, interfaces : matériaux, systèmes, usages (INTERFACES)
Spécialité de doctorat: Mathématiques appliquées
Graduate School : Sciences de l'ingénierie et des systèmes
Référent : CentraleSupélec

Thèse préparée dans l'unité de recherche MICS (Université Paris-Saclay, CentraleSupélec), sous la direction de Céline HUDELOT, Professeure, et l'encadrement de Paul-Henry COURNÈDE, Professeur.

Thèse soutenue à Paris-Saclay, le 28 Mars 2023, par

Jun ZHU

THESE DE DOCTORAT

NNT : 2023UPAST046

Composition du jury

Marc AIGUIER Professeur, Université Paris-Saclay	Président & Examinateur
Lynda TAMINE-LECHANI Professeure, Université Toulouse-III-Paul-Sabatier	Rapporteur & Examinatrice
Julien VELCIN Professeur, Université Lumière Lyon-II	Rapporteur & Examinateur
Fragkiskos MALLIAROS Maître de conférences, Université Paris-Saclay	Examinateur

université PARIS-SACLAY | ÉCOLE DOCTORALE Interfaces: matériaux, systèmes, usages

Titre: Apprentissage de la Représentation Liée au Recrutement à partir de Graphes et de Données Séquentielles
Mots clés: Recrutement, Graphe, Apprentissage de Représentations, Traitement de la Langue Naturelle

Résumé: L'E-recrutement est devenu un outil de recrutement essentiel dans la société moderne, facilitant le processus de recrutement et générant une quantité significative de données sous différents formats. Afin de gérer et d'analyser efficacement ces données, l'industrie a besoin de méthodes pour représenter ces données, car la représentation des données est fondamentale pour leur gestion et leur analyse. Inspirée par le succès de l'Intelligence Artificielle (IA) dans d'autres domaines, cette thèse vise à **exploiter l'IA pour aider à l'apprentissage de la représentation de données liées au recrutement**.
Les données de recrutement peuvent être organisées de manière explicite ou implicite en structures de graphe, telles que des taxonomies de compétences prédéfinies et des graphes de transition d'emploi construits à partir des expériences professionnelles. Malgré leur richesse en connaissances normatives, ces données structurées graphiquement sont moins utilisées. Dans ce contexte, cette thèse tente d'**exploiter ces données de recrutement structurées implicitement ou explicitement pour améliorer les représentations apprises liées au recrutement**.
Plus précisément, dans le Chapitre 4, nous apprenons des représentations de titres de postes à partir de graphes de transition enrichis. (ii) Dans le Chapitre 5, nous apprenons des représentations de compétences à partir d'un graphe de co-occurrence de compétences construit et d'une taxonomie de compétences prédéfinie. (iii) Dans le Chapitre 6, nous apprenons la préférence professionnelle d'un demandeur d'emploi à partir de séquences de demandes historiques afin de prédire le prochain emploi le plus probable auquel le demandeur d'emploi pourrait postuler.

Title: Learning Recruitment-Related Representation from Graphs and Sequential Data
Keywords: Recruitment, Representation Learning, Graph, Natural Language Processing

Abstract: E-recruitment has emerged as a crucial recruitment tool in modern society, facilitating the recruitment process and generating a significant amount of digital data in various formats. In order to manage and analyse this data effectively, the industry requires suitable methods to represent this data, as data representation is the basis for data management and analysis. Inspired by the success of Artificial Intelligence (AI) in other fields, this thesis aims to **leverage AI to help recruitment-related data representation learning**. Recruitment data can be organised explicitly or implicitly into graph structures, such as predefined skill taxonomies and job transition graphs built from work experiences. Despite containing a wealth of normative knowledge and additional information, these graphically structured data are less utilised. In such context, this thesis attempts to **exploit these implicit or explicit structured recruitment data to improve learned recruitment-related representations**.
Specifically, in Chapter 4, we learn job title representations from enriched job transition graphs built on work histories. (ii) In Chapter 5, we learn skill representations from a built skill co-occurrence graph and a predefined skill taxonomy. (iii) In Chapter 6, we learn the professional preference of a job seeker from historical job application sequences in order to predict the next most likely job that the job seeker might apply for.

Université Paris-Saclay

Doctoral Thesis

Learning Recruitment-Related Representations from Graphs and Sequential Data

Author:
Jun Zhu

Supervisor:
Prof. Céline Hudelot

A thesis was prepared under the supervision of Professor Céline HUDELOT

within the

Mathematics and Computer Science Laboratory for Complexity and Systems
CentraleSupélec, Université Paris-Saclay

Mars 28, 2023

Abstract

E-recruitment has emerged as a crucial recruitment tool in modern society, facilitating the recruitment process and generating a significant amount of digital data in various formats. In order to manage and analyse this data effectively, the industry requires suitable methods to represent this data, as data representation is the basis for data management and analysis. Inspired by the success of Artificial Intelligence (AI) in other fields, this thesis aims to **leverage AI to help recruitment-related data representation learning**.

Recruitment data can be organised explicitly or implicitly into graph structures, such as predefined skill taxonomies and job transition graphs built from work experiences. Despite containing a wealth of normative knowledge and additional information, these graphically structured data are less utilised. In such context, this thesis attempts to **exploit these implicit or explicit structured recruitment data to improve learned recruitment-related representations**.

Specifically, in Chapter 4, we learn job title representations from enriched job transition graphs built on work histories. (ii) In Chapter 5, we learn skill representations from a built skill co-occurrence graph and a predefined skill taxonomy. (iii) In Chapter 6, we learn the professional preference of a job seeker from historical job application sequences in order to predict the next most likely job that the job seeker might apply for.

Résumé

L'E-recrutement est devenu un outil de recrutement essentiel dans la société moderne, facilitant le processus de recrutement et générant une quantité significative de données sous différents formats. Afin de gérer et d'analyser efficacement ces données, l'industrie a besoin de méthodes pour représenter ces données, car la représentation des données est fondamentale pour leur gestion et leur analyse. Inspirée par le succès de l'Intelligence Artificielle (IA) dans d'autres domaines, cette thèse vise à **exploiter l'IA pour aider à l'apprentissage de la représentation de données liées au recrutement**.

Les données de recrutement peuvent être organisées de manière explicite ou implicite en structures de graphe, telles que des taxonomies de compétences prédéfinies et des graphes de transition d'emploi construits à partir des expériences professionnelles. Malgré leur richesse en connaissances normatives, ces données structurées graphiquement sont moins utilisées. Dans ce contexte, cette thèse tente d'**exploiter ces données de recrutement structurées implicitement ou explicitement pour améliorer les représentations apprises liées au recrutement**.

Plus précisément, dans le Chapitre 4, nous apprenons des représentations de titres de postes à partir de graphes de transition enrichis. (ii) Dans le Chapitre 5, nous apprenons des représentations de compétences à partir d'un graphe de co-occurrence de compétences construit et d'une taxonomie de compétences prédéfinie. (iii) Dans le Chapitre 6, nous apprenons la préférence professionnelle d'un demandeur d'emploi à partir de séquences de demandes historiques afin de prédire le prochain emploi le plus probable auquel le demandeur d'emploi pourrait postuler.

A Short Introduction in French

Aujourd'hui, la technologie de l'Intelligence Artificielle, en particulier l'émergence de l'apprentissage profond, a apporté des changements révolutionnaires dans de nombreux domaines tels que l'éducation, la santé, le commerce, les transports et le divertissement. Elle est considérée comme une force puissante capable de transformer nos vies quotidiennes tout en augmentant l'efficacité globale de la société.
Avec le développement continu de la société, la concurrence pour les ressources est devenue de plus en plus féroce, et beaucoup d'entre elles sont centrées sur les ressources humaines. Le talent que possède un pays, une institution ou une entreprise détermine sa position dans la compétition, et l'acquisition de talents doit se faire par le biais d'un processus de recrutement. Le recrutement est donc important pour le développement de l'ensemble de l'entreprise.
À cette fin, l'objectif global de cette thèse est d'étudier comment l'IA pilotée par les données peut aider et améliorer le domaine du recrutement.

L'IA Ouvre une Nouvelle Ère pour le Recrutement: Le domaine du recrutement a connu différentes périodes au fil de l'évolution de la science et de la technologie. Aujourd'hui, l'ère de l'IA, comme d'autres domaines, s'est progressivement renforcée avec le développement de la technologie de l'IA. Elle s'appuie sur les développements de l'ère numérique et utilise la technologie de l'IA pour améliorer les systèmes.
Avec l'aide de l'IA, le recrutement électronique est encore plus puissant. L'IA offre de nouvelles possibilités à l'ensemble du domaine du recrutement, mais en raison de la particularité de ce domaine, certains défis ont été rencontrés dans la mise en œuvre réelle, qui seront examinés dans la section 1.3. Les experts de l'industrie, les chercheurs et les spécialistes de l'Intelligence Artificielle ont encore beaucoup à explorer. **Par conséquent, la manière d'appliquer correctement la technologie de l'IA dans le système de recrutement pour le rendre plus efficace est un sujet d'importance pratique dans la réforme du domaine du recrutement, et c'est également la motivation de recherche de cette thèse**.

Randstad Veut un Système de Recrutement basé sur l'IA: Randstad est le leader mondial des services de ressources humaines. En tant que pionnier du secteur, Randstad a déjà expérimenté les technologies de l'IA à plusieurs étapes du processus de recrutement, telles que la sélection des CV, l'engagement des candidats, les entretiens avec les candidats et le développement de carrière.
Randstad soutient également la recherche universitaire dans l'espoir d'améliorer l'efficacité du recrutement en améliorant les méthodes existantes ou en développant de nouveaux outils avancés. C'est pourquoi cette thèse est financée par la chaire d'enseignement et de recherche «L'Intelligence Artificielle au service du recrutement»,[1] une collaboration entre Randstad France et CentraleSupélec. Cette chaire vise à améliorer et à faciliter le processus de recrutement en tirant parti des avancées et des progrès dans le domaine de l'IA et des données de recrutement riches. En particulier, cette thèse se concentre sur **l'apprentissage de la représentation des données de recrutement**, c'est-à-dire la représentation des talents, des emplois et des compétences, afin d'améliorer les tâches en aval telles que la classification et la recommandation des emplois ou des compétences.

Les Défis Posés par les Données de Recrutement: L'utilisation généralisée du recrutement en ligne a entraîné une explosion massive des données numériques sur le recrutement. Ces données sont diverses et complexes et posent de nombreux défis scientifiques:

[1] https://www.centralesupelec.fr/fr/randstad-et-centralesupelec-sassocient-pour-creer-une-chaire-sur-lintelligence-artificielle-et-le

- **Défi 1: Manque de données annotées supervisées**. Pour des raisons de confidentialité, il y a relativement moins d'ensembles de données annotées ouvertes dans le domaine du recrutement que dans d'autres domaines [Junhua Liu et al., 2019]. Ce manque d'ensembles de données annotées accessibles au public est problématique, car la plupart des approches de pointe axées sur les données de l'IA à haute performance reposent principalement sur un paradigme d'apprentissage supervisé, qui suppose la disponibilité de vastes ensembles de données annotées.

- **Défi 2: Les données changent constamment**. Les données de recrutement sont constamment mises à jour et itérées. La plupart des modèles d'IA actuels ne s'adaptent pas à ces variations de données. En effet, ils sont principalement basés sur l'hypothèse d'une distribution i.i.d. entre les données d'apprentissage et les données de production.

- **Défi 3: Faire bon usage des données implicitement ou explicitement structurées**. Les données de recrutement contiennent des données structurées cachées, principalement des graphes (réseaux), qui contiennent de nombreuses informations supplémentaires. Toutefois, ces informations supplémentaires sont rarement remarquées. Outre ces informations graphiques cachées, le domaine du recrutement contient également des données prédéfinies et bien structurées, telles que la taxonomie des compétences et des professions. Ces taxonomies contiennent souvent beaucoup de connaissances normatives, mais aucun travail ne les a encore utilisées, que ce soit directement ou indirectement. Par conséquent, l'utilisation de ces données implicitement ou explicitement structurées est un sujet stimulant et prometteur.

En conclusion, les défis mentionnés ci-dessus sont principalement dus à la particularité des données de recrutement. Dans cette thèse, **nous tentons de relever partiellement ces défis**. En outre, **nous utilisons des données de recrutement structurées implicites ou explicites (graphes) pour améliorer les représentations apprises liées au recrutement** afin de combler cette lacune de la recherche.

Tâches Traitées dans cette Thèse: Les tâches que nous abordons dans cette thèse sont:

- **Apprendre la représentation des titres d'emploi à partir des graphes**(Chapitre 4): apprendre des représentations de titres de postes à partir de graphes de transition enrichis.

- **Apprendre la représentation des compétences à l'aide d'un graphe hiérarchique** (Chapitre 5): apprendre des représentations de compétences à partir d'un graphe de co-occurrence de compétences construit et d'une taxonomie de compétences prédéfinie.

- **Prédiction de la prochaine demande d'emploi à partir de séquences de demandes d'emploi** (Chapitre 6): apprendre la préférence professionnelle d'un demandeur d'emploi à partir de séquences de demandes historiques afin de prédire le prochain emploi le plus probable auquel le demandeur d'emploi pourrait postuler.

Organisation de la Thèse: Le Chapitre 2 présente une étude des travaux connexes dans le domaine du recrutement. Le Chapitre 3 présente les connaissances préliminaires qui seront utilisées dans cette thèse, y compris les bases de l'intégration des graphes et de la recommandation séquentielle, ainsi que certains travaux représentatifs qui seront utilisés comme lignes de base dans les expériences. Dans le Chapitre 4, nous présentons notre méthode proposée pour l'apprentissage de la représentation des titres d'emploi. Le Chapitre 5 traite de l'efficacité de la taxonomie des compétences pour l'apprentissage de la représentation des compétences. Le Chapitre 6 présente notre méthode pour le problème de prédiction de la prochaine demande d'emploi. Pour les Chapitres 4, 5 et 6, nous présenterons les expériences et l'analyse des résultats. Enfin, dans le Chapitre 7, nous conclurons notre travail et envisagerons les travaux futurs.

Acknowledgements

Some words:

Three days before my defence, I wrote the following words, not feeling very calm at the time. However, aside from these words, all the gratitude expressed below is sincere and heartfelt.

As Howard once said, *"I guess the sad truth is not everyone will accomplish something great. Some of us may just have to find meaning in the little **moments** that make up life."*

While this doctoral thesis may not earn me any awards, it represents a small yet significant **moment** that makes up my life, and for that, it is undoubtedly meaningful. The giver of this meaning is not only myself, but also all the people mentioned below (If anyone is not mentioned, it is not because of my poor memory, but rather because my gratitude towards you is so great that it cannot fit on these few pages). The names are sorted alphabetically, without any preference.

To those who contributed to the existence of this thesis,

To Céline Hudelot, in addition to my deep gratitude towards her, another emotion I feel is admiration. This admiration came when, after making certain revision to a paper, she said to me, "Jun, I need to return to my role as a mother." She manages the balance between her personal and professional life perfectly, and she excels at both. For me, she is a great role model. I am very grateful to her for both her academic assistance and the spiritual strength she has given me.

To Paul-Henry Cournède, he is my "Bo Le" (a term commonly used in Chinese culture), which means that he is the person who appreciates me and is willing to give me opportunities. I am very grateful to him for his trust in me, for providing the opportunity to pursue my PhD in the MICS laboratory, and for all the help I have received during this doctoral period.

To Lynda Tamine-Lechani and Julien Velcin, I would like to express my gratitude to them for accepting to be my rapporteurs. They provided very detailed evaluations and made very constructive comments. To Marc Aiguier, Armelle Brun, and Fragkiskos Malliaros, I would also like to thank them for accepting to be my jury members, despite their busy schedules. Their valuable feedback and constructive suggestions have inspired me to think in new and innovative ways, and have greatly enriched the quality of my research.

To Gautier Viaud, I would like to thank him for his timely help (which was only two metres away) at the beginning of my PhD and for his meaningful suggestions after each discussion. To Hélène Theuré, I would like to thank her for her valuable comments and suggestions during our regular meetings. I also extend my gratitude to the Randstad corporate research chair for providing the financial support that made this thesis possible.

To those who created a comfortable environment for this thesis,

To Vincent Mousseau, I would like to thank him for patiently engaging in two conversations with me and for his warm and friendly greetings every time we met. I am thankful to the highly competent administrative and technical staff of the school, who helped me on many occasions with various administrative tasks and technical problems, allowing me to dedicate my time and efforts to my research. Thanks to Dany, Fabienne, Guillaume, Jana, Raphaëlle and Suzanne for everything they have done for me.

To François, Marc, Salim, Thomas, Victor B., Victor P., and Yassine, I would like to thank them for every discussion at the end of seminars and group meetings. Marc, thank you for showing me the ship-like building. Salim, thank you for working together late into the night and for your great coding skills. Yassine, thank you for your always-available help and the many warm conversations we've had.

To all colleagues who make the laboratory full of vitality, Aaron, Agathe, Andreas, Antonin, Brice, Elvire, Gautier, Gurvan, Hakim, Jad, Joseph, Laura, Leo, Lily, Ludovic, Mahmoud, Maria, Marin, Martin, Mathilde, Othmane, Paul, Quentin, Romain, Sarah, Stefania, Stergios, Sylvain, Véronique, Walid, Xiangtuo, Yoann, and anyone I may have missed. I would like to thank them for their kindness towards me. Thank them for the happy time we had in Corfu, playing Spikeball, eating crepes, playing codenames, bouldering, going to restaurants, and all the other great memories. I would like to express my gratitude to each of you personally, but I'm afraid I must cut this short and prepare my slides. Please be assured that when we next meet I will be sure to pass on my thanks.

To all friends who witnessed this thesis,

To Patrick and Zahra, thank you for making me feel at home in that lovely house on a certain road in Orsay, France. You are my family in France. I feel so lucky to have met you in 2016, and I hope my wish to become your "daughter" has come true. Patrik, every one of your dessert ideas is a success. Zahra, every word you say warms my heart. To Thiago, thank you for taking photos and playing Switch with me every summer.

To Brigitte, thank you for sharing your beautiful clothes and delicious meals with me. To Christophe, thank you for making me emotional as we edited my cover letter together. To Diane, thank you for each of your carefully prepared gifts. To Weiguo, thank you for helping me with mathematics and English. Thanks to Charles, Gustave and Plume for their company. I hope Charles will be happy in his next life.

To Antonin, Brice, Mathilde, and Myriam, thank you for the enjoyable time spent together, whether it was trying new pastries, drinking bubble tea, or exploring new restaurants. Antonin, thank you for your discussion on the cultural differences between China and France. Brice, thank you for driving me to the Guichet station and practicing Chinese with me so that I could improve. Mathilde, thank you for the lavender T-shirt and the bunny with my name. Looking forward to our trip to China! Myriam, thanks for your company in Lyon and for being so warm.

To Hepeng, Jiannan, and Lingzhou, thank you for all the delicacies I enjoyed with them in Orsay, Barcelona, and Changsha. A special thanks to Jiannan for always being there. To

Li, Peipei, and Shanshan, thank you for all the happy times we spent together shopping, taking photos and enjoying Shan Shan's desserts. To Boyang, Chao, Giuseppe, Jian, Siqi, Nan and Xuewen, thank you all for the BBQ and Restaurant Royal des Ulis. To Chaochao, Jiao, Lei, Mia, Peng, Sizhuo, Tao, Weihan, and Xi, thank you for your willingness to share your knowledge and experience, provide words of encouragement, and take on the important role of belaying each other. A special thanks to Lei for the wonderful yoga session that helped me calm down.

To Dan, Nan, Rui, Sheng, Xiaoyan and Yang, thank you the incredible memories we created together on our travels and bouldering adventures. And a special thanks to Dan for her amazing cooking and hospitality. To Chenmin and Jiao, thank you for your warm invitation and for sharing your interesting views. To Changzhen, Chenlin, thank you for your help with mathematics. To Ruiqing and Zheng, thank you for including me in the research project, and a special thanks to Zheng for helping me with my job search."

To my family who are proud of me, with or without this thesis,

To my family, thank you for always being proud of me and for providing a healthy environment for me to grow up in. A special thanks to my grandpa and grandma on the other side of the world, I miss you so much!

To Weichao, who is another **moment** of my life besides this thesis:

YES, LIFE IS BETTER WITH A DOG.

To my parents, who gave birth to the author of this thesis,

My parents, Guozhong Zhu and Xilu Guan, are my most important asset and greatest source of pride. Your unconditional love and support have enabled me to realize my dreams, including my Ph.D. journey. You are simply the best, and I will always love you.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence.
AR	Association Rule.
BERT	Bidirectional Encoder Representations from Transformers.
BiGRU	Bi-directional Gated Recurrent Unit.
BiLSTM	Bi-directional Long Short Term Memory.
BoW	Bag-of-Words.
BPR	Bayesian Personalized Ranking.
CBOW	Continuous Bag Of Words.
CNN	Convolutional Neural Network.
CNNs	Convolutional Neural Networks.
CommDGI	Community Deep Graph Infomax.
CV	Computer Vision.
DANN	Domain Adversarial Neural Networks.
DGI	Deep Graph Infomax.
DL	Deep Learning.
DM	Data Mining.
DNN	Deep Neural Network.
DNNs	Deep Neural Networks.
DSSM	Deep Structured Semantic Model.
ELMo	Embedding from Language Models.
ESCO	European Skills, Competences, Qualifications and Occupations.
FC	Fully-Connected.
GAL	Graph Attentional Layer.
GAN	Generative Adversarial Network.
GAT	Graph Attention Network.
GCN	Graph Convolutional Network.
GloVe	Global Vectors.
GNN	Graph Neural Network.

GNNs	Graph Neural Networks.
GRN	Graph Recurrent Network.
GRU	Gated Recurrent Unit.
GRU4Rec	Gated Recurrent Unit network for Recommendation.
HetG	Heterogeneous Graph.
HetGs	Heterogeneous Graphs.
HomG	Homogeneous Graph.
HomGs	Homogeneous Graphs.
HR	Human Resource.
IKNN	Item-based K-Nearest Neighbors.
IR	Information Retrieval.
ISCO	International Standard Classification of Occupations.
JD	Job Description.
KB	Knowledge-Based.
KNN	K-Nearest Neighbors.
LDA	Latent Dirichlet Allocation.
LINE	Large-scale Information Network Embedding.
LSTM	Long Short Term Memory.
M-NMF	Modularized Nonnegative Matrix Factorization.
MC	Markov Chain.
MF	Matrix Factorization.
ML	Machine Learning.
MLP	Multilayer Perceptron.
MRR	Mean Reciprocal Rank.
NDCG	Normalized Discounted Cumulative Gain.
NER	Named Entity Recognition.
NLP	Natural Language Processing.
NN	Neural Network.
NNs	Neural Networks.
O*NET	Occupational Information Network.
POS	Part Of Speech.
RGCN	Relational Graph Convolutional Network.

RNN	Recurrent Neural Network.
RNNs	Recurrent Neural Networks.
ROME	Répertoire Opérationnel des Métiers et des Emplois.
SG	Skip-Gram.
SGD	Stochastic Gradient Descent.
SKNN	Session-based K-Nearest Neighbors.
SNN	Siamese Neural Network.
SOC	Standard Occupation Classification.
TF	Term Frequency.
TF-IDF	Term Frequency Inverse Document Frequency.

Chapter 1

Introduction

1.1 Context and Motivation

Today, Artificial Intelligence (AI) technology, especially the emergence of Deep Learning (DL), has brought revolutionary changes to many fields, such as education, health, commerce, transportation, and entertainment. It is seen as a powerful force that can transform our daily lives while increasing the overall efficiency of society. For example, education applications [Singh et al., 2017; Nie, Brunskill, and Piech, 2021] use AI technology to automatically grade assignments, allowing teachers to spend more time instructing students. E-commerce [Smith and Linden, 2017; Jizhe Wang et al., 2018] is equipped with AI technology to improve the quality of product recommendations, thereby increasing revenue and improving user satisfaction. Healthcare organizations [McKinney et al., 2020] develop AI tool to help doctors detect cancer so patients can get timely treatment. Therefore, AI technology has attracted considerable interest in many research fields such as Computer Vision (CV) and Natural Language Processing (NLP), and various industries urgently need to implement this high-end technology and integrate it with the market.

With the continuous development of society, the competition for resources has become increasingly fierce, many of which are centered on human resources. The talents possessed by a country, an institution, or an enterprise determine its position in the competition, and talent acquisition needs to be done through a recruitment process. Therefore, recruitment is significant for the development of the whole society.

Aim of This Thesis

The global aim of this thesis is to investigate how data-driven AI can help and improve the recruitment domain.

In the following, we first explain the context and motivation of this thesis.

1.1.1 AI Opens up a New Era for Recruitment

Recruitment is the whole process of recruiting talent for vacancies, which can be generally divided into five steps,[1] including (i) *Recruitment Planning*, (ii) *Submit Resume/CV*, (iii) *Screening and Shortlisting*, (iv) *Interview and Assessment* and (v) *Offer Made*, as shown in Figure 1.1.

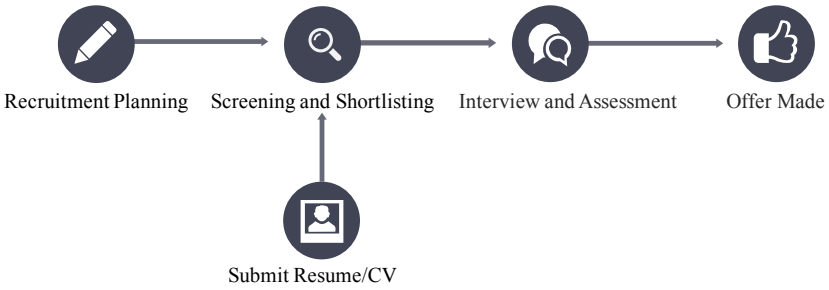


FIGURE 1.1: The recruitment procedure.

- **Step 1: Recruitment Planning**: Identification and analysis of vacant positions, including job duties, work responsibilities, and skill requirements, followed by preparation and assignment of job postings to attract talent.
- **Step 2: Submit Resume/CV**: Job seekers submit their resumes or Curriculum Vitaes (CVs)[2] to apply for the job, where they describe their abilities and explain why they are suitable for this job.
- **Step 3: Screening and Shortlisting**: Screening is the process of filtering the applications of job seekers based on their resumes/CVs for further selection. Shortlisting is the process of identifying short-listed candidates from the applicant pool who best meet the required criteria for the vacant position.
- **Step 4: Interview and Assessment**: The shortlisted candidates enter the interview process to assess their job-related knowledge, skills, and abilities, then determine which candidate should be further interviewed, hired, or excluded.
- **Step 5: Offer Made**: A formal offer can be made once a final candidate is identified and a background check is completed.

With the development of science and technology, these five steps have been affected to varying degrees, and a series of changes has taken place. Next, we briefly explore the different periods in the recruitment field. Similar to [*The Role of Digital Age in Recruitment* n.d.], we propose dividing the recruitment development into three periods based on the different tools used to attract qualified candidates and process information.

[1] https://en.wikipedia.org/wiki/Recruitment

[2] In this thesis, "resumes" and "CVs" are documents created by job seekers to show their profiles. Thus "resumes" and "CVs" are interchangeable.

(i) Print Media and Communication Tool Age began around the 1980s and continued until the late 1990s. Newspapers were the main channel for posting job advertisements. Other traditional ways included some communication tools such as fax machines, telephones, or distribution by television, radio, personal delivery, agents, and postal services. Such tools result in a lack of timeliness in the transmission of information and limited coverage. Moreover, during this period, resumes were usually in the paper version. Recruiters needed to browse a large number of resumes first and then manually selected potential candidates from these resumes. This pre-selection process was time-consuming, labor-intensive, and required recruiters to have relevant knowledge. In addition, the volume of data and the tediousness of this pre-selection process forced recruiters to take many shortcuts when reviewing resumes, which may lead to missing out on the best candidates.

(ii) Digital Age began in the 1990s and continues to the present. With the continuous and rapid development of the Internet, traditional recruitment methods have undergone revolutionary changes. E-recruitment/Online-recruitment, the process of recruiting candidates through the Internet to fill vacant positions, has become the primary recruitment channel for many organizations and companies at the time. They use different E-platforms, e.g., their own websites, third-party recruitment websites (e.g., LinkedIn,[3] CareerBuilder,[4] and Monster[5]), or social media platforms (e.g., Facebook,[6] Twitter[7], and Instagram[8]), to post job advertisements and accept resumes, in order to attract or find competent job seekers and provide them with more opportunities. Moreover, many organizations now include the Internet in the *Screening and Shortlisting* step, which saves a lot of time and resources, especially when more applicants respond. Online attitude and aptitude tests are becoming increasingly popular as a way to select suitable candidates. Video conferences have also emerged, allowing online interviews, job invitations, and online acceptances, as well as electronic signatures that allow applicants to accept and confirm job opportunities online. In addition to employers posting job postings through different media to attract talents, another trend is to create online databases where job seekers can store their resumes so that employers can search for candidates that meet their needs (e.g., LinkedIn,[9] Indeed,[10] and PhDTalent[11]). Such databases speed up the application process and allow people interested in changing jobs to prove their availability anytime without actively applying for jobs.

Since E-recruitment makes the recruitment process more productive as well as less costly, it provides more opportunities and liberates a lot of labor [Okolie and Irabor, 2017], which is beneficial to both the organization and the job seeker. E-recruitment is widely used and plays a vital role in the current recruitment process. As a result, the amount of online recruitment information (i.e., the personal information of employees and the recruitment information of employers) has increased dramatically in the past ten years. For example [*81 LinkedIn Statistics You Need to Know in 2022* n.d.], in 2021, there were 756 million users in more than 200 countries and territories worldwide,

[3] https://www.linkedin.com
[4] https://www.careerbuilder.com
[5] https://www.monster.com/
[6] https://www.facebook.com/
[7] https://twitter.com/
[8] https://www.instagram.com/
[9] https://www.linkedin.com
[10] https://www.indeed.jobs/
[11] https://careerfair.phdtalent.fr/en/home-2/

over 57 million companies listed on LinkedIn, with more than 15 million open job listings. Forty million people search for jobs weekly through LinkedIn, and three people are hired every minute on LinkedIn. This situation might be labor-intensive and exhausting for Human Resource (HR) management without dedicated decision-aided tools, requiring manual review of candidate resumes and assessment of applicant suitability for a position. Alternatively, it is difficult for job seekers to find jobs that suit them accurately and quickly. In this context, new decision-aided tools need to be conceived and developed to handle some basic operations.

To deal with such large amounts of data, a series of works [Shaha and Mourad, 2012] has been proposed over the past two decades to handle some basic operations on behalf of humans. These works revolve around different tasks of job recommender systems, such as **matching people and jobs** [Malinowski et al., 2006] and **classifying jobs** [D. H. Lee and Brusilovsky, 2007]. However, these systems use simple technologies, such as rule-based and statistics-based, and thus cannot cope well with growing data volumes and increasingly complex information in the E-recruitment field. Obviously, as E-recruitment continues to evolve, there is an urgent need for more effective technology to process and analyze these large amounts of data in the recruitment field. For example, a job seeker wants the system to recommend jobs that match his/her background to help him/her find suitable jobs efficiently. On the other hand, companies also hope that the system can improve the recruitment process to recruit qualified talents at a lower cost. In general, an effective E-recruitment system, which can automatically interpret the available information in the recruitment domain and measure how well talent qualifications match the requirements of recommended jobs, is a powerful assistant for the steady and rapid development of recruitment activities.

(iii) AI Age is gradually gaining power with the development of AI technology, as in other domains. The essence is to use AI technology to improve the system on the basis of the development of the digital age. AI techniques have received increasing attention in the recruitment industry [Upadhyay and Khandelwal, 2018; Van Esch, Black, and Ferolie, 2019]. These techniques are designed to systematically address the limitations of conventional recruitment procedures by incorporating AI-based methods, especially for repetitive, high-volume tasks. Such tasks include the **job classification** task, which classifies newly published job postings into predefined categories; the **skill identification** task, which extracts skills from job postings or resumes; and **person-job matching** task, which measures how personal profiles match job requirements, etc. Several existing works [Javed, Q. Luo, et al., 2015; Javed, McNair, et al., 2016; Zhu et al., 2018; Qin et al., 2018] have demonstrated that the use of AI techniques can effectively reduce the cost, time and effort required by traditional recruitment methods and improve the efficiency and functionality of E-recruitment systems. With the help of AI, E-recruitment is even more powerful,[12] i.e., it can (i) reach more potential candidates and identify outstanding talents more accurately, (ii) help talents have more opportunities to be hired and get more objective evaluations about their abilities and skills, (iii) make the hiring process faster, more efficient and less costly. Since AI technology has so many benefits in the recruitment field, especially for employers, organizations are starting to adopt and capitalize AI functionality in E-recruitment platforms, to build more powerful job recommender systems.

[12] https://en.wikipedia.org/wiki/Artificial_intelligence_in_hiring

AI brings new opportunities to the entire recruitment field, but due to the particularity of the recruitment field, some challenges have been encountered in the actual implementation, which will be discussed in Section 1.3. There is still a lot of room for exploration by industry experts, researchers, and AI scientists. **Therefore, how to correctly apply AI technology in the recruitment system to make it more effective is a topic of practical significance in the reform of the recruitment field, and it is also the research motivation of this thesis**.

1.1.2 Industrial Context: Randstad Wants an AI Recruitment System

Randstad[13] is the global leader in the HR services industry, which was founded in 1960 and is headquartered in Diemen, the Netherlands. As an industry pioneer, Randstad has already experimented with AI technologies at multiple steps of the recruitment process, such as resume screening, candidate engagement, candidate interviewing, and career development.[14] Randstad also supports academic research in hopes of improving recruitment efficiency by improving existing methods or developing new and advanced tools. Therefore, this thesis is funded by the *chaire d'enseignement et de recherche « L'Intelligence Artificielle au service du recrutement »* (chair of teaching and research « Artificial Intelligence at the service of recruitment »),[15] a collaboration between Randstad France[16] and CentraleSupélec.[17] This chair aims to improve and facilitate the recruitment process by leveraging the advances and progress in the field of AI and the richness of recruitment data. In particular, the research program of the chair includes:

- **Representation learning for recruitment**: Massive recruitment data requires us to find effective ways to represent them better, thereby facilitating follow-up tasks, e.g., job classification and job recommendation. Therefore, one of the research directions of this chair is to learn complete and accurate representations of various recruitment-related information (e.g., talent, job, skill, etc.) from diverse and heterogeneous data. These heterogeneous data include resumes, job postings, historical data, browsing data, profiles on professional social networks, video interview data, and conversation data. Moreover, although not at the core of the chair program, the question of fairness and interpretability of learned representation is also among the subjects of interest.
- **New soft skill analysis tool**: With the development of society, talent competition is becoming more and more fierce. In addition to the prominence of expertise, soft skills are another factor that must be considered when hiring. For example, analyzing the emotion and personalities of a candidate during a video interview can provide additional advice on candidate selection. In this context, proposing a new tool for analyzing candidate behavior (soft skill) is another research direction of the chair.

[13] https://www.randstad.com/

[14] https://www.randstad.com/workforce-insights/hr-tech/how-artificial-intelligence-human-resources-will-work-together-future/

[15] https://www.centralesupelec.fr/fr/randstad-et-centralesupelec-sassocient-pour-creer-une-chaire-sur-lintelligence-artificielle-et-le

[16] https://www.grouperandstad.fr/

[17] https://www.centralesupelec.fr/

- **Design and optimization of algorithms:** Job recommender systems are the core of the entire recruitment system. They aim at retrieving a list of job positions that satisfy the desire of job seekers or the best set of potential candidates that meet the requirements of recruiters. An efficient job recommender system can make the recruitment process simpler and less costly, so building a job recommender system [Al-Otaibi and Ykhlef, 2012; Geyik, Ambler, and Kenthapadi, 2019; Kenthapadi, B. Le, and Venkataraman, 2017] is usually regarded as the goal of many organizations, and naturally, it is also the goal of Randstad. Around the job recommender system, many related algorithms need to be optimized or redesigned, including but not limited to person-job matching algorithms, search techniques, and candidate or job recommendation algorithms.

- **Knowledge representation and reasoning for the recruitment domain** : Another scientific objective of this chair is related to the management and improvement of knowledge in the recruitment field. Indeed, the field of recruitment has long been an area where knowledge representation and reasoning tools, such as ontologies, have been studied and considered to have great potential [Bizer et al., 2005; Malherbe and Aufaure, 2016]. Consequently, many knowledge models have been proposed in the literature over the last decade, such as skill ontologies [Lundqvist, Baker, and Williams, 2008; Fazel-Zarandi and Fox, 2009; Malherbe and Aufaure, 2016], job ontologies [Khobreh, 2017], or, more generally, models of the Human Resources domain [Kessler, Lapalme, and Tondo, 2016]. This is also a general trend among the players in the field, as they have identified the potential of these approaches in improving the recruitment process (e.g., Textkernel, Google). Around knowledge models, there are many research lines, such as ontology matching to solve interoperability problems, knowledge-rich learning models, and many others.

This thesis is part of this large research program. Its objective is not to tackle all aspects of the program. Our work mainly involves the first and third aspects. In particular, this thesis focuses on **recruitment data representation learning**, i.e., the representation of talents, jobs, and skills, to improve downstream tasks such as job or skill classification and recommendation. We mainly focus on the use of **Deep Learning (DL)**-based models.

1.1.3 Why Deep Learning?

Many practical applications that are frequently used in different fields, such as machine translation, recommendation systems, image segmentation, and speech recognition, used to rely heavily on manual feature engineering to extract informative features and human analysis to make decisions. However, recent advances in AI, especially via the DL paradigm, have shown that they can speed up these operations and significantly improve application performance, thanks to their ability to learn complex representations of data [Bengio, Courville, and Vincent, 2013]. Similarly, DL can also be widely used in the field of recruitment to help various operations, thereby improving efficiency. The main reasons are [S. Zhang et al., 2019]:

- **Effective for Representation Learning**: Deep Neural Networks (DNNs) can effectively learn useful representations from the input data [Bengio, Courville, and Vincent, 2013], such as Convolutional Neural Networks (CNNs) [Ye Zhang and Wallace, 2015] for images, Recurrent Neural Networks (RNNs) [Sundermeyer, Schlüter, and Ney, 2012] for text and Transformers [Vaswani et al., 2017] for text, images and other structured data. The widespread use of E-recruitment brings with it large amounts of digital data that recruitment systems need to handle properly, and one of the important considerations is how to represent them effectively. Most of the recruitment data are non-structured or semi-structured textual data. **Therefore, DNNs are a natural choice for learning recruitment-related representations**.
- **Effective for Sequence Modeling**: DNNs have shown promising results on many sequence modeling tasks such as machine translation [Bahdanau, Cho, and Bengio, 2014], speech recognition [Graves, Mohamed, and Hinton, 2013], and next-item/basket recommendation [F. Yu et al., 2016; Hidasi, Karatzoglou, et al., 2015; Hidasi, Quadrana, et al., 2016]. In the recruitment system, some scenarios need to process sequential data, such as predicting the next job a talent will take based on his/her previous work experience and predicting the next job a job seeker will apply for based on his/her application history. **Therefore, DNNs are well suited for this sequential pattern mining task**.
- **Effective for Graph Analysis:** Recently, encouraged by the success of DL on other domains, e.g., images, texts, and videos, a large number of methods [J. Zhou, G. Cui, S. Hu, et al., 2020; Z. Wu et al., 2020] have extended the DNNs for graph data. Such methods are called graph embeddings and Graph Neural Networks (GNNs), and have been shown to be effective on various tasks such as node classification and link prediction. Graphs/networks are ubiquitous in recruitment data, such as job transition histories, skill-occupation demand relationships, and occupation taxonomies. These graphs reflect simple/complex, static/dynamic item relationships in low/high-linked data so that recruitment data can be analyzed against these graphs. **Therefore, graph representation learning can be an interesting research line for recruitment data, and GNNs can be an effective tool**.

1.2 Available Data in the Recruitment Field

As we said before, the widespread use of E-recruitment has led to a massive explosion of digital recruitment data. These data are diverse and complex, bringing many scientific challenges. In this section, we focus on the different data types and their characteristics and analyze the challenges posed by these characteristics in the next section.

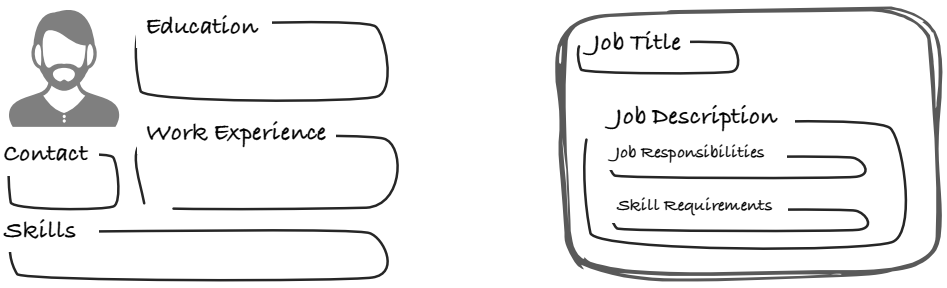
Recruitment data can be of the following types:

- **Long text**: Most recruitment data fall into this type, with job postings and resumes being the most common. As both employers and talents turn to online employment portals, they receive/send large amounts of job postings and resumes, which are usually loaded/uploaded as textual digital documents in different formats such as .pdf, .doc, .html, or .rtf. These documents are usually relatively long, and non-structured or semi-structured (i.e., resumes and job postings are organized in sections).

- **Short text**: Unlike resumes and job postings, which usually have hundreds or even thousands of words, there are also some short text data, such as messages sent in chatbots, responses sent by emails, and assessments recorded during interviews.
- **Terminological resouces, ontologies**: To standardize the field of recruitment, some organizations establish different terminology systems and references. For example, O*NET (B.2.3), SOC (B.2.2), ISCO (B.2.1) and ESCO (B.2.4) contain hundreds of occupation and skill definitions, respectively. Moreover, companies also have their own references or ontologies.
- **Interaction records**: Interaction records include all actions taken from the talent or organization side, such as *click*, *share* and *apply* operations when a talent views job postings, as well as *keep* and *remove* operations when a recruiter views application records. These actions are recorded in a time-series format, usually divided into different sessions based on a specific period (i.e., an hour or a day).
- **Video**: Video interviews, like face-to-face interviews, are about assessing the qualification of a candidate by asking job-related questions. The difference is that video interviews are conducted remotely, using video technology as a medium of communication, and the interview of the candidate can be recorded in the video.
- **Other**: In addition to the above data types, there also exist some statistics generated by online attitude and aptitude tests, as well as the information scanned from social media, such as code on GitHub and portfolios on personal websites.

While textual data is not the only type of recruitment data, in this thesis, we only focus on textual data regardless of the format in which it is presented. In the following, we further characterize these textual recruitment data:

- **Characteristic 1: Non-unified structured data.** Job postings and resumes are the most common long-text data, and these documents have a general structure, e.g., *Education* and *Work Experiences* sections in resumes, or *Job Title*, *Job Description* and *Company Description* in job postings, as shown in Figure 1.2. However, there is no standard structure, and the corresponding content is expressed freely. In addition, to make the resume look more attractive, it may contain tables and other formats such as bar charts, progress bars, and pie charts.



(a) A resume template. (b) A job posting template.

FIGURE 1.2: A resume with *Education* and *Work Experiences* sections and a job posting with *Job Description* and *Skill Requirements* sections.

- **Characteristic 2: Free, messy and vague vocabularies and terms.** Data in the recruitment field is highly messy since it is expressed freely. People have different interpretations of words and ideas, they express themselves freely in various ways, such as job titles/descriptions used in resumes and job postings. Therefore, it is not easy to collect and organize useful information.
- **Characteristic 3: Strong country/industry/company-specific references and standards.** Generally speaking, the recruitment domain varies from country to country, industry to industry, or company to company. Although there are some standard references (i.e., O*NET[18] and SOC[19]), companies prefer to define their own naming conventions, occupational taxonomies, and evaluation standards. They use these reference materials and standards to write job postings, manage human resource systems and recruit talents. All these make the standardization and unification of the recruitment field very challenging.
- **Characteristic 4: Dynamic information and evolving knowledge.** In addition to the changes in references and standards between different countries, industries, and companies mentioned above, the recruitment field also changes from time to time. One aspect is the fast update and growing number of items (i.e., a large number of job postings are added or removed daily, and hundreds of candidate profiles are created or updated). Another is the evolving recruitment field (i.e., new occupations, new formations, new skills, and new terms). All of these reflect the dynamic and evolving nature of the recruitment field.
- **Characteristic 5: Privacy.** Data privacy is a top priority for every company, especially in recruitment. E-recruitment platforms collect a large amount of personal data of each talent, including nationality, email address, phone number, age, race, work experience, and photos. All these data are very private and important. If they are used carelessly by a malicious third party, in addition to causing mental harm and property damage to talents, it will also greatly impact the reputation of the information collectors (E-recruitment platforms). For this reason, collecting datasets in the recruitment field is much more complicated than in other areas, especially annotated datasets.
- **Characteristic 6: Strong domain knowledge reliance.** Unlike other domains, data processing in the recruitment field requires a certain professional background. For example, the same skill may be expressed differently, or the same term may mean different skills in different contexts (e.g., "Java" in *Java coffee* and *Java programming language*).

1.3 Challenges Posed by Recruitment Data

The newly available recruitment big data enables researchers to conduct recruitment analysis in more quantitative ways. Significantly, AI techniques, especially NLP, have been extensively studied in the recruitment domain from academia and industry. Although many studies have demonstrated their effectiveness, there are still many challenges, some of which are due to the specificity of recruitment data. In this section, we summarize these challenges:

[18] http://www.onetcenter.org/taxonomy/2010/list.html
[19] https://www.bls.gov/soc/2018/home.htm

- **Challenge 1: Lack of supervised annotated data**.
  Due to privacy reasons (i.e., Characteristic 5), there are relatively fewer open annotated datasets in recruitment compared to other fields [Junhua Liu et al., 2019]. This lack of publicly available annotated datasets is problematic because most state-of-the-art high-performance AI data-driven approaches are mainly based on a supervised learning paradigm, which assumes the availability of large annotated datasets.[20] For example, we can take the Person-Job Fit problem, one of the most striking tasks in the recruitment field. This task aims to measure the matching degree between person-job pairs, where the person and the job are mainly represented using their corresponding textual information (resume and job posting). Many approaches with different deep architectures have been proposed in the literature [Qin et al., 2018; Zhu et al., 2018; Bian, W. X. Zhao, et al., 2019]. These methods typically deal with the Person-Job Fit task as a supervised binary text matching problem by training a matching label classifier based on a set of labeled person-job matching records (i.e., the label of a person-job pair label is 1 if the person got the job, and 0 otherwise). The amount and quality of available matching records directly affect the performance of Person-Job Fit algorithms, and it is unrealistic to assume either the availability or the annotation work of the needed dataset.[21] Therefore, this lack of annotated datasets challenges the use of purely supervised learning AI and brings the research to other learning schemes such as unsupervised or semi-supervised learning. An interesting way we explore in this thesis is the use of graphs to aid data learning.

- **Challenge 2: Data is constantly changing**.
  Recruitment data is constantly updated and iterated (i.e., Characteristic 4). Most current AI models are not adaptive to these data variations.[22] For example, there is a lot of research on job classification [Javed, Q. Luo, et al., 2015; Javed, McNair, et al., 2016], which is a task to classify jobs into predefined categories (e.g., SOC and O*NET). The classifier output of these job classification models is usually a probability vector whose dimension is the number of available occupation categories. However, new occupations may be created due to the evolving nature of the recruitment domain, and most of the time, taking into account these new occupations means retraining the whole model on an updated dataset. The AI community tackles this important issue as the open set recognition problem.

- **Challenge 3: Explanation and fairness are needed**.
  The recruitment process is a series of decision-making tasks, from the initial screening of resumes to deciding whom to hire [Schumann et al., 2020]. Explaining each decision-making process, e.g., why resumes are selected and what skill deficiencies caused the job seeker to be incompetent for the position, can improve the transparency, persuasiveness, effectiveness, credibility, and user satisfaction of the recruitment results. Moreover, the decision-making process needs to comply with many AI policy and regulation initiatives, such as the GDPR[23] in Europe. There is a rapid increase in the literature on fair and explainable AI [Yongfeng Zhang

[20] See, for instance, the importance of the ImageNet dataset in the advancement of Deep Learning for visual recognition.

[21] Except for big companies like LinkedIn can build pseudo-annotated datasets from the interactions of their users on their websites, but such datasets are private.

[22] Indeed, they are mostly based on the assumption of i.i.d distribution between training data and production data.

[23] https://en.wikipedia.org/wiki/General_Data_Protection_Regulation

and X. Chen, 2018; Arrieta et al., 2020], but the private nature of recruitment data makes it particularly challenging.

- **Challenge 4: Make good use of implicitly or explicitly structured data**.
  The data processed and analyzed in the existing works mainly include: (i) *person-job matching records* for person-job fit [Zhu et al., 2018; Bian, W. X. Zhao, et al., 2019], (ii) *career trajectories* for job mobility analysis [L. Li et al., 2017; Meng et al., 2019], and (iii) *jobs or resumes* for classification [Javed, Q. Luo, et al., 2015; Jingya Wang et al., 2019], skill extraction/analysis [M. Zhao et al., 2015; T. Xu et al., 2018], or representation learning [Junhua Liu et al., 2019; Denghui Zhang et al., 2019], as summarized in Table 2.7. There are some hidden structured data in these data, mainly graphs (networks), which contain a lot of extra information. However, this additional information is rarely noticed, except that [Dave et al., 2018; Denghui Zhang et al., 2019] learn job title representations from graphs. In addition to this hidden graph information, the recruitment domain also contains some predefined well-structured data, such as skill and occupation taxonomy. These taxonomies often contain much normative knowledge, but no work has yet made use of it, whether directly or indirectly. Therefore, using these implicitly or explicitly structured data is a challenging and promising topic.

In conclusion, the above challenges are mainly caused by the particularity of recruitment data. In this thesis, we attempt to **address these challenges partially**. Additionally, we **leverage implicit or explicit structured recruitment data (i.e., graphs)** to improve the learned recruitment-related representations to fill this research gap. The tasks we deal with in this thesis are outlined in subsequent sections and chapters.

1.4 Tasks Handled in this Thesis

1.4.1 Job Title Representation Learning from Graphs

Objective of This Task

Learn better job title representations to facilitate various subtasks of the recruitment process, such as improving talent profile modeling, facilitating search and recommendation, and benefiting the recruitment analysis.

A job title is a short text that uses a few words to describe a job position, but it usually carries essential information, indicating the primary responsibilities and the level of the position. It appears in many scenarios, such as resumes and job postings, which is the first part of attracting job seekers or employers. Learning good job title representation benefits three important downstream tasks: (i) improving the modeling of talent profiles by using, for instance, their working histories, (ii) facilitating the job and skill alignment, and (iii) benefiting the recruitment data analysis. However, learning the proper job title representation is a challenging task for the following reasons:

- **Noisy data** (i.e., **Characteristic 1**): job title data is noisy due to personal subjective reasons (i.e., spelling errors) or objective reasons (i.e., the resume parser is not perfect).
- **Messy data** (i.e., **Characteristic 2**): job titles are messy because people have different ways of thinking. For example, there are many alternative job titles for the same position, e.g., "purchasing clerk" and "buyer". Another problem is that due to the ambiguity of certain terms, they can refer to different positions in different contexts, e.g., "registered nurses sandwich rehab" and "sandwich maker".
- **Non-standard naming conventions** (i.e., **Characteristic 3**): naming conventions vary by company, industry, and country. Therefore, there are a lot of subjective and non-standard naming conventions for the same position.

For these reasons, standard approaches that aggregate (e.g., mean or sum) word representations to get job title semantic representations may lead to mismatches. Moreover, these semantic-based methods ignore hidden relationships between job titles, e.g., titles in the same resume may be similar. In order to meet these challenges, we propose to learn job title representations from graphs generated from the working history of talents. Indeed, in recent years, graph/network embedding technology has been widely used in many scenarios, especially the node representation learning problem. The learned representations are beneficial to various prediction and analysis tasks in many application fields, ranging from computational social science to computational biology [Hamilton, Ying, and Leskovec, 2017a]. As mentioned earlier, using graphs can also help with data learning.

Research Scope Inspired by the achievements of graph/network embedding and taking advantage of the benefit that graphs can aid data learning (i.e., Challenge 1), we propose leveraging graphs for job title representation learning. Such graphs can be job-transition graphs extracted from the talent career experience but can also be extended with additional information, leading to heterogeneous graphs (different types of node and types of edges, see, for instance, Figure 1.3).

Talent A automotive technician → automotive shop manager.

Talent B purchase agent → purchasing manager → staff account purchasing manager.

Talent C staff accountant → staff account purchasing manager → purchasing manager.

Talent D customer service → telemarketer → purchasing clerk.

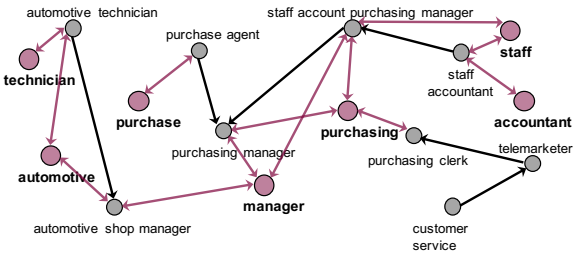


FIGURE 1.3: *Job-Transition-Tag Graph* with grey/purple circles representing job titles/tags. Black lines represent job transitions and purple lines represent "has/in" relationships.

In this task, we focus on answering the following Research Questions (RQs):

Research Question 1.4.1.1 : *Can the graph structure provide more useful information for job title representation learning?*

Research Question 1.4.1.2 : *Does additional information resulting in complex heterogeneous graph help learn better job title representations?*

1.4.2 Skill Representation Learning by Leveraging Hierarchical Graph

Objective of This Task

Learn better skill representations by leveraging the predefined taxonomy data, which can aid occupation/job/talent classification.

Skills are technical knowledge (hard skills) or personal traits (soft skills) that arise in many scenarios in the labor market. For example, a set of basic skills is predefined for each occupation, job requirements are mainly reflected through skills requirements in job postings, and talents attract employers by expressing their skills on resumes, as shown in Figure 1.4.

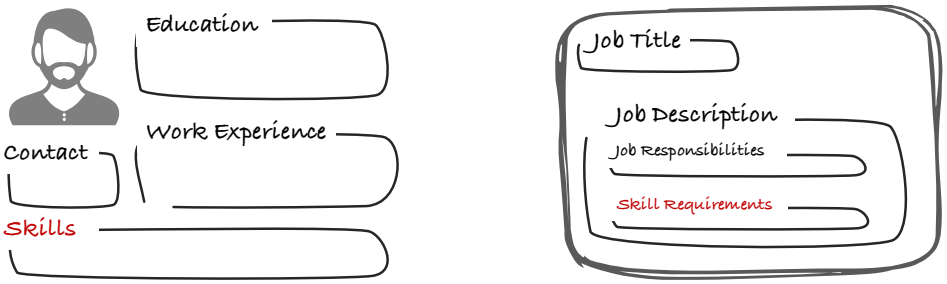


FIGURE 1.4: The scene where the skill appears.

Many recruiting activities are primarily skills-based, e.g., using skills to quantify talents, recruiting vacancies by measuring the skill fit, and alleviating the skill gap to increase opportunities for talents and employers. Therefore, how to represent skills correctly and effectively is an essential task in the recruitment field. Since occupations/job postings/talents can be represented as a set of skills they require or possess, we can also take benefit of graphs for this specific task. In particular, we can build a *skill co-occurrence graph*. Vertices represent skills and edges represent co-occurrence relationships, i.e., if two skills are required or possessed in the same job posting or resume, an edge is added between them. Moreover, in the recruitment field, skills tend to be in a tree-like structure, representing a hierarchical relationship, called a *skill taxonomy*. There are many predefined *skill taxonomies* in the recent labor market, such as SOC,[24] O*NET-SOC,[25] ISCO,[26] and ESCO.[27] These taxonomies provide a standardized language for the activities of people in the labor market [Ospino, 2018]. While this information is well-structured and informative, it has rarely been studied directly and, to the best of our knowledge, has never been used for skill representation learning. Therefore, in this task, we

[24] https://www.bls.gov/soc/2018/home.htm
[25] https://www.onetcenter.org/taxonomy.html
[26] https://www.ilo.org/public/english/bureau/stat/isco/isco08/
[27] https://ec.europa.eu/esco/portal

exploit this hierarchical information, i.e., the *skill taxonomy* to learn better skill representations, along with the *skill co-occurrence graph*.

Research Scope For this task, our orientation is the same as for the previous task, i.e., taking advantage of advances in graph representation learning. More precisely, we draw our inspiration from graph embedding works that try to preserve both the neighborhood information and the community structure of graphs. In our case, not only the node proximity in the *skill co-occurrence graph* needs to be preserved, but also the information carried by *skill taxonomy* needs to be considered. More specifically, the *skill taxonomy* is similar to the hierarchical community structure, where communities at lower levels are smaller and more cohesive than larger communities at higher levels. This hierarchy of communities reflects similarities at different granularities. In other words, skills belonging to the same lower-level skill category have more specific similarities, e.g., "algebra" and "probability theory" are skills that mathematics teachers need. In contrast, skills that belong to the same higher-level skill category are similar in more general contexts, e.g., "algebra" and 'biomedicine" are science-related skills. To this end, we exploit the hierarchical community preserving methods to improve the informativeness of learned skill representations.

In this task, we focus on answering the following Research Questions (RQs):

Research Question 1.4.2.1 : *Can the graph structure provide more useful information for skill representation learning than the semantic representation?*

Research Question 1.4.2.2 : *Does the hierarchical information (i.e., skill taxonomy) contribute positively to skill representation learning?*

1.4.3 Next-Application Prediction from Job Application Sequences

Objective of This Task

Recommend the next job application (i.e., the job posting that the job seeker will apply for) based on the historical application record of job seekers, which is an important task of the job recommender system.

Next-job prediction is one of the vital tasks of job recommender systems, where the job can be the next job position or the next job application.

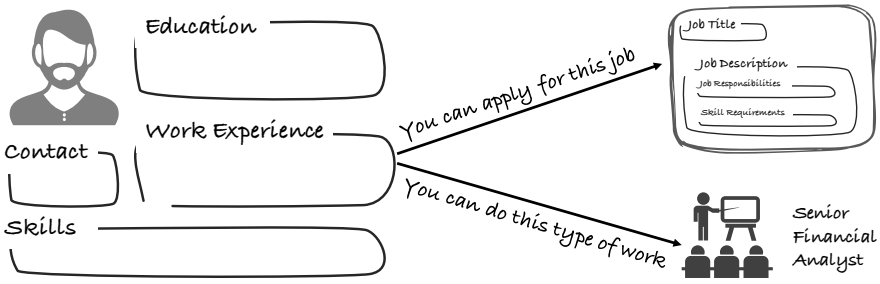


FIGURE 1.5: Illustration of *Next-job application* prediction task.

In this work, we target to predict the next job application for which the job seeker might apply. We propose a hybrid Personalized-Attention Next-Application Prediction model (PANAP), which aims to answer the following issues partially:

- **Evolving Job-Person interactions** (i.e., Characteristic 4): The first issue is related to intrinsically dynamic and constantly evolving Job-Person interactions, which can be built from job application records or working histories. We assume that the temporal relation between application records tends to point towards sequential/session-based recommendations. Although various session-based methods have been proposed in other fields, there is less research on session-based job recommendations, especially the next job application prediction.
- **Constantly updated items** (i.e., Characteristic 4): The second issue is also due to the evolving nature of the recruitment field. The constantly updated E-recruitment website and domain intensify the cold-start problem. Therefore, building a static method that cannot be adjusted quickly as the data constantly changes is impractical.
- **Data privacy** (i.e., Characteristic 5): The third issue is related to the sensitivity and privacy of data in the recruitment domain. Therefore, collecting datasets in the recruitment field is much more complicated than in other domains, especially annotated datasets.
- **Personalization of recommendations**: The last issue is related to the personalization of recommendations. Indeed, two job seekers can apply for the same job with different motivations. As a consequence, the important information in a job is specific to the job seeker. Thus, weighing the jobs differently in the modeling tasks of job seeker careers is essential. General-purpose approaches are often unable to capture such specific information. In addition, the geographic location of job seekers often has a substantial impact on the application decision and the recommendation result, but few works consider this information.

Research Scope For this task, our positioning is to model the next-job prediction problem as a sequential recommendation problem, recently studied in other domains such as news or product recommendation [Souza Pereira Moreira, Ferreira, and Cunha, 2018; J. Li et al., 2017; F. Yu et al., 2016; Fang, Danning Zhang, et al., 2020]. Moreover, we also want to provide personalized recommendations. For this, we rely on the well-known attention mechanism [Bahdanau, Cho, and Bengio, 2014; Vaswani et al., 2017] (see Appendix F).

In this task, we answer the following Research Questions (RQs):

Research Question 1.4.3.1 : *In the recruitment field, do we really need the sequential recommendation method?*

Research Question 1.4.3.2 : *Can the personalized-attention mechanism better capture the personal career preference?*

Research Question 1.4.3.3 : *Are job textual information and context information (e.g., geographical location and educational background) important in job recommendation?*

Research Question 1.4.3.4 : *How to take into account context information in the recommendation process?*

1.5 Thesis Organization

Chapter 2 presents a survey of related works in the field of recruitment. Chapter 3 gives the preliminary knowledge that will be used in this thesis, including the basics of graph embedding and sequential recommendation, as well as some representative works that will be used as baselines in experiments. In Chapter 4, we introduce our proposed method for job title representation learning. Chapter 5 discusses the effectiveness of skill taxonomy for skill representation learning. Chapter 6 introduces our method for next-application prediction problem. For Chapter 4, 5 and 6, we will demonstrate the experiments and the analysis of results. Finally, in Chapter 7, we will conclude our work and look forward to future work.

Chapter 2

Related Works in the Recruitment Field

Recruitment is an essential process in the highly competitive labor market. The traditional methods of recruitment mainly rely on relatively "low" technology. For example, employers distribute job postings through the newspaper, radio, television, or courier service. In addition, the recruitment process is almost manual and requires a lot of human resources. In recent years, as E-recruitment becomes more and more common, Artificial Intelligence (AI) techniques, such as Machine Learning (ML), Information Retrieval (IR), and Data Mining (DM), have been applied to many models to handle various tasks in the recruitment market. To help us gain an extensive overview of the current industry and guide our research, we review the latest works related to E-recruitment in this chapter. We categorize the existing methods into five categories based on the information they processed: (i) *Skill*, (ii) *Job title*, (iii) *Career path*, (iv) *Matching record*, and (v) Other, as illustrated in Figure 2.1.

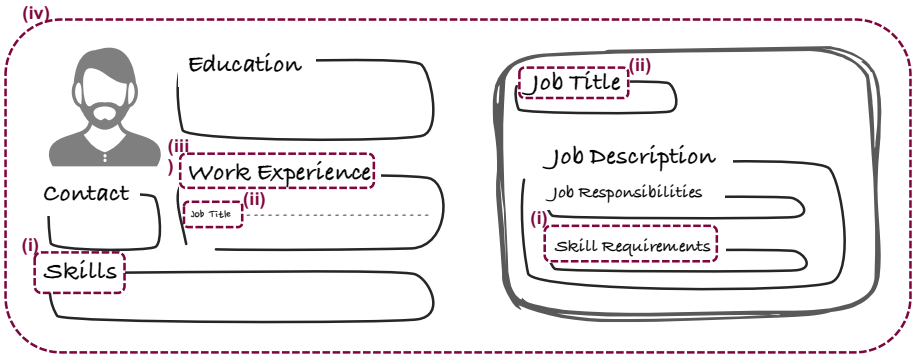


FIGURE 2.1: Illustration of the different types of information processed in recruitment-related works.

Although a paper may belong to more than one category for each evaluation axis in many cases, we chose the most predominant one for our categorization. A summarization is shown in Table 2.1. We start with a description of terminologies and a definition of notations. Then, we present the methods and techniques used in existing works in the recruitment domain.

2.1 Terminology and Notation

In practice, recruitment data usually consist of three parts, namely (i) job posting, (ii) resume, and (iii) interaction history. Specifically, a job posting contains a job title and description (e.g., job duties and requirements). A resume contains the profile of a talent. The interaction history records the behaviors of talents regarding the available job information (e.g., click, apply and share). In the following, we first define the terminology used frequently throughout this thesis.

- **Employer**: is an individual, company, or organization that employs people.
- **Employee**: is an individual who is paid for working for an employer.
- **Talent**: is an individual who has a specific set of skills or abilities.
- **Job Seeker**: is an individual who is looking for a job.
- **Applicant**: is an individual who applies for jobs by sending a resume or otherwise.
- **Candidate**: is an individual who has applied for a particular job position and is being further considered by the employer.
- **Skill**: is the technical knowledge (hard skill) or personal trait (soft skill) required for a job position. It usually appears in job postings or resumes.
- **Resume/Curriculum Vitae (CV)**: is a formal document created by a job seeker to show his/her personal information (e.g., age, gender, and interests), professional background (e.g., education background, and work experience), and abilities (e.g., hard skills and soft skills).
- **Career Path/Trajectory/History**: is the professional growth in the career life of an individual. It comprises several distinct career stages, and each stage contains a set of work experience items.
- **Job Posting/Advertisement/Announcement**: is an informative text that describes a vacant position. It usually consists of *job title*, *company description*, *job responsibilities*, *skill requirements*, and *application instructions*. In this thesis, *job responsibilities*, *skill requirements* are combined as *job description*.
- **Job Title**: is a set of several important terms used to describe the job position held by the job seeker (described in the resume) or defined by the employer (published in the job posting). It is the first part of attracting job seekers or employers, indicating the primary responsibilities and position level.
- **Job Description (JD)**: is a textual statement that provides detailed information about a specific position. It details related duties, responsibilities, and skill requirements (e.g., qualifications about skills or experiences).

TABLE 2.1: Recruitment-related works categorized by information handled.

Information	Task	Paper
Skill	Skill Extraction	[Kivimäki et al., 2013; M. Zhao et al., 2015; Javed, Hoang, et al., 2017; Sayfullina, Malmi, and Kannala, 2018; Khaouja et al., 2019; Gugnani and Misra, 2020; Wings, Nanda, and Adebayo, 2021]
	Skill Analysis	[W. Zhou et al., 2016; T. Xu et al., 2018; Börner et al., 2018]
	Skill Representation Learning	[Dave et al., 2018; Nigam et al., 2020]
Job Title	Job Title Representation Learning	[Dave et al., 2018; Junhua Liu et al., 2019; M. Liu et al., 2019; Denghui Zhang et al., 2019; Decorte et al., 2021; L. Zhang et al., 2021; Wings, Nanda, and Adebayo, 2021; Yamashita et al., 2022]
	Job Title Classification	[Javed, Q. Luo, et al., 2015; Javed, McNair, et al., 2016; Jingya Wang et al., 2019]
	Job Title Analysis	[H. Xu, Z. Yu, B. Guo, et al., 2018; Denghui Zhang et al., 2019; Yamashita et al., 2022]
Career Path	Job Mobility Prediction	[Ye Liu et al., 2016; L. Li et al., 2017; James et al., 2018; Y. Yang, Zhan, and Jiang, 2018; Meng et al., 2019; L. Zhang et al., 2021; Decorte et al., 2021]
	Career Path Analysis	[H. Xu, Z. Yu, J. Yang, et al., 2016]
Matching Record	Person-Job Fit	[Qin et al., 2018; Zhu et al., 2018; Shen et al., 2018; Yan et al., 2019; Bian, W. X. Zhao, et al., 2019; Y. Luo et al., 2019; Bian, X. Chen, et al., 2020]
Other		[Lacic et al., 2020]

2.2 Skill Oriented

With the advent of the era of the knowledge economy, the competition between people has become extremely fierce. Talents with employer-targeted skills will be easily hired and get attractive rewards, such as high salaries and fast promotions. However, there still exists a "Skill Gap" between employers and job seekers. The term "Skill Gap" describes the mismatch between the skills employers want and job seekers possess. This mismatch makes it difficult for job seekers to find jobs, and it also makes it difficult for employers to find suitable employees. Recently, many efforts have been made to analyze, alleviate and eliminate the "Skill Gap". Generally speaking, the first step is to detect skills automatically from resumes and job postings, and then to determine the value of skills, such as quantifying the skill relevance and measuring the skill popularity. After that, the next step is to provide career advice in terms of skills that the candidate should acquire and provide resume service by advising which skills should be highlighted in the resume. Therefore, we further categorize related works according to different steps in the process of solving the "Skill Gap" problem:

- **Skill extraction** [Kivimäki et al., 2013; M. Zhao et al., 2015; Javed, Hoang, et al., 2017; Sayfullina, Malmi, and Kannala, 2018; Khaouja et al., 2019; Gugnani and Misra, 2020; Wings, Nanda, and Adebayo, 2021].
- **Skill analysis** [W. Zhou et al., 2016; T. Xu et al., 2018; Börner et al., 2018].
- **Skill representation learning** [Dave et al., 2018; Nigam et al., 2020; Wings, Nanda, and Adebayo, 2021].

2.2.1 Skill Extraction

Skill extraction is a process of extracting standard professional skills terms from a given text. The general idea of this type of work is to match the text (i.e., job postings or resumes) with some off-the-shelf databases (e.g., predefined skill sets and Wikipedia) to extract skills from it. For example, Elisit [Kivimäki et al., 2013] first calculates similarities between an input document and Wikipedia pages and then uses a biased, hub-avoiding Spreading Activation algorithm on the Wikipedia hyperlink graph to associate the Wikipedia page with LinkedIn skills [Bastian et al., 2014]. Then the skills relevant to the input text are those associated with similar Wikipedia pages. CareerBuilder[1] has built an in-house system, SKILL [M. Zhao et al., 2015; Javed, Hoang, et al., 2017], for professional skill identification and normalization. It extracts skill keywords from job postings and resumes and then uses the Wikipedia API[2] for deduplication and normalization. It also uses Word2Vec [Mikolov, K. Chen, et al., 2013] to disambiguate skills. TaxoSoft [Khaouja et al., 2019] uses DBpedia[3] and Word2Vec to find terms related to soft skills from job postings and then uses social network analysis to build a term hierarchy for generating the soft skill taxonomy. [Gugnani and Misra, 2020] first leverages three parallel modules, Named Entity Recognition (NER), Part Of Speech (POS) Tagger, and Skill Dictionary, to identify skill-terms from candidate profiles and job postings. Each module separately extracts a set of "candidate" terms along with a module-specific score for each term, and the union of these sets forms a probable skill set. The Word2Vec module then computes the cosine similarity between each term in this set and all skill phrases found in the Skill Dictionary. By combining four module-specific scores, each identified term is assigned a score indicating how likely it is to be a skill.

Unlike the approaches mentioned above that tackle the skill extraction task by finding skill-related terms from free text, [Sayfullina, Malmi, and Kannala, 2018] treats the soft skill matching problem as a binary classification, i.e., predicting whether a potential soft skill describes a candidate rather than the company, using the context in which the skill occurs. In addition, [Wings, Nanda, and Adebayo, 2021] formulates the skill extraction problem as a multi-class classification problem with three classes, i.e., *not skill*, *soft skill*, and *hard skill*, and the classifier input is representations of pre-annotated skill phrases obtained through a language model (e.g., Word2Vec [Mikolov, K. Chen, et al., 2013], FastText [Bojanowski et al., 2017], and BERT [Devlin et al., 2018]).

2.2.2 Skill Analysis

After extracting skills, some works analyze skills from different aspects, such as relevance and popularity. For example, as a follow-up work of SKILL [M. Zhao et al., 2015], [W. Zhou et al., 2016] quantifies the relevance between skills and job titles. It uses Term Frequency Inverse Document Frequency (TF-IDF) based scores to calculate base relevant scores of skills for job titles. Then these base scores are modified with different factors, including the skill uniqueness and the global and local frequency of the skill. [T. Xu et al., 2018] proposes a skill popularity based topic model for modeling the popularity of job skills based on the analysis of large-scale recruitment data via estimating the probability of skill in the related topic. [Börner et al., 2018] analyzes skill discrepancies between research, education,

[1] https://www.careerbuilder.com/
[2] en.wikipedia.org/w/api.php
[3] https://www.dbpedia.org

and job, as well as proposes visual and computational models to help decision-makers understand the evolving structure of skills so they can develop educational programs that meet workforce needs.

2.2.3 Skill Representation Learning

In addition to directly analyzing the relevance or popularity of skills, learning appropriate skill representations is another meaningful task. The learned representations can serve other downstream tasks, such as characterizing talents according to the skills they possess and helping with job classification according to the skills those jobs require. A typical work is [Dave et al., 2018], which first builds a job transition graph, job skill graph, and skill co-occurrence graph from the working history section of the resume. Then, job title and skill representations are jointly learned from these three graphs.

Similar to methods that adapt Transformer-based language models, such as BERT [Devlin et al., 2018], to their specific domains (e.g., BioBERT [J. Lee et al., 2020] for the medical domain or sciBert [Beltagy, Lo, and Cohan, 2019] for scientific literature), in [Nigam et al., 2020], the authors propose SkillBERT, a BERT model trained from scratch on manually annotated skills datasets that can be used for skill classification. In [Wings, Nanda, and Adebayo, 2021], besides BERT, other language models such as Word2Vec and BERT are also used for modeling skills.

2.2.4 Available Skill Datasets and Tools

To facilitate domain research, we list in Table 2.2 the datasets and online tools we know of that are relevant to the skill.

TABLE 2.2: A summary of available skill datasets and tools.

Source	Description
EMSI skills API	• 32,000+ skills gathered from online job postings, profiles, and resumes. • skill names, IDs, and types (Specialized skills, common skills, certifications) • categories and subcategories for each skill
Fine-tuned BERT	• model proposed in [Wings, Nanda, and Adebayo, 2021]
Soft skill dataset	• created by [Sayfullina, Malmi, and Kannala, 2018] • soft skills and the context in which they occur are annotated with 1 (i.e., if it is a soft skill of the candidate) and 0 (i.e., otherwise)

Summary This section presents a non-exhaustive view of works using AI and its advances in Natural Language Processing (NLP) for **SKILL**-oriented tasks. We underline the following points from this analysis of existing works:

- Recent Deep Learning (DL)-based language models are very promising for improving different skill-oriented tasks, and the previous works are good proofs of the interest in deep models for the recruitment domain.

- Nonetheless, most of the approaches described in this section rely heavily on manually constructing annotated datasets, which are often unavailable to the public. Table 2.7 provides a summary of the data and tools used in these works. This emphasizes both Challenge 1 and Challenge 2 because in order to account for new skills, the dataset has to be incremented and the model retrained. Moreover, most of the proposed models are specific to English only.
- At last, the study of these related works also shows that the structured information between jobs and skills (skill-job graphs) or between skills (skill taxonomy or skill co-occurrence graphs) is rarely exploited.

2.3 Job Title Oriented

The job title usually implies the responsibility and level of the job position. Although composed of only a few words, it is the first part to attract people. For example, when job seekers browse recruitment pages, they will be attracted by the job titles first, and recruiters will first check the work experience section. Previous studies are focusing on:

- **Job title representation learning** [Dave et al., 2018; Junhua Liu et al., 2019; M. Liu et al., 2019; Denghui Zhang et al., 2019; Decorte et al., 2021; L. Zhang et al., 2021; Yamashita et al., 2022].
- **Job title classification** [Javed, Q. Luo, et al., 2015; Javed, McNair, et al., 2016; Jingya Wang et al., 2019].
- **Job title analysis** [H. Xu, Z. Yu, B. Guo, et al., 2018; Denghui Zhang et al., 2019; Yamashita et al., 2022].

2.3.1 Job Title Representation Learning

Like skill representation learning, job representation learning is also an important task that benefits various downstream tasks, such as facilitating job classification and improving talent profile modeling. [Dave et al., 2018] jointly learns skill representations and job title representations from three graphs built from resumes. This approach is experimentally tested on an unavailable dataset from CareerBuilder. Title2vec [Junhua Liu et al., 2019] is a contextual job title vector representation obtained by fine-tuning from the pre-trained model, ELMo (Embedding from Language Model) [Peters et al., 2018] on the IPOD dataset. This dataset consists of 192k job titles belonging to 56k LinkedIn users, and each word of these job titles is manually labeled with a Named Entity (NE) tag indicating its level of seniority, field of work, and location. More descriptions can be found in Appendix B.1.3. The obtained representations can then be used for Named Entity Recognition (NER) task. Unlike the typical NER task that uses general tags, such as **LOC**ation, **PER**son, and **ORG**anization. [Junhua Liu et al., 2019] proposes domain-specific NE tags to denote the properties of occupations, such as **RES**ponsibility, **FUN**ction and **LOC**ation. JobBERT [Decorte et al., 2021] is designed for the job title normalization task, which is based on the premise that skills are the basic elements of defining jobs. It first applies a simple distant supervision rule to create a noisy training set of job titles with associated

skills through literal string matches in job descriptions with an extensive proprietary reference skill list. Then JobBERT uses BERT (Bidirectional Encoder Representations from Transformer) [Devlin et al., 2018] as the backbone to encode job titles and applies the loss of Skip-Gram [Mikolov, Sutskever, et al., 2013] with negative sampling to train the model.[4]

In addition to the above works that take learning job title representation as the target task, some works consider learning job title representation as a step in the process of achieving the final task. For example, Job2Vec [Denghui Zhang et al., 2019] learns collective multi-view job title representations for job title benchmarking, aiming to match job titles with similar expertise levels across various companies. Similarly, [Yamashita et al., 2022] also learns multi-aspect job title representations in order to map user-created job titles to predefined standard job titles (Section 2.3.3). [L. Zhang et al., 2021] learn company and position representations from a heterogeneous company-position graph by applying an attentive heterogeneous graph embedding model. The learned representations are then used for the task of job mobility prediction (Section 2.4.1). As with most methods mentioned, the last three methods are experimentally constructed and evaluated on private datasets, for which the datasets are unavailable.

2.3.2 Job Title Classification

In the job recruitment domain, categorizing large datasets consisting of job postings and resumes into pre-defined or custom occupation categories is important for many downstream applications, such as matching jobs with resumes, searching for positions, recommending jobs, and labor market analysis [Jingya Wang et al., 2019]. Carotene [Javed, Q. Luo, et al., 2015; Javed, McNair, et al., 2016], proposed by the CareerBuilder team, is a ML-based hierarchical, two-stage, semi-supervised, proximity-based, multi-class, and multi-label job title classification system, which uses the pre-existing O*NET-SOC 2010[5] hierarchy as target classes. DeepCarotene [Jingya Wang et al., 2019] is a more advanced model of Carotene. It uses an end-to-end multi-stream Convolutional Neural Network (CNN) to learn semantic features on both character and word levels. More specifically, DeepCarotene considers both job title and job description as input and creates a hierarchical taxonomy by adding additional subcategories to the 23 Major Groups of SOC 2018[6] system (Appendix B.2.2). Because it is difficult to obtain large-scale job title labels and ensure the correctness of liberalization, DeepCarotene first uses a weakly supervised model to obtain large-scale training data with noisy labels from Carotene. Then, it jointly trains the character stream for local matching (i.e., based on spelling and pattern) and the word stream for semantic matching (i.e., based on distributed representation). A multi-task loss function is used to predict (i) the SOC 2018 Major Groups and (ii) the overall job title (i.e., including SOC Major Groups and the subcategories).

Most of the works mentioned in the previous section can also be used for job title classification. Once again, for all this work, a big part is related to the building of a good and sufficient large annotated dataset to train the corresponding proposed model. Most of the works mentioned in the previous section can also be used for job title classification. Again, for all of these works, a large part

[4] The dataset use in the JobBERT model that consists of a list of Job tiles, each tagged with an ESCO occupation has been made available here: https://github.com/jensjorisdecorte/JobBERT-evaluation-dataset

[5] https://www.bls.gov/soc/2018/major_groups.htm

[6] https://www.bls.gov/soc/2018/home.htm

has to do with constructing well-annotated datasets large enough to train the corresponding proposed models, and most of these datasets, with the exception of the one used in JobBERT, are unavailable, thus making it difficult to reproduce these methods.

2.3.3 Job Title Analysis

Since the job title plays a vital role in both the resume and job posting, extensive job title analysis is performed to benefit various downstream tasks. For example, [H. Xu, Z. Yu, B. Guo, et al., 2018] adopts a Gaussian Bayesian network to model the joint distribution of job title ranks and the difficulty of promotion from one job title to another in a career trajectory, thereby extracting the job title hierarchy from the career trajectory data. Job2Vec [Denghui Zhang et al., 2019] is a collective multi-view representation learning method for job title benchmarking, aiming to match job titles with similar expertise levels across various companies. Specifically, the authors first construct a job graph, where the nodes represent the job titles affiliated with specific companies and the edges represent the correlations between job titles. Along this line, they reformulate job title benchmarking as the link prediction task over the job graph. Similar to Job2Vec, [Yamashita et al., 2022] proposes a learning method for multi-aspect representations (i.e., syntactic, semantic, and topology) of job titles for the job title mapping task, which aims to map user-created job titles to predefined standard job titles.

2.3.4 Available Job/Job Title Datasets and Tools

To facilitate domain research, we list in Table 2.3 the datasets and online tools we know of that are relevant to the job/job title.

TABLE 2.3: A summary of available job/job title datasets and tools.

Source	Description
Job Salary Prediction dataset	• job ads • IDs, job titles/descriptions, location, company, category, salary
Real/Fake Job Posting Prediction	• 18,000 job ads, about 800 of which are fake • IDs, job titles/descriptions/requirements, location, salary, company
IPOD	• created by [Junhua Liu et al., 2019] • 475,085 job titles crawled from Linkedin, with NE tags prefixed
JobBERT evaluation dataset	• created by [Decorte et al., 2021] • 30,926 job titles, each tagged with an ESCO occupation (B.2.4)
EMSCAD	• 17,880 job ads • contains 17,014 legitimate and 866 fraudulent job ads • IDs, job titles/descriptions/requirements, location, salary, company
O*NET-SOC AutoCoder	• a tool that provides an occupational code (i.e., SOC2018, O*NET2019, and OES2020) for a job, resume and, UI claim

Summary This section presents a non-exhaustive view of works using AI and its advances in Machine Learning (ML) and Deep Learning (DL) for **JOB TITLE** tasks. We underline the following points from this analysis of existing works:

- Using language models like BERT and ELMo is a straightforward way to model job title representations [Junhua Liu et al., 2019; Decorte et al., 2021]. Another common direction is to explore other hidden relations between job titles via graph topology, i.e., job transition graph, to improve the learned representations [Dave et al., 2018; Denghui Zhang et al., 2019; L. Zhang et al., 2021].
- Nevertheless, graph-based methods have attempted to consider more transition attributes, such as transition duration and number [Denghui Zhang et al., 2019; L. Zhang et al., 2021], but most of them ignore the most basic semantic information or learn semantic and topological information separately.
- Indeed, the learned job title representation can also be used for classification and analysis tasks. However, the annotated and open-source datasets are difficult to access due to privacy and industry competition reasons (i.e., Characteristic 1).

2.4 Career Path Oriented

A career path/trajectory/history is a record of the jobs or professions that a person has engaged in his/her working years. Indeed, the wealth of information on talent career trajectories provides a new paradigm for recruitment analysis. Studies on the career path can provide potential benefits for employees and employers. For example, employees can get information about their current career stages and the full picture of their career paths to plan their next career move better, and employers can learn about the career development of employees and decide the best time to promote employees or increase salaries. Consequently, career path modeling is a research topic with high potential and has many real-world applications, such as job mobility prediction and career advice.

- **Job mobility prediction** [Ye Liu et al., 2016; L. Li et al., 2017; James et al., 2018; Y. Yang, Zhan, and Jiang, 2018; Meng et al., 2019; L. Zhang et al., 2021].
- **Career path analysis** [H. Xu, Z. Yu, J. Yang, et al., 2016].

2.4.1 Job Mobility Prediction

Job mobility prediction is the task of predicting the next career move of talents based on their career paths and other available data. It is an emerging research topic that can benefit both organizations and talents in various ways, such as job recommendation, talent recruitment, and career planning. Recent works have proposed different structures for various prediction scenarios, such as predicting whether an employee will leave, the next position he will go to, and how long he will stay in the new job. For example, [Ye Liu et al., 2016] presents a multi-view multi-task learning approach for career path prediction. This model fuses information from multiple social networks (e.g., Twitter, Facebook, and

LinkedIn) to comprehensively describe talents and characterize the progressive properties of their career paths. NEMO [L. Li et al., 2017] is a contextual Long Short Term Memory (LSTM) model [Hochreiter and Schmidhuber, 1997], which integrates the profile context (e.g., skills, education, and location) and career path dynamics to predict the next career move of the employee (which company with what job title). In [James et al., 2018], they predict which scientists will move in the next year and which institution will be selected based on the career quality, scientific environment, and the structure of the scientific collaboration network of scientists. [Y. Yang, Zhan, and Jiang, 2018] investigates several ML-models to help improve the talent demission prediction. The talent demission prediction problem is formalized as a binary classification problem, and a causal structure learning based method is proposed to calculate the causal relationships between different features (e.g., age, department, and education) and labels, which can help the management to make decisions with the fewest considerations. [Meng et al., 2019] integrates three levels of knowledge: *personal-specific*, *company-specific*, and *position-specific* features to predict the next employer of an individual and how long he/she will stay in the new position. Specifically, the model applies a local attention mechanism and a global attention mechanism, respectively, to follow two LSTMs layers to learn the influences of internal and external job mobility on the next job decision of individuals. [L. Zhang et al., 2021] first applies an attentive heterogeneous graph embedding model on a heterogeneous company-position graph constructed from the massive career trajectory data to learn company and position representations. A dual-GRU model is then used to capture sequential interaction information between companies and positions, and the final output is used for prediction.

2.4.2 Career Path Analysis

In addition to predicting future behavior based on career paths, organizations can be categorized by analyzing the work histories of talents. In [H. Xu, Z. Yu, J. Yang, et al., 2016], they propose to detect talent circles from an organization-level job transition network, where nodes stand for organizations and edges represent the job transition among organizations. Each circle contains organizations with similar talent exchange patterns, and clusters of organizations are found by inferring the existence probabilities of edges in different circles.

2.4.3 Available Resume/Career Path Datasets and Tools

To facilitate domain research, we list in Table 2.4 the datasets and online tools we know of that are relevant to the resume/career path.

TABLE 2.4: A summary of available resume/career path datasets and tools.

Source	Description
US Resume Dataset on DataStock	• about 8M resumes extracted from Indeed.com • IDs, job titles or job profiles, location
Resume Dataset	• 2, 400+ resumes collected from livecareer.com • IDs, resume texts, categories of the job the resume was used to apply • categories include: HR, Designer, Teacher, Advocate, Healthcare ...

Summary This section presents a non-exhaustive view of works using AI and its advances in Recurrent Neural Network (RNN) for **CAREER PATH**-oriented tasks. We underline the following points from this analysis of existing works:

- Recurrent Neural Networks (RNNs) have demonstrated their superiority on sequence modeling tasks. Thus, they are widely used in job mobility prediction tasks.
- Career path records have many usages. In addition to modeling the professional profile of talents, it can also be used to support the learning of recruitment representations, such as learning job and skill representations [Dave et al., 2018; Denghui Zhang et al., 2019; L. Zhang et al., 2021].

2.5 Matching Records Oriented

A matching record is a collection of job applications $\mathcal{A} = \{(r, j, y)\}_{r \in \mathcal{R}, j \in \mathcal{J}}$, where each application contains a personal profile (usually expressed by a resume r) and a job posting j. Here, $\mathcal{R}$ and $\mathcal{J}$ denote resume set and job postings set, respectively. Correspondingly, the matching label y indicates whether the candidate with the resume r got the job j, i.e., $y = 1$ means he/she succeeded, while $y = 0$ means he/she failed. An example of matching record is given in Figure 2.1, i.e., the part (iv).

2.5.1 Person-Job Fit

The study of measuring the matching degree between the abilities of a person and the demands of a job, namely Person-Job Fit [Edwards, 1991; Sekiguchi, 2004], has become one of the most compelling topics in the recruitment field. The Person-Job Fit is defined as estimating the matching degree y for the resume-job pair (r, j) in an application.

FIGURE 2.2: Illustration of *Person-Job Fit* task.

The early research work on Person-Job Fit can be traced back to [Malinowski et al., 2006] in 2006, where authors used person and job information to build a bilateral person-job recommendation system in order to find a good match between a people and a job. In recent studies [Qin et al., 2018; Zhu et al., 2018], a typical approach is to consider the Person-Job Fit task as a supervised text matching problem. Consider a set of labeled data (i.e., person-job matching records) as training data, the purpose of the Person-Job Fit task is to train a matching label (i.e., matching or not matching) classifier based on the text content of job postings and candidate resumes. Several works have proposed diverse end-to-end models that use different architectures to represent documents (i.e., job

postings and resumes) and then concatenate the learned representations as input to a classifier with binary cross-entropy loss. For example, [Qin et al., 2018] leverages a Bi-directional LSTM (BiLSTM) to generate word-level semantic representations for job requirements and experiences. Then, four hierarchical ability-aware attention strategies are designed to measure the different importance of job requirements to the job semantic representation and measure the different contributions of work experiences to a specific ability requirement. Finally, the jointly learned representations of job postings and resumes are fed into a classifier to evaluate their matching degree. [Bian, W. X. Zhao, et al., 2019] employs a hierarchical attention-based BiGRU to model both sentences and documents (i.e., job postings and resumes). Then, the global semantic interaction between each sentence in the job posting and each sentence in the resume is modeled. Finally, job posting, resume, and the global match representations are concatenated as input to a matching label predictor. [Y. Luo et al., 2019] studies the effectiveness of adversarial training for the Person-Job Fit problem. The proposed model uses an adversarial learning based framework [Makhzani et al., 2015] to learn more expressive representations for resumes and job postings, thus leading to better representations for Person-Job Fit. [Bian, X. Chen, et al., 2020] proposes a multi-view co-teaching network to integrate a *text-based matching model* and a *relation-based matching model* to also take into account the implicit relational information between candidates (their resumes) and job postings. For that, the authors build a job-resume relation graph using category labels of job postings (i.e., jobs of the same high-level category are linked in the graph) and shared keywords between resumes and job postings. Also, some works use distance-based loss instead of cross-entropy loss, so the Person-Job Fit problem is no longer treated as a classification problem with fixed classes. For example, [Zhu et al., 2018] proposes a CNN based model to jointly learn representations of job postings and resumes. Then, the fitness between the person and the job is measured by the distance between their latent representations. [Shen et al., 2018] develops a latent variable model to model job descriptions and candidate resumes jointly for Person-Job Fit and interview assessment. In addition to the often-used personal information and job requirements, interview records also contain a lot of useful information for fitness matching. Some works consider the interview records in the representation modeling. For example, [Yan et al., 2019] proposes an end-to-end learnable NN to learn the preferences of recruiters and job seekers from their previous interview and application histories, respectively, to improve job-resume matching.

We summarize the different representation learning structures and matching degree measurement frameworks used in the works related to Person-Job Fit in Table 2.5. The vast majority of works rely on deep networks to learn the job and resume representations, and among these works, some further consider other information beyond job-resume pairs to help measure suitability.

TABLE 2.5: A summary of works related to Person-Job Fit, including the architecture of representation learning and the framework of matching degree measurement.

Paper	Representation Learning Architecture	Matching Degree Measurement Framework
[Qin et al., 2018]	• BiLSTM: requirement/experience representation • Hierarchical ability-aware attention mechanism: job/resume representation	Concatenation of job and resume + binary cross entropy loss
[Zhu et al., 2018]	• CNN: requirement/experience representation • Max/mean-pooling: job/resume representation	Minimize/Maximize the distance of positive/negative job-resume pairs
[Shen et al., 2018]	• Topic distribution: job/resume representation	Minimize the distance between job and resume probability distributions
[Yan et al., 2019]	• GRU-based profiling memory model: preference-aware job and resume representation • The model memorizes the interviewed candidates for the job and memorizes the job application history for the resume	Minimize/Maximize the distance of positive/negative job-resume pairs
[Bian, W. X. Zhao, et al., 2019]	• Hierarchical attention-based BiGRU: job/resume sentence representation • Hierarchical attention-based BiGRU: job/resume representation • Compute the global semantic interactions between every sentence in the job and every sentence in the resume	Concatenation of job and resume + binary cross entropy loss
[Y. Luo et al., 2019]	• ELMo: job/resume word/phrase representation • Hierarchical attention BiLSTM: experience representation • Attention scheme: skill representation • CNN+adversarial learning mechanism: job representation	Concatenation of job and resume + binary cross entropy loss
[Bian, X. Chen, et al., 2020]	• BERT+Transformer: job/resume sentence representation+job/resume representation • Relational Graph Convolutional Network on a job-resume relation graph: job/resume representation	Concatenation of job and resume + binary cross entropy loss + Multi-view co-teaching network

2.5.2 Available Person-Job Matching Datasets and Tools

To facilitate domain research, we list in Table 2.6 the datasets and online tools we know of that are relevant to the person-job matching.

TABLE 2.6: A summary of available person-job matching datasets and tools.

Source	Description
Matching Records	• used in [Yan et al., 2019]
Matching Records	• used in [Bian, W. X. Zhao, et al., 2019]
Matching Records	• used in [Bian, X. Chen, et al., 2020]

Summary This section presents a non-exhaustive view of works using AI and its advances in Deep Learning (DL) for **PERSON-JOB FIT** tasks. We underline the following points from this analysis of existing works:

- Most of the models described in this section typically deal with the Person-Job Fit task as a supervised binary text matching problem by training a matching label classifier based on a set of labeled person-job matching records. However, such datasets are not easy to collect due to the privacy reason (i.e., Characteristic 1).
- Various deep architectures, such as Convolutional Neural Network (CNN), GRU, and LSTM have been proposed with the aim of better modeling jobs and persons, or some works suggest using pre-trained language models such as ELMo and BERT to obtain their representations.
- Finally, the study of these related works also shows that the information used to construct representations, whether jobs or talents, is usually based on textual semantics, but the relationships between them are rarely exploited, except for [Bian, X. Chen, et al., 2020] considering a job-resume graph.

2.6 Others

In addition to using the above data types, several works have explored interaction records (e.g., job seekers *click*, *bookmark*, and *apply* jobs) for job recommendation tasks. For example, [Lacic et al., 2020] encodes interactions between job seekers and jobs using different autoencoder architectures, and then recommends the next job posting in a K-nearest-neighbor manner.

TABLE 2.7: A summary of data used in recruitment-related works. *NS* means that the authors did not provide a specific name for the data source. **no** indicates that the authors did not give or we did not find a link to the same dataset used in the paper.

	Paper	Data & Source & Availability	Model	Graph Used	Add. Tool
Skill Extraction	Elisit [Kivimäki et al., 2013]	• skill: LinkedIn: **no**	ML	Wiki hyperlink graph	Wikipedia
	SKILL [M. Zhao et al., 2015]	• job ad: CareerBuilder: **no** • CV: CareerBuilder: **no**	ML+DL	**no**	Wikipedia, Word2Vec
	[Sayfullina, Malmi, and Kannala, 2018]	• job ad: LinkedIn: **yes** • CV: indeed: **yes** • skill: NB: **yes**	ML+DL	**no**	-
	TaxoSoft [Khaouja et al., 2019]	• job ad: CareerBuilder, Apec, Keljob, Rekrute...: **no**	ML+DL	**no**	DBpedia, Word2Vec
	[Gugnani and Misra, 2020]	• job ad: NS: **no** • CV: borrowed: **no** • skill: predefined: **no**	ML+DL	**no**	Word2Vec
	[Wings, Nanda, and Adebayo, 2021]	• job ad: EMSCAD: **yes** • hard skill: EMSI: **yes** • soft skill: : **yes**	ML+DL	**no**	TaxoSoft

Continued on next page

Table 2.7 – continued from previous page

	Paper	Data & Source & Availability	Model	Graph Used	Add. Tool
Skill Analysis/ Representation Learning	[W. Zhou et al., 2016]	• job ad: CareerBuilder: **no**	ML	**no**	SKILL, Carotene
	[T. Xu et al., 2018]	• job ad: not specify: **no** • skill: CSDN 2016: **no**	ML	**no**	-
	[Dave et al., 2018]	• CVs: CareerBuilder: **no**	ML	job transition, job-skill, skill-skill graphs	SKILL, Carotene
	SkillBERT [Nigam et al., 2020]	• skills: predefined: **yes**	ML+DL	**no**	BERT
Job Title Representation Learning	[Dave et al., 2018]	• CV: CareerBuilder: **no**	ML	job transition, job-skill, skill-skill graphs	SKILL, Carotene
	Title2vec [Junhua Liu et al., 2019]	• job title: LinkedIn: **yes**	ML+DL	**no**	ELMo
	Job2Vec [Denghui Zhang et al., 2019]	• career-path: NS: link wrong	ML	job transition graph	-
	JobBERT [Decorte et al., 2021]	• job title: MYFutureJobs: **yes**	ML+DL	**no**	BERT
	[L. Zhang et al., 2021]	• career-path: LinkedIn: **no**	ML+DL	company-position graph	-
	JAMES [Yamashita et al., 2022]	• resumes: FutureFit AI: **no**	ML+DL	job transition graph	BERT
Job Title Classification/Analysis	Carotene [Javed, Q. Luo, et al., 2015]	• job title: CareerBuilder: **no**	ML	**no**	-
	DeepCarotene [Jingya Wang et al., 2019]	• job title: CareerBuilder: **no**	ML+DL	**no**	Carotene
	[H. Xu, Z. Yu, B. Guo, et al., 2018]	• career-path: NS: **no**	ML	job-title hierarchy graph	-
	Job2Vec [Denghui Zhang et al., 2019]	• career-path: NS: link wrong	ML	job transition graph	-
	JAMES [Yamashita et al., 2022]	• resumes: FutureFit AI: **no**	ML+DL	job transition graph	BERT
Job Mobility Prediction/ Career Path Analysis	[Ye Liu et al., 2016]	• career-path: About.me: **no**	ML	**no**	-
	NEMO [L. Li et al., 2017]	• career-path: LinkedIn: **no**	ML+DL	**no**	-
	[James et al., 2018]	• career-path: APS: **no**	ML	**no**	-
	[Y. Yang, Zhan, and Jiang, 2018]	• career-path: IBM: **no**	ML	**no**	-
	[Meng et al., 2019]	• career-path: NS: **no**	ML+DL	**no**	-
	[L. Zhang et al., 2021]	• career-path: LinkedIn: **no**	ML+DL	company-position graph	-
	[H. Xu, Z. Yu, J. Yang, et al., 2016]	• career-path: not specify: **no**	ML	job transition graph	-
Person-Job Fit	[Qin et al., 2018]	• job-CV pair: Baidu: **no**	ML+DL	**no**	-
	[Zhu et al., 2018]	• job-CV pair: Baidu: **no**	ML+DL	**no**	-
	[Shen et al., 2018]	• job-CV pair: Baidu: **no**	ML	**no**	-
	[Yan et al., 2019]	• job-CV pair & interview records: Boss Zhipin: **yes**	ML+DL	**no**	-

Continued on next page

Table 2.7 – continued from previous page

	Paper	Data & Source & Availability	Model	Graph Used	Add. Tool
	[Bian, W. X. Zhao, et al., 2019]	• job-CV pair: Boss Zhipin: **yes**	ML+DL	**no**	-
	[Y. Luo et al., 2019]	• job-CV pair: private: **no**	ML+DL	**no**	ELMo
	[Bian, X. Chen, et al., 2020]	• job-CV pair: Boss Zhipin: **yes**	ML+DL	job-resume graph	BERT
Next-Job Prediction	[Lacic et al., 2020]	• job ad & interaction: Studo, RecSys17,CareerBuilder12: **yes**	ML+DL	**no**	-

2.7 Summary and Positioning

AI, especially the deep models, have been widely applied to handle the different recruitment tasks mentioned above and achieved encouraging performance. These works use different types of recruitment data based on different model structures and process schemes, which are summarized in detail in Table 2.7. We underline some points by analyzing the table and all the works mentioned above:

- **Terminology resource receive little attention**: As seen from the table, different tasks mainly focus on specific data types, such as career paths for job mobility prediction and job-resume pairs for Person-Job Fit, but job postings and resumes are frequently used in all tasks. In addition to these data, interaction records and interviewee records are also explored. However, as described in Section 1.2, the recruitment field also has a wealth of predefined terminology resources, especially **skill taxonomies** and **occupation taxonomies**, which are well-structured knowledge graphs to show the hierarchical relationship between standard terms. These taxonomies often contain much normative knowledge, but no work has yet made use of it, whether directly or indirectly. In such a context, we explore the utility of this data type, especially in representation learning. More specifically, we take the skill taxonomy into account to learn more informative skill representations, which will be explained in Chapter 5.

- **Graph topology information is as important for representation learning as semantic information**: Much data in the field of recruitment can be represented by graph structures, and there also exist graph-like terminology resources, as mentioned earlier. However, this important information is less studied because current works and methods focus more on semantic information. For example, different architectures have been designed to learn job and person representations for Peron-Job Fit or classification tasks. Some works have attempted to explore graph structures to assist various tasks, such as associating skills with Wikipedia pages [Kivimäki et al., 2013], matching job titles with similar expertise levels across various companies [Denghui Zhang et al., 2019], predicting job mobility [L. Zhang et al., 2021], and matching talents with jobs [Bian, X. Chen, et al., 2020]. The core of these works is the learning of related representations. However, these methods either ignore the semantic information carried by the manipulated data or learn topological or semantic information separately. In

Chapter 4, we propose a new graph-based scheme for learning job title representations from graphs by combining topological and semantic information.

- **Lack of open datasets**: Due to personal privacy and industry competition, few annotated datasets are available in the field of recruitment (Challenge 1). Table 2.7 demonstrates this argument that most works do not make public the datasets they use. This lack of publicly available annotated datasets is problematic because most state-of-the-art high-performance AI data-driven approaches are mainly based on a supervised learning paradigm, which assumes the availability of large annotated datasets, such as Person-Job Fit and job classification tasks. Therefore, this lack of annotated datasets inspires other learning schemes, such as unsupervised or semi-supervised learning. In Chapter 6, we learn job representations in an unsupervised manner and use the learned representations for next-application prediction, which is less studied in the current domain. This is another reason why we focus or research on learning representation using graphs.

Summary of Chapter 2

This chapter reviews existing AI-based works, open datasets and online tools in the recruitment field. By comparing these different works, we summarize current issues and challenges to highlight the strengths of our proposed methods and the importance of our research questions, i.e., **how to leverage implicit (i.e., interaction or transition records) or explicit (i.e., taxonomy) structured recruitment data to improve the learned recruitment-related representations**.

Chapter 3

Preliminary

This chapter, as a preparatory part, consists of two parts: (i) an introduction to graph embeddings and (ii) an introduction to recommendation models, especially sequential recommendation, both of which will be studied in our work. Throughout this chapter, we use bold uppercase characters to denote matrices (e.g., $\boldsymbol{W}$) and bold lowercase characters to denote vectors (e.g., $\boldsymbol{v}$). We employ non-bold letters (e.g., n, N) to represent scalars and use Greek letters (e.g., λ) as parameters. The cardinality of a set is denoted by the vertical bar $|\cdot|$.

3.1 Review of Graph Embedding Models

Real-world data usually comes together with graph structures, where nodes represent individuals and edges represent relations or interactions between nodes, such as social networks, citation networks, and biological networks, as shown in Figure 3.1.

(a) Social network. (b) Citation network.

FIGURE 3.1: Examples of graphs.

Similarly, in the field of recruitment, there are various graph structures such as job transition graphs, job company affiliation graphs, and predefined occupational taxonomies. Inspired by the success of graph embedding methods in other applications, we therefore try to explore graph embedding methods in the recruitment domain, especially in representation learning. This section first formally introduces the basic concepts of graph embedding. For facilitating illustration, we list some important mathematical notations used throughout this section in Table 3.1, unless particularly specified. Then we introduce some representative graph embedding methods for both *homogeneous* and *heterogeneous* input graphs, which will be used as baselines for *job title representation learning* task (Chapter 4) and *skill representation learning* task (Chapter 5).

TABLE 3.1: Mathematical notations used in *Review of Graph Embedding Models*.

Notation	Description
G	graph/network
$\mathcal{V}, \mathcal{E}$	node set, edge (link) set
$\boldsymbol{X}$	feature matrix
$\boldsymbol{A}$	adjacency matrix
v_x, e_{xy}	x-th node, edge between node pair (v_x, v_y)
$\texttt{weight}_{xy}$	weight of edge e_{xy}
$\texttt{deg}_{out}(v)$	out-degree of node v
$\texttt{deg}_{in}(v)$	in-degree of node v
$\mathcal{T}_V, \mathcal{T}_E$	node type set and edge type set
$\psi(v)$	node type mapping function
$\varphi(e)$	edge type mapping function
$v_{w(i)}$	node at i-th position in the random walk
c	window size of context
$\Phi(\cdot)$	embedding mapping function
$\mathcal{N}(v)$	neighbors of node v

3.1.1 Basic Concepts

Traditionally, a basic graph/network[1] is represented as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$ is the set of nodes (vertices/entities),[2] and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges (relation/links),[3] each edge is represented by a node pair $e_{xy} = (v_x, v_y)$, $x, y \in [1, |\mathcal{V}|]$. In some graphs, nodes are equipped with node features, $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{|\mathcal{V}|}\}, \boldsymbol{x}_i \in \mathbb{R}^F$, where F is the number of features. A graph can be represented by an adjacency matrix $\boldsymbol{A}$ of dimension $|\mathcal{V}| \times |\mathcal{V}|$, and the elements of the matrix are $\{a_{xy}\}$, indicating whether the nodes v_x and v_y are connected in the graph, i.e., $a_{xy} = 1$ if $e_{xy} \in \mathcal{E}$, 0 otherwise. If $a_{xy} \neq a_{yx}$, G is a *directed* graph, or else it is an *undirected* graph. If a_{xy} is weighted by $\texttt{weight}_{xy} \in \boldsymbol{W}$, $\boldsymbol{W} \in \mathbb{R}^+$, $G = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$ is a *weighted* graph, otherwise it is an *unweighted* graph. Nodes and edges can optionally have a type, which is represented as $\mathcal{T}_V$ and $\mathcal{T}_E$. A graph with a single type of node and a single type of edge, i.e., $|\mathcal{T}_V| = 1$ and $|\mathcal{T}_E| = 1$, is called *homogeneous*. On the other hand, a graph with two or more types of nodes and/or two or more types of edges, i.e., $|\mathcal{T}_V| + |\mathcal{T}_E| > 2$, is called *heterogeneous*. We formally define these two types of graphs as follows:

Definition 3.1.1 Homogeneous Graph (HomG): *Homogeneous graph is defined as $G = (\mathcal{V}, \mathcal{E}, \mathcal{T}_V, \mathcal{T}_E)$, in which each node $v \in \mathcal{V}$ and each edge $e \in \mathcal{E}$ are associated with their type mappings, i.e., $\psi(v) \colon \mathcal{V} \to \mathcal{T}_V$ and $\varphi(e) \colon \mathcal{E} \to \mathcal{T}_E$. $\mathcal{T}_V$ and $\mathcal{T}_E$ represent the node type and edge type sets, respectively. In homogeneous graphs, nodes and edges have a single type, i.e., $|\mathcal{T}_V| = 1$ and $|\mathcal{T}_E| = 1$.*

Example 3.1.1 : *An example of HomG is a job transition network, as shown in Figure 3.2, where nodes represent "job positions" (job titles) and edges represent "career moves", where there is only one type of*

[1] In this chapter, we use *graph* and *network* interchangeably.

[2] In this chapter, we use *node*, *entity* and *vertex* interchangeably.

[3] In this chapter, we use *edge*, *link* and *relation* interchangeably.

node and edge.

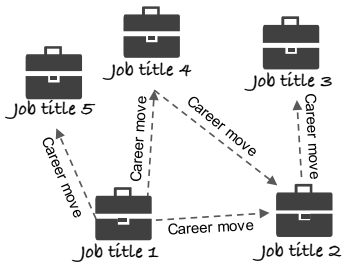


FIGURE 3.2: Example of HomG: nodes represent "job titles" and edges represent "career moves".

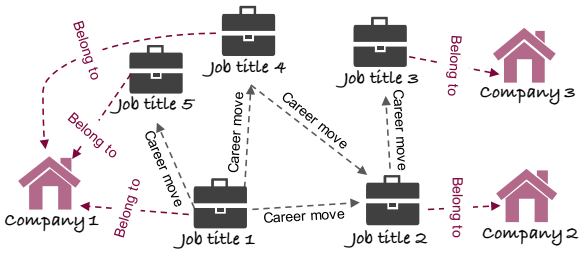


FIGURE 3.3: Example of HetG: with two node types, i.e., "job titles" and "company names", and two edge types, i.e., "career moves" and "belong to".

Definition 3.1.2 Heterogeneous Graph (HetG): *Heterogeneous graph is defined as* $G = (\mathcal{V}, \mathcal{E}, \mathcal{T}_V, \mathcal{T}_E)$ *in which there are multiple types of nodes/edges, i.e.,* $|\mathcal{T}_V| + |\mathcal{T}_E| > 2$.

Example 3.1.2 : *An example of HetG is a job transition graph with different types of nodes (i.e., "company names" and "job positions"), as shown in Figure 3.3. The edge between the two "job position" nodes has the type of "career move", and the edge between a "job position" and a "company name" node is the "belong to" type.*

Other important concepts are the graph adjacency matrix and the graph Laplacian, which we define below.

Definition 3.1.3 Adjacency Matrix: *The adjacency matrix of the graph* $G = (\mathcal{V}, \mathcal{E})$ *is defined as* $\boldsymbol{A} \subseteq \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ *(unweighted) or* $\boldsymbol{A} \subseteq \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ *(weighted), where the entry* a_{xy} *is non-zero if and only if there is an edge* $e_{xy} \in \mathcal{E}$ *between* $v_x, v_y \in \mathcal{V}$*, otherwise* $a_{xy} = 0$

Definition 3.1.4 Graph Laplacian: *The graph Laplacian is defined as* $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ *(or* $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$*), where* $\boldsymbol{D} \subseteq \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ *is the diagonal degree matrix and* $\boldsymbol{I}$ *is the identity. Through spectral decomposition,* $\boldsymbol{L}$ *can be written by* $\boldsymbol{L} = \boldsymbol{U}diag(\boldsymbol{\lambda})\boldsymbol{U}^T$*, where each column of* $\boldsymbol{U} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ *is an eigenvector of* $\boldsymbol{L}$*,* $\boldsymbol{\lambda} \in \mathbb{R}^{|\mathcal{V}|}$ *gathers the eigenvalues of* $\boldsymbol{L}$ *and* $diag(\cdot)$ *function creates a diagonal matrix whose diagonal elements are from a given vector.*

3.1.2 Graph Embedding

Graph embedding consists in learning low-dimensional feature representations of nodes, edges, or entire graphs. The principle is that the similarity in the embedding space reflects a certain proximity relationship in the graph, which is usually quantified by the following proximity measures [Cai, Zheng, and Chang, 2018; Daokun Zhang et al., 2018]:

Definition 3.1.5 First-Order Proximity: *The first-order proximity applies only to local pairwise similarities between two nodes directly connected by an edge. It is usually defined as the edge weight. The larger the edge weight between two nodes, the larger the first-order proximity between them.*

Definition 3.1.6 Second-Order Proximity: *The second-order proximity measures the similarity of the neighborhood structures of two nodes. The more similar the neighborhoods of the two nodes are, the larger the second-order proximity value between them.*

Definition 3.1.7 High-Order Proximity: *The high-order proximity measures the relationship between two nodes that are not directly adjacent. For example, the third-order proximity between two nodes measures the similarity between the neighborhood of their neighborhoods.*

The output of graph embedding is a set of low-dimensional vectors that are learned by preserving one or more of the proximity mentioned above. These vectors can be divided into four types [Cai, Zheng, and Chang, 2018]: (i) node embedding, (ii) edge embedding, (iii) hybrid embedding, and (iv) whole graph embedding. Different types of embeddings benefit different downstream tasks.

- **Node embedding** is to learn node representation in a low-dimensional space. It is the most common output of graph embeddings and can benefit various node-related graph analyses, such as node clustering and classification. In this thesis, we focus on learning node embeddings.
- **Edge embedding** is to learn edge representation in a low-dimensional space. Besides direct learning, edge representations can also be represented by a pair of node representations. It benefits edge-related graph analysis, such as link prediction and knowledge graph relation prediction.
- **Hybrid embedding** is to learn the combined representation of different types of graph components, such as arbitrary combinations of nodes and edges, as well as communities. This combined representation can be learned directly or derived by aggregating individual node and edge embeddings.
- **Whole graph embedding** is to learn the representation of the whole graph. It facilitates graph classification.

Based on the definition of different proximities and different embeddings, we formally define the graph embedding as follows:

Definition 3.1.8 Graph Embedding: *Given an input graph G (HomG or HetG) and its adjacency matrix $\boldsymbol{A}$, graph embedding consists in learning how to convert G into a d-dimensional space ($d \ll |\mathcal{V}|$) while preserving the properties of the graph. Graph properties can be quantified using proximity measures such as first- and high-order proximity. The graph is ultimately embedded as a d-dimensional vector (i.e., for the whole graph) or a set of d-dimensional vectors, each representing a graph component (e.g., node and edge).*

The resulting representations, especially node representations, can serve as precursors for numerous downstream machine learning and optimization tasks, such as link prediction [Liben-Nowell and Kleinberg, 2007; Lü and T. Zhou, 2011], node classification [Bhagat, Cormode, and Muthukrishnan,

2011], community detection [Cavallari et al., 2017; X. Wang, P. Cui, et al., 2017], recommender system [Shi et al., 2018], and network visualization [Jian Tang, Jingzhou Liu, et al., 2016]. Among them, node classification and link prediction are two important tasks related to node embedding and edge embedding, which have many applications and will be used frequently in this thesis. We define them as follows:

Definition 3.1.9 Node Classification: *Node classification is a task whose goal is to predict a label $y \in \mathcal{Y}$ for each node $v \in \mathcal{V} \setminus \mathcal{V}_{train}$ given a training set of nodes $\mathcal{V}_{train} \subset \mathcal{V}$ whose true labels are known, where $\mathcal{Y}$ is a set of predefined categories, types, or attributes. In general, node classification is conducted by applying a classifier on the set of node embeddings. Compared to classical supervised classification, in the case of graphs, we can not assume that each data point is statistically independent of all other data points.*

Definition 3.1.10 Link Prediction: *Link prediction is a task whose goal is to predict whether an edge exists between two nodes. More formally, given a set of nodes $\mathcal{V}$ and an incomplete set of edges between these nodes $\mathcal{E}_{train} \subset \mathcal{E}$. Our goal is to use this partial information to infer the missing edges $\mathcal{E} \setminus \mathcal{E}_{train}$.*

3.1.3 Representative Node-Level Graph Embedding Models

In this thesis, we mainly focus on learning node embeddings, which we call node-level graph embeddings. Following the categorization of [Cai, Zheng, and Chang, 2018], different node-level graph embedding approaches can be classified in different ways according to the input graph type (i.e., homogeneous or heterogeneous graphs) and learning model and paradigm (i.e., shallow or deep models, semi-supervised or unsupervised learning). Early graph embedding methods focus on the use of **shallow models**. These works first randomly initialize the node embeddings and then refine node embeddings through optimizing some well-designed objective functions. Inspired by the successful application of Skip-Gram [Q. Le and Mikolov, 2014] in Natural Language Processing (NLP), a series of **Skip-Gram-based models** have been proposed in recent years to encode the graph structure into continuous spatial vector representations, such as Deepwalk [Perozzi, Al-Rfou, and Skiena, 2014] or Node2Vec [Grover and Leskovec, 2016]. Recently, driven by Neural Networks (NNs), e.g., Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and autoencoders, there is an increasing interest in extending NNs to handle graphs with arbitrary structures [Z. Wu et al., 2020], namely **Graph Neural Networks (GNNs)**. Deep models are designed to use advanced NNs to learn embedding from node attributes or nonlinear relationships between nodes. A typical Deep Learning (DL)-based model is Graph Convolutional Network (GCN) [Kipf and Welling, 2016] and a series of GCN variants [Hamilton, Ying, and Leskovec, 2017b; Veličković, Cucurull, et al., 2017; Schlichtkrull et al., 2018; Z. Wu et al., 2020] have been proposed.

This section reviews the state-of-the-art node-level graph embedding models through the hierarchical taxonomy shown in Table 3.2. We first divide the models into two categories according to the network type: *homogeneous graph embedding* and *heterogeneous graph embedding*. According to whether nodes labels are provided for learning, we further categorize each category into two subgroups:

unsupervised and *semi-supervised*. Moreover, models can further be divided into *shallow* and *deep* categories according to the structure of the representation methods used.

TABLE 3.2: Graph embedding works categorized by graph type and learning method.

			Paper
HomG	Unsupervised	Shallow	DeepWalk [Perozzi, Al-Rfou, and Skiena, 2014], LINE [Jian Tang, Qu, et al., 2015], Node2Vec [Grover and Leskovec, 2016],
		Deep	DGI [Veličković, Fedus, et al., 2018]
	Semi-supervised	Shallow	-
		Deep	GCN [Kipf and Welling, 2016], GAT [Veličković, Cucurull, et al., 2017], GraphSAGE [Hamilton, Ying, and Leskovec, 2017b]
HetG	Unsupervised	Shallow	metapath2vec [Dong, Chawla, and Ananthram Swami, 2017], PME [H. Chen et al., 2018]
		Deep	
	Semi-supervised	Shallow	
		Deep	RGCN [Schlichtkrull et al., 2018], HAN [X. Wang, Ji, et al., 2019]

Homogeneous Graph Embedding

The following respectively introduces the methods of (i)*unsupervised graph embedding* and (ii)*semi-supervised graph embedding* on HomGs.

(i) Unsupervised In this setting, no labeled nodes are provided for learning node representations. Most existing algorithms fall into this category.

Inspired by Word2Vec [Mikolov, K. Chen, et al., 2013; Mikolov, Sutskever, et al., 2013], especially the Skip-Gram model, a number of recent research works have proposed network embedding learning frameworks based on random walks. The general idea of random-walk-based methods is to transform the network structure into node sequences called random walks. Then they learn latent node representations by processing random walks as the equivalent of sentences in the Skip-Gram model. Skip-Gram, described in more detail in Appendix D.1, is a language model whose goal is to find word representations that are useful for predicting the surrounding words in a sentence or document. More formally, given a sequence of training words $\{w_1, w_2, \ldots, w_l\}$, the objective of the Skip-Gram model is to maximize the following conditional probability:

$$\frac{1}{l}\sum_{i=1}^{l}\sum_{-c\leq j\leq c, j\neq 0}\log Pr\left(w_{i+j}|w_i\right), \tag{3.1}$$

where c is the size of context window. This probability represents the likelihood that w_{i+j} is the context of the word w_i.

The probability is defined as a softmax function on the embeddings of w_{i+j} and w_i:

$$Pr(w_{i+j}|w_i) = \frac{\exp\left(\Phi(w_{i+j})^T\Phi(w_i)\right)}{\sum_{w\in\mathcal{V}}\exp\left(\Phi(w)^T\Phi(w_i)\right)},$$

where $\mathcal{V}$ is the vocabulary set[4], $\Phi(\cdot)$ is the embedding mapping function. This mapping function, defined as $\Phi\colon \mathcal{V} \to \mathbb{R}^d$, embeds each word w in the vocabulary $\mathcal{V}$ into a d-dimensional latent space.[5] In order to achieve efficient optimization, negative sampling and hierarchical softmax have been proposed in [Mikolov, K. Chen, et al., 2013; Q. Le and Mikolov, 2014]. The general idea of negative sampling is to maximize the similarity of a word w_i to words in the same context (i.e., positive samples) while minimizing the similarity to those words that are not in the same context (i.e., negative samples). Furthermore, instead of using all words that are not in the same context, we randomly sample a small set of them as negative samples to reduce the number of parameters that need to be updated. Hierarchical softmax is another way to reduce the computation cost. The idea is to replace the direct softmax calculation with the computation on the binary tree, thereby reducing the computation cost from $\mathcal{O}(|\mathcal{V}|)$ to $\mathcal{O}(\log_2|\mathcal{V}|)$. A more detailed explanation is given in the Appendix D.1.

For graphs, in random-walk-based methods, random walks are considered as sentences in the language model. More specifically, a random walk of length l starting from a node $v_{w(1)}$ can be modeled as a sequence $\mathcal{W}[v_{w(1)}] = \{v_{w(1)}, \ldots, v_{w(l)}\}$, where $v_{w(i)} \in \mathcal{V}$ represents the node at the i-th position in the walk. In general, different methods differ in the strategies used to generate random walks and how the skip-gram objective is carried out. The two most typical works are:

- **DeepWalk** [Perozzi, Al-Rfou, and Skiena, 2014]: Given a graph $G = (\mathcal{V}, \mathcal{E})$, DeepWalk first samples uniformly a node v as the root of the random walk and generates a random walk of fixed-length l, each step of which is sampled from the neighbors of the last visited node in the current walk with the following distribution:

$$Pr(v_{w_{(i+1)}} = v_y | v_{w_{(i)}} = v_x) = \begin{cases} \frac{\pi_{xy}}{Z}, & \text{if } e_{xy} \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}, 1 < i < l,$$

  where $\pi_{xy} = \texttt{weight}_{xy}$ in weighed graphs else $\pi_{xy} = 1$ in the case of unweighted graphs, and Z is the normalizing constant. Then, the generated random walk is used to update the representations using the Skip-Gram approach with a hierarchical softmax [Mikolov, K. Chen, et al., 2013], formulated as Equation 3.1.

- **Node2Vec** [Grover and Leskovec, 2016]: Node2Vec can be seen as an extension of DeepWalk. It designs a biased random walk process that can balance Depth First Search (DFS) and Breadth First Search (BFS) for neighborhood exploration. More formally, it defines a second-order random walk with two parameters p and q. For instance, consider a random walk that traverses from node v_x to node v_y and now resides at node v_y, as shown in Figure 3.4. The walk now needs to decide the next step, so it evaluates the transition probabilities π_{yz} on edges e_{yz} leading from v_y. In Node2Vec, the unnormalized transition probability is set as

[4] Note that $\mathcal{V}$ here represents the vocabulary set, not the previously defined node set.

[5] If we switch to the context of graphs, we denote the mapping function from node to feature representation as $\Phi\colon \mathcal{V} \to \mathbb{R}^d$.

$\pi_{yz} = \alpha(x,z) \cdot \texttt{weight}_{yz}$, where

$$\alpha(x,z) = \begin{cases} \frac{1}{p}, & \text{if } d_{xz} = 0 \\ 1, & \text{if } d_{xz} = 1 \\ \frac{1}{q}, & \text{if } d_{xz} = 2 \end{cases},$$

and d_{xz} denotes the shortest path distance between nodes v_x and v_z, the distance here can only be $\{0,1,2\}$, because Node2Vec considers the second-order random walk. For example, in Figure 3.4, the shortest distance between node v_x and v_{z_1} is 1, so $\alpha(x,z_1) = 1$, while $\alpha(x,z_2) = 1/q$ for node v_{z_2}.

After generating random walks, Node2Vec uses negative sampling [Q. Le and Mikolov, 2014] to optimize the Skip-Gram model, in contrast to the hierarchical softmax used in DeepWalk. Negative samples are drawn from a modified unigram distribution $P_n(v) \propto \texttt{deg}_{out}(v)^{3/4}$ as proposed in [Q. Le and Mikolov, 2014], where $\texttt{deg}_{out}(v)$ is the out-degree of node v.

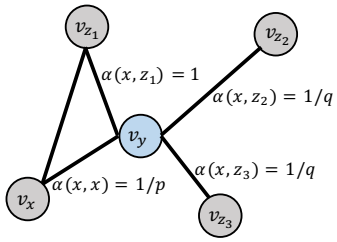


FIGURE 3.4: Illustration of the random walk generation procedure in Node2Vec, adapted from [Grover and Leskovec, 2016].

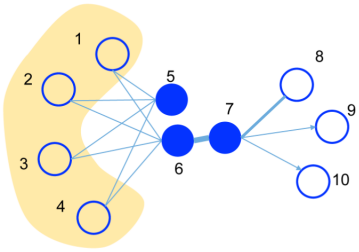


FIGURE 3.5: An example of the first-order and second-order structures in a network, from [Jian Tang, Qu, et al., 2015].

In the family of unsupervised methods, we also have models that do not use random walks, but either use explicit modeling of proximity relations or mutual information.

- **Large-scale Information Network Embedding (LINE)** [Jian Tang, Qu, et al., 2015]: LINE learns node representations by explicitly modeling the *first-order proximity* and *second-order proximity*, as defined in Definition 3.1.5 and 3.1.6, rather than exploiting random walks to capture network structure.

  - **First-order proximity** is the observed local pairwise proximity between two nodes, such as the observed edge between nodes 6 and 7 in Figure 3.5. Given an undirected edge e_{xy}, the first-order proximity between nodes v_x and v_y can be modeled by the following joint probability distribution:

$$Pr_1(v_x, v_y) = \frac{1}{1 + \exp(-\Phi(v_x)^T \Phi(v_y))}.$$

This distribution is defined over the space $\mathcal{V} \times \mathcal{V}$, with $\Phi(\cdot)$ defined in $\mathbb{R}^d$, and its empirical probability can be derived as

$$\hat{Pr}_1(v_x, v_y) = \frac{\texttt{weight}_{xy}}{\sum_{e_{x'y'} \in \mathcal{E}} \texttt{weight}_{x'y'}}.$$

To preserve the first-order proximity, we can minimize the distance between these two distributions. In LINE, the authors propose to use the KL-divergence as the distance, resulting in the following objective function (some constant omitted):

$$\mathcal{L}_1 = -\sum_{e_{xy} \in \mathcal{E}} \texttt{weight}_{xy} \log \left(\frac{1}{1 + \exp(-\Phi(v_x)^T \Phi(v_y))} \right).$$

– **Second-order proximity** is determined by the similarity of the contexts (neighbors) of two nodes. For example, the second-order similarity between nodes 5 and 6 can be obtained by the fact that they share similar neighbors 1, 2, 3, and 4 in Figure 3.5. For each directed edge (an undirected edge can be considered as two directed edges in opposite directions), the probability of generating the "context" node v_y from the source node v_x is defined as:

$$Pr_2(v_y|v_x) = \frac{\exp\left(\Phi'(v_y)^T \Phi(v_x)\right)}{\sum_{v \in \mathcal{V}} \exp\left(\Phi'(v)^T \Phi(v_x)\right)},$$

where $\Phi'(\cdot)$ is the embedding mapping function for the context node. In fact, each node v has two representations: $\Phi(v)$ (when it is treated as the source node) and $\Phi'(v)$ (when it is treated as the "context" of other nodes). The empirical distribution $\hat{Pr}_2(v_x, v_y)$ is defined as:

$$\hat{Pr}_2(v_x, v_y) = \frac{\texttt{weight}_{xy}}{\sum_{v_{y'} \in \mathcal{N}(v_x)} \texttt{weight}_{xy'}},$$

where $\mathcal{N}(v_x)$ denotes the neighbors of v_x in graph G. Similarly, LINE minimizes the KL-divergence between $Pr_2(\cdot|\cdot)$ and $\hat{Pr}_2(\cdot|\cdot)$ to preserve the second-order proximity, resulting in the following objective function:

$$\mathcal{L}_2 = -\sum_{e_{xy} \in \mathcal{E}} \texttt{weight}_{xy} \log \left(\frac{\exp\left(\Phi'(v_y)^T \Phi(v_x)\right)}{\sum_{v \in \mathcal{V}} \exp\left(\Phi'(v)^T \Phi(v_x)\right)} \right).$$

Besides the unsupervised methods described above of learning node representations by preserving various proximity between nodes, another learning scheme is by aggregating neighbor information. In particular, the aggregator utilizes the architecture of Neural Network (NN)s, such as Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM), so this kind of learning model is called Graph Neural Network (GNN), which is usually learned under supervision, as described below. Next, we explain an example of GNN that learns node representations without using labels.

- **Deep Graph Infomax (DGI) [Veličković, Fedus, et al., 2018]**: DGI is a deep model that learns node representations in an unsupervised way. It relies on maximizing the mutual information between patch representations obtained by summarizing patches around targeted nodes via neighborhood aggregation and the corresponding high-level graph summaries, both derived using the established GCN architectures [Kipf and Welling, 2016], rather than random walks. More specifically, given a graph $G = (\mathcal{V}, \mathcal{E})$, node features $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{|\mathcal{V}|}\}$ and the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, DGI learns node representations using the following loss function:

$$\mathcal{L} = \frac{1}{|\mathcal{V}| + |\mathcal{V}'|}\left(\sum_{i=1}^{|\mathcal{V}|}\mathbb{E}_{(\boldsymbol{X},\boldsymbol{A})}\left[\log \mathtt{D}(\boldsymbol{h}_i, \boldsymbol{s})\right] + \sum_{i=1}^{|\mathcal{V}'|}\mathbb{E}_{(\tilde{\boldsymbol{X}},\tilde{\boldsymbol{A}})}[\log(1 - \mathtt{D}(\tilde{\boldsymbol{h}}_i, \boldsymbol{s}))]\right), \tag{3.2}$$

  where $\boldsymbol{h}_i$ represents the high-level representation for each node, and $\boldsymbol{s}$ represents the graph-level summary vector that captures the global information content of the entire graph, $\boldsymbol{s}$ is obtained by passing node representations through a *readout function*, $\mathtt{R}\colon \mathbb{R}^{|\mathcal{V}|\times d} \to \mathbb{R}^d$, i.e., $\boldsymbol{s} = \mathtt{R}(\Phi(\boldsymbol{X}, \boldsymbol{A}))$, where $\Phi\colon \mathbb{R}^{|\mathcal{V}|\times F} \times \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|} \to \mathbb{R}^{|\mathcal{V}|\times d}$ is an embedding mapping function that takes the node feature matrix and the adjacency matrix as input. $\mathtt{D}\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *discriminator*, and $\mathtt{D}(\boldsymbol{h}_i, \boldsymbol{s})$ represents the probability score assigned to a node-summary pair $(\boldsymbol{h}_i, \boldsymbol{s})$. The right side of the loss calculates the probability score assigned to negative samples, provided by pairing the summary from $(\boldsymbol{X}, \boldsymbol{A})$ with "fake" node representation $\tilde{\boldsymbol{h}}_j$ of an alternative graph $(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{A}})$.

TABLE 3.3: A quick summary of the above unsupervised HomG embedding methods.

Paper	Method
DeepWalk [Perozzi, Al-Rfou, and Skiena, 2014]	random walk
LINE [Jian Tang, Qu, et al., 2015]	first- & second-order proximity
Node2Vec [Grover and Leskovec, 2016]	random walk
DGI [Veličković, Fedus, et al., 2018]	GNN + mutual information

(ii) Semi-supervised In this setting, some labeled nodes are used for representation learning. Since the category labels of nodes are strongly correlated with the graph structure and node attributes, semi-supervised graph embedding models are proposed to take advantage of available node labels for seeking more effective representations. This section introduces some semi-supervised methods that learn node embeddings through supervised tasks, such as node classification and link prediction.

Recently, motivated by Neural Networks (NNs), there has been a growing number of works extending NNs to deal with graphs of arbitrary structures, namely Graph Neural Networks (GNNs) [Z. Wu et al., 2020]. The general idea of GNNs is to iteratively propagate the state of each node and then generate a representation based on its state and neighboring nodes through a NN structure until equilibrium is reached. A common scheme for defining GNNs is the message-passing framework [Gilmer et al., 2017], which in its simplest form works in three steps: gathering all messages from the neighbors of each node, aggregating these messages, and updating the node representation. More formally, for

the message-passing phase from layer $(l-1)$ to layer (l), and for node v_x, we consider the message $\boldsymbol{m}_{x\leftarrow y}^{(l)}$ passed from its neighbor node v_y:

$$\boldsymbol{m}_{x\leftarrow y}^{(l)} = \texttt{Msg}^{(l)}\left(\boldsymbol{h}_x^{(l-1)}, \boldsymbol{h}_y^{(l-1)}, \boldsymbol{e}_{xy}\right),$$

where $\texttt{Msg}(\cdot)$ is the message function and $\boldsymbol{e}_{xy}$ is the edge feature of the edge e_{xy}. $\boldsymbol{h}_x^{(l)} \in \mathbb{R}^{d^{(l)}}$ is the hidden state of node v_x at l-th layer, with $d^{(l)}$ being the dimension of the representation. The messages of all neighbors $\mathcal{N}(v_x)$ of node v_x are integrated through the aggregation function $\texttt{Agg}(\cdot)$ to build $\boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)}$ as:

$$\boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)} = \texttt{Agg}_{y\in\mathcal{N}(v_x)}^{(l)}\left(\boldsymbol{m}_{x\leftarrow y}^{(l)}\right).$$

Then the state of node v_x is updated by an update function $\texttt{Upt}(\cdot)$:

$$\boldsymbol{h}_x^{(l)} = \texttt{Upt}^{(l)}\left(\boldsymbol{h}_x^{(l-1)}, \boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)}\right).$$

The message function $\texttt{Msg}(\cdot)$ and update function $\texttt{Upt}(\cdot)$ are learned differentiable functions that can be designed to build various GNN architectures. In the following, we present some important architectures proposed in the literature. For an exhaustive review of GNNs, readers are referred to [Z. Wu et al., 2020].

- **Graph Convolutional Network(GCN)** [Kipf and Welling, 2016]: GCN can be understood as a special case of the message-passing framework, with the following message function $\texttt{Msg}(\cdot)$ and update function $\texttt{Upt}(\cdot)$. Here, we consider

  $$\texttt{Msg}^{(l)}\left(\boldsymbol{h}_x^{(l-1)}, \boldsymbol{h}_y^{(l-1)}, \boldsymbol{e}_{xy}\right) = \frac{1}{c_{xy}}\boldsymbol{W}^{(l-1)}\boldsymbol{h}_y^{(l-1)} + \boldsymbol{b}^{(l-1)},$$

  where $c_{xy} = \sqrt{|\mathcal{N}(v_x)|\cdot|\mathcal{N}(v_y)|}$ is a normalization constant for the edge e_{xy}, $\boldsymbol{W}^{(l-1)}$ and $\boldsymbol{b}^{(l-1)}$ are layer-specific weight matrix and bias, respectively.

  The aggregate function $\texttt{Agg}(\cdot)$ is the summation:

  $$\boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)} = \sum_{y\in\mathcal{N}(v_x)} \boldsymbol{m}_{x\leftarrow y}^{(l)}.$$

  Then the hidden state $\boldsymbol{h}_x^{(l)}$ of v_x at l-th layer is updated by:

  $$\texttt{Upt}^{(l)}\left(\boldsymbol{h}_x^{(l-1)}, \boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)}\right) = \sigma\left(\boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)}\right),$$

  where $\sigma(\cdot)$ is an activation function.

- **SAmple and aggreGatE (GraphSAGE)** [Hamilton, Ying, and Leskovec, 2017b]: GraphSAGE generalizes the GCN approach with trainable aggregate functions. In the embedding generation step, nodes aggregate information from their local neighbors. As this process iterates, nodes

gradually gain more and more information from further out in the graph. The aggregate function $\texttt{Agg}(\cdot)$ is formulated as:

$$\boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)} = \texttt{Agg}^{(l)}\left(\{\boldsymbol{h}_y^{(l-1)}\}_{y\in\mathcal{N}(v_x)}\right).$$

After this aggregation, GraphSAGE concatenates it with the current representation of the source node v_x. Then, this concatenated vector is fed through a fully-connected layer with a nonlinear activation function $\sigma(\cdot)$ to generate representations $\boldsymbol{h}_x^{(l)}$, used in the next step.

$$\boldsymbol{h}_x^{(l)} = \sigma\left(\boldsymbol{W}^{(l)} \cdot \texttt{Concat}\left(\boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)}, \boldsymbol{h}_x^{(l-1)}\right)\right).$$

Four aggregate functions $\texttt{Agg}(\cdot)$ are tested in [Hamilton, Ying, and Leskovec, 2017b]:

- **Mean aggregator**: it takes the element-wise mean of the neighbor embedding vectors, then concatenates it with the source node embedding vector:

$$\boldsymbol{m}_{\mathcal{N}_{(v_x)}}^{(l)} = \texttt{Mean}\left(\{\boldsymbol{h}_y^{(l-1)}\}_{y\in\mathcal{N}(v_x)}\right),$$
$$\boldsymbol{h}_x^{(l)} = \sigma\left(\boldsymbol{W}^{(l)} \cdot \texttt{Concat}\left(\boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)}, \boldsymbol{h}_x^{(l-1)}\right)\right).$$

- **GCN aggregator**: it takes the mean operation over representations of all nodes (i.e., neighbors and the source) directly:

$$\boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)} = \texttt{Mean}\left(\{\boldsymbol{h}_y^{(l-1)}\}_{y\in\mathcal{N}(v_x)} \cup \boldsymbol{h}_x^{(l-1)}\right),$$
$$\boldsymbol{h}_x^{(l)} = \sigma\left(\boldsymbol{W}^{(l)} \cdot \boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)}\right).$$

- **LSTM aggregator**: it applies a Long Short Term Memory (LSTM) to a random permutation of node neighbors:

$$\boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)} = \texttt{LSTM}\left(\texttt{Perm}(\mathcal{N}(v_x))\right),$$
$$\boldsymbol{h}_x^{(l)} = \sigma\left(\boldsymbol{W}^{(l)} \cdot \texttt{Concat}\left(\boldsymbol{m}_{\mathcal{N}(v_x)}^{(l)}, \boldsymbol{h}_x^{(l-1)}\right)\right).$$

- **Pooling aggregator**: it feeds each neighbor vector into a fully-connected NN, then it applies an element-wise max-pooling operation:

$$\boldsymbol{h}_x^{(l)} = \max\left(\left\{\sigma\left(\boldsymbol{W}_{pool}^{(l-1)}\boldsymbol{h}_y^{(l-1)} + \boldsymbol{b}^{(l-1)}\right)\right\}_{y\in\mathcal{N}(v_x)}\right).$$

- **GAT** [Veličković, Cucurull, et al., 2017]: In the GAT approach, the authors propose to leverage the self-attention strategy [Vaswani et al., 2017] into the context of the graph to take into account the importance between a node and its neighbors. More specifically, a single Graph Attentional Layer (GAL) is introduced, where the input is a set of N node features $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}, \boldsymbol{x}_i \in \mathbb{R}^F$. The attention coefficient between two nodes v_x and v_y is calculated

as:

$$a_{xy} = \texttt{Att}\left(\boldsymbol{W}\boldsymbol{x}_x, \boldsymbol{W}\boldsymbol{x}_y\right),$$

where $\boldsymbol{W} \in \mathbb{R}^{F' \times F}$ is a weight matrix used to map node features to a new F'-dimensional space. The attention mechanism $\texttt{Att}$ used in the original paper [Veličković, Cucurull, et al., 2017] is a single-layer feed-forward neural network, parametrized with a weight vector $\boldsymbol{q} \in \mathbb{R}^{2F'}$ and a LeakyReLU activation function:

$$a_{xy} = \texttt{LeakyReLU}\left(\boldsymbol{q}^T\left[\boldsymbol{W}\boldsymbol{x}_x \oplus \boldsymbol{W}\boldsymbol{x}_y\right]\right).$$

Here, $\oplus$ is the concatenation operation, which joins two vectors end-to-end.

The graph structure is injected into the attention mechanism by performing masked attention. In other words, we only calculate a_{xy} for nodes in the neighborhood set of the node v_x (i.e., $v_y \in \mathcal{N}(v_x)$). Additionally, to make attention coefficients easy to compare across different nodes, a_{xy} is normalized by a softmax function:

$$\alpha_{xy} = \texttt{softmax}(a_{xy}) = \frac{\exp(a_{xy})}{\sum_{v_{y'} \in \mathcal{N}(v_x)} \exp\left(a_{xy'}\right)}.$$

In order to stabilize the learning process of self-attention, the final output feature of each node is a concatenation of the features obtained by K-independent attention mechanisms [Vaswani et al., 2017]:

$$\boldsymbol{h}_x = \|_{k=1}^{K} \sigma\left(\sum_{y \in \mathcal{N}(v_x)} \alpha_{xy}^k \boldsymbol{W}^k \boldsymbol{x}_y\right),$$

where $\sigma(\cdot)$ is the sigmoid function. Note that, in this setting, the final hidden state $\boldsymbol{h}_x$ consists of KF' features (instead of F') for each node.

TABLE 3.4: A quick summary of the above semi-supervised HomG embedding methods.

Paper	Method
GCN [Kipf and Welling, 2016]	message-passing + graph convolutional aggregation
GAT [Veličković, Cucurull, et al., 2017]	self-attention mechanism
GraphSAGE [Hamilton, Ying, and Leskovec, 2017b]	GCN approach + trainable aggregate function

Heterogeneous Graph Embedding

Heterogeneous graphs are composed of different types of nodes and/or edges. The previously described methods cannot be directly used for heterogeneous graphs since they do not consider the types of nodes and edges. In the following, we present some important approaches for heterogeneous graph embedding following the same introduction structure as homogeneous graphs: (i) *unsupervised* and (ii) *semi-supervised*.

(i) Unsupervised Many unsupervised heterogeneous models are also based on random walk strategies, but they have to be adjusted to account for node and edge types. In particular, the concept of meta-path, introduced in [Dong, Chawla, and Ananthram Swami, 2017], can be used to guide random walks on HetGs.

Definition 3.1.11 Meta-Path: *Given a HetG $G = (\mathcal{V}, \mathcal{E}, \mathcal{T}_V, \mathcal{T}_E)$, a meta-path scheme $\mathcal{P}$ is defined as the path, expressed as $T_V^1 \xrightarrow{T_E^1} T_V^2 \xrightarrow{T_E^2} \dots \xrightarrow{T_E^{L-1}} T_V^L$, where $T_V^i \in \mathcal{T}_V$ is a node type and $T_E^j \in \mathcal{T}_E$ is an edge type. $T_E^1 \circ T_E^2 \cdots \circ T_E^{L-1}$ defines the composite relations (edge types) between node types T_V^1 and T_V^L, and $\circ$ denotes a composition operator on relations.*

We give an example of such a meta-path in the job transition heterogeneous graph depicted in Figure 3.6, where node types include "person" (P), "job position" (J) and "company" (C). The "colleague" relation can be described by the 5-length meta-path, $P \xrightarrow{\text{work-as}} J \xrightarrow{\text{in}} C \xrightarrow{\text{has}} J \xrightarrow{\text{work-by}} P$, or short as PJCJP if there is no ambiguity, e.g., Person 1 $\xrightarrow{\text{work-as}}$ Job title 1 $\xrightarrow{\text{in}}$ Company 1 $\xrightarrow{\text{has}}$ Job title 4 $\xrightarrow{\text{work-by}}$ Person 4.

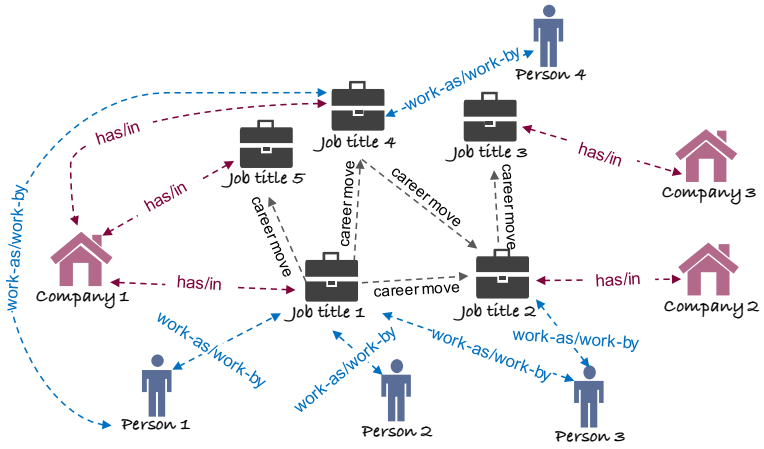


FIGURE 3.6: An heterogeneous job transition graph with three node types, i.e., "person", "job titles" and "company names", and five edge types, i.e., "work-as", "work-by", "career moves", "has", and "in".

Using this concept of meta-path, we can now define the meta-path-based neighborhood and apply the meta-path to random walks.

Definition 3.1.12 Meta-Path-Based Neighbors: *Given a node v and a meta-path scheme $\mathcal{P}$ in a HetG $G = (\mathcal{V}, \mathcal{E}, \mathcal{T}_V, \mathcal{T}_E)$, the meta-path-based neighbors $\mathcal{N}(v)^{\mathcal{P}}$ of node v are defined as the set of nodes which connect with node v via the meta-path $\mathcal{P}$.*

Definition 3.1.13 Meta-Path-Based Random Walk: *Given a HetG $G = (\mathcal{V}, \mathcal{E}, \mathcal{T}_V, \mathcal{T}_E)$, and a meta-path scheme $\mathcal{P} = T_V^1 \xrightarrow{T_E^1} T_V^2 \xrightarrow{T_E^2} \dots \xrightarrow{T_E^{L-1}} T_V^L$. A meta-path-based random walk is a path built with the following transition probability at each step i, and conditioned on a predefined meta-path scheme $\mathcal{P}$. The transition probability at i-th step, i.e., considering the step $v_x \to v_{x+1}$ of the walk and knowing*

that the type of node v_x is T_V^t (i.e., $\psi(v_x) = T_V^t$) , is defined as follows:

$$p(v_{x+1} \mid v_x, \mathcal{P}) = \begin{cases} \frac{1}{\left|\mathcal{N}(v_x)^{T_V^{t+1}}\right|} & (v_x, v_{x+1}) \in \mathcal{E}, \psi(v_{x+1}) = T_V^{t+1}, \\ 0 & (v_x, v_{x+1}) \in \mathcal{E}, \psi(v_{x+1}) \neq T_V^{t+1}, \\ 0 & (v_x, v_{x+1}) \notin \mathcal{E}, \end{cases}$$

where $\mathcal{N}(v_x)^{T_V^{t+1}}$ is the set of neighbor nodes of v_x of type T_V^{t+1}.

The meta-path-based random walk strategy ensures that the semantic relations between different types of nodes can be properly incorporated into the Skip-Gram model. With this strategy, the Skip-Gram strategy used in homogeneous graphs can be used for heterogeneous graph embeddings. We present below two of them.

- **metapath2vec** [Dong, Chawla, and Ananthram Swami, 2017]: metapath2vec is a representative work for HetG, which introduces a heterogeneous Skip-Gram model. To incorporate heterogeneous network structures into Skip-Gram, it uses meta-path-guided random walks to model the heterogeneous neighborhoods of nodes. More specifically, given a meta-path $\mathcal{P} : T_V^1 \xrightarrow{T_E^1} T_V^2 \xrightarrow{T_E^2} \dots \xrightarrow{T_E^{L-1}} T_V^L$ for guiding meta-path-based random walks, the objective of heterogeneous Skip-Gram is:

$$\frac{1}{l}\sum_{i=1}^{l}\sum_{T_V^t \in \{T_V^1,\dots,T_V^L\}}\sum_{-c\leq j\leq c, j\neq 0} \log Pr\left(v_{x+j}^{T_V^{t+j}} \mid v_x^{T_V^t}\right),$$

  where $\{v_{x+j}^{T_V^{t+j}}\}_{-c\leq j\leq c, j\neq 0}$ is the T_V^{t+j} type context of node $v_x^{T_V^t}$. The probability $Pr(v_{x+j}^{T_V^{t+j}} \mid v_x^{T_V^t})$ is defined as:

$$Pr\left(v_{x+j}^{T_V^{t+j}} \mid v_x^{T_V^t}\right) = \frac{\exp\left(\Phi(v_{x+j}^{T_V^{t+j}})^T \Phi(v_x^{T_V^t})\right)}{\sum_{v\in\mathcal{V}} \exp\left(\Phi(v)^T \Phi(v_x^{T_V^t})\right)},$$

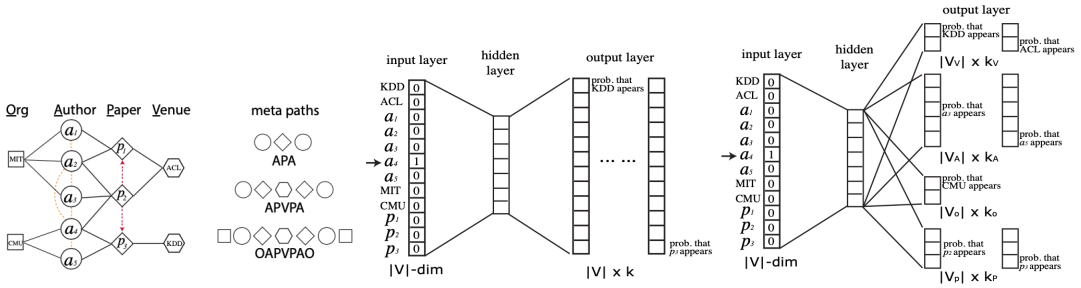
  where $\Phi(\cdot)$ is the embedding mapping function.

- **metapath2vec++** [Dong, Chawla, and Ananthram Swami, 2017]: metapath2vec++ extends metapath2vec by improving heterogeneous negative sampling, where the softmax function is normalized with respect to the node type of the context. Specifically, $Pr(v_{x+j}^{T_V^{t+j}} \mid v_x^{T_V^t})$ is adjusted to the specific node type T_V^{t+j}:

$$Pr\left(v_{x+j}^{T_V^{t+j}} \mid v_x^{T_V^t}\right) = \frac{\exp\left(\Phi(v_{x+j}^{T_V^{t+j}})^T \Phi(v_x^{T_V^t})\right)}{\sum_{v\in\mathcal{V}^{T_V^{t+j}}} \exp\left(\Phi(v)^T \Phi(v_x^{T_V^t})\right)},$$

  where $\mathcal{V}^{T_V^{t+j}}$ is the node set of type T_V^{t+j} in the graph. metapath2vec++ specifies a set

of multinomial distributions for each type of neighborhood in the output layer of the Skip-Gram model. Recall that in metapath2vec, DeepWalk and Node2Vec, the dimension of the output multinomial distribution is equal to the number of nodes in the network. However, in metapath2vec++, the multinomial distribution dimension for type T_V^t nodes is determined by the number of nodes of type T_V^t. An illustration is shown in Figure 3.7c.



(a) An academic network.

(b) Skip-Gram in DeepWalk, Node2Vec, and metapath2vec.

(c) Skip-Gram in metapath2vec++

FIGURE 3.7: An illustrative example of a heterogeneous academic network and Skip-Gram architectures of metapath2vec and metapath2vec++ for embedding this network, from [Dong, Chawla, and Ananthram Swami, 2017].

TABLE 3.5: A quick summary of the above unsupervised HetG embedding methods.

Paper	Method
metapath2vec [Dong, Chawla, and Ananthram Swami, 2017]	meta-path-based random walk
metapath2vec++ [Dong, Chawla, and Ananthram Swami, 2017]	metapath2vec + heterogeneous negative sampling

(ii) Semi-supervised In order to account for graph heterogeneity when learning node representations, some works propose ways to adapt homogeneous GNNs to heterogeneous graphs by considering node types and/or edge types. In the following, two representative heterogeneous GNNs are introduced:

- **Relational Graph Convolutional Network (RGCN)** [Schlichtkrull et al., 2018]: RGCN is primarily motivated as an extension to GCN [Kipf and Welling, 2016], which introduces relation-specific transformations. The model is formulated as:

$$\boldsymbol{h}_x^{(l)} = \sigma\left(\sum_{T_E^t \in \mathcal{T}_E} \sum_{v_y \in \mathcal{N}(v_x)^{T_E^t}} \frac{1}{c_{v_x}^{T_E^t}} \boldsymbol{W}_{T_E^t}^{(l-1)} \boldsymbol{h}_y^{(l-1)} + \boldsymbol{W}_0^{(l-1)} \boldsymbol{h}_x^{(l-1)}\right),$$

where $\mathcal{N}(v_x)^{T_E^t}$ denotes the neighbor set of node v_x with respect to the edge type $T_E^t \in \mathcal{T}_E$, and $c_{v_x}^{T_E^t}$ is a normalization constant equal to $|\mathcal{N}(v_x)^{T_E^t}|$. $\boldsymbol{W}_{T_E^t}^{(l-1)}$ is the weight matrix specific to the T_E^t-type edges at $(l-1)$-layer, and $\boldsymbol{W}_0^{(l-1)}$ is the self-connection weight matrix. Intuitively,

the aggregate function $\texttt{Agg}(\cdot)$ used in RGCN is a normalized summation like in GCN. Different from the regular GCN, RGCN considers relation-specific transformations, i.e., depending on the edge type. A single self-connection representation of a particular edge type is added to ensure that the representation of a specific node at $(l+1)$-th layer can also be informed by the corresponding representation at (l)-th layer each node in the graph. The updates for a particular node (i.e., red) are shown in Figure 3.8, where the neighboring node representations (i.e., blue) are gathered and then transformed separately for each relation type (for both in- and outgoing edges). The resulting representations (i.e., green) are accumulated in a normalized sum and passed through an activation function. Such per-node updates can be computed in parallel with shared parameters across the whole graph. The final hidden states of nodes are then fed into the softmax activation (per node) for the node classification task.

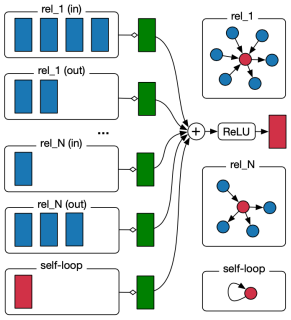


FIGURE 3.8: The computation graph for a single node update in the RGCN model, from [Schlichtkrull et al., 2018].

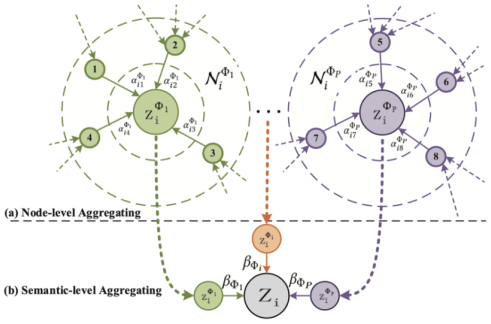


FIGURE 3.9: Illustration of the node-level and semantic-level aggregating process in HAN, from [X. Wang, Ji, et al., 2019].

- **HAN** [X. Wang, Ji, et al., 2019]: As with homogeneous networks, several works have proposed to leverage the attention mechanism for heterogeneous graph embedding. In particular, HAN proposes a hierarchical attention mechanism, i.e., node-level and semantic-level, for HetGs, as illustrated in Figure 3.9. Specifically, the node-level attention mechanism is similar to GAT [Veličković, Cucurull, et al., 2017], except that HAN further considers the node type. Therefore, HAN can assign different weights to neighbors according to the meta-path. More specifically, the authors first use a type-specific transformation matrix $\boldsymbol{W}_{\psi(v_x)}$ to project the features of different types of nodes into the same feature space, i.e., $\boldsymbol{h}_x = \boldsymbol{W}_{\psi(v_x)} \cdot \boldsymbol{x}_x$, where $\boldsymbol{x}_x$ and $\boldsymbol{h}_x$ are the original and projected features of node v_x and $\psi(\cdot)$ is the node type mapping function that returns the type of node v_x. Afterward, a self-attention mechanism is used to learn node-level attention $a^{\mathcal{P}}_{xy}$ between each pair of nodes (v_x, v_y) connected by a meta-path $\mathcal{P}$. The calculation formula is as follows:

$$a^{\mathcal{P}}_{xy} = \texttt{att}_{node}\left(\boldsymbol{h}_x, \boldsymbol{h}_y, \mathcal{P}\right) = \sigma\left(\boldsymbol{a}^{\mathcal{P}^T} \cdot \left[\boldsymbol{h}_x \oplus \boldsymbol{h}_y\right]\right),$$

where σ represents the activation function, $\oplus$ represents the concatenate operation, and $\boldsymbol{a}^{\mathcal{P}}$ is the node-level attention vector of the meta-path $\mathcal{P}$.

The structural information is injected into the model through the masked attention similar to GAT [Vaswani et al., 2017], which means only calculating $a_{xy}^{\mathcal{P}}$ for node $v_y \in \mathcal{N}(v_x)^{\mathcal{P}}$, so the attention $a_{xy}^{\mathcal{P}}$ is normalized by the following softmax function:

$$\alpha_{xy}^{\mathcal{P}} = \texttt{softmax}(a_{xy}^{\mathcal{P}}) = \frac{\exp\left(a_{xy}^{\mathcal{P}}\right)}{\sum_{v_{y'} \in \mathcal{N}(v_x)^{\mathcal{P}}} \exp\left(a_{xy'}^{\mathcal{P}}\right)}.$$

Then the meta-path-based embedding of node v_x is aggregated by the projected features of meta-path-based neighbors with corresponding attention scores:

$$\boldsymbol{z}_x^{\mathcal{P}} = \|_{k=1}^{K} \sigma\left(\sum_{v_y \in \mathcal{N}(v_x)^{\mathcal{P}}} \alpha_{xy}^{\mathcal{P}} \cdot \boldsymbol{h}_y\right).$$

Given the meta-path set $\{\mathcal{P}_1, \ldots, \mathcal{P}_P\}$, after feeding node features into node-level attention, there are P groups of semantic-specific node embeddings, expressed as $\{\boldsymbol{Z}^{\mathcal{P}_1}, \ldots, \boldsymbol{Z}^{\mathcal{P}_P}\}$. Then the semantic-level attention learns the importance of different meta-paths as:

$$b^{\mathcal{P}_p} = \texttt{att}_{semantic}\left(\boldsymbol{Z}_x^{\mathcal{P}_1}, \ldots, \boldsymbol{Z}_x^{\mathcal{P}_P}\right) = \frac{1}{|\mathcal{V}|} \sum_{v_x \in \mathcal{V}} \boldsymbol{b}^T \cdot \texttt{tanh}\left(\boldsymbol{W}_s \cdot \boldsymbol{z}_x^{\mathcal{P}_p} + \boldsymbol{b}_s\right).$$

The semantic-specific node representation $\boldsymbol{z}_x^{\mathcal{P}_p}$ is first transformed through a Multilayer Perceptron (MLP) layer with parameters $\boldsymbol{W}_s$ and $\boldsymbol{b}_s$. Then, the importance of the semantic-specific representation is measured as the similarity between the transformed representation and the semantic-level attention vector $\boldsymbol{b}$, which is then normalized by the softmax function:

$$\beta^{\mathcal{P}_p} = \frac{\exp(b^{\mathcal{P}_p})}{\sum_{p'=1}^{P} \exp(b^{\mathcal{P}_{p'}})}.$$

The weight of the meta-path $\mathcal{P}_p$ can be interpreted as the contribution of the meta-path $\mathcal{P}_p$ to a specific task. With these weights, the final representation $\boldsymbol{Z}$ of a specific task is obtained by fusing all semantic-specific representations:

$$\boldsymbol{Z} = \sum_{p=1}^{P} \beta^{\mathcal{P}_p} \cdot \boldsymbol{Z}^{\mathcal{P}_p}.$$

TABLE 3.6: A quick summary of the above semi-supervised HetG embedding methods.

Paper	Method
RGCN [Schlichtkrull et al., 2018]	GCN + consider edge type
HAN [X. Wang, Ji, et al., 2019]	node-level attention (GAT + consider node type) + semantic-level attention

3.2 Related Works on Recommendation Model

This section introduces the necessary background of recommender systems, including the formulation of the recommendation problem and the main families of traditional recommendation methods, such as (i) *collaborative-filtering-based*, (ii) *content-based*, and (iii) *hybrid filtering*. We then briefly introduce widely used sequential recommendation models, which are used in the *next-application prediction* task (Chapter 6). For facilitating illustration, we list some important mathematical notations used throughout this section in Table 3.7, unless particularly specified.

TABLE 3.7: Mathematical notations used in *Related Works on Recommendation Model*.

Notation	Description
$\mathcal{U}$	user set
$\mathcal{I}$	item set
$\mathcal{U}_{\text{sim}}^{(u)}$	users who are "similar" to user u using the similarity measure
$\mathcal{I}_{\text{liked}}^{(u)}$	items "liked" by u
$\mathtt{R}(u,i)$	utility function between user u and item i
$\mathtt{content}(i), \mathtt{content}(u)$	feature of item i, profile of user u
$\mathcal{H}^u = \{S_1^u, \ldots, S_T^u\}$	historical behavior sequence of user u
$S_t^u = \{i_{t,1}, \ldots, i_{t,n_t}\}$	session of user u at the time step t
$\mathcal{S}$	session set

3.2.1 Background on Recommender Systems

Recommender systems take advantage of the past online experiences of users and then predict what users might like in the future, i.e., telling users what to buy (e.g., Amazon), which movies to watch (e.g., Netflix), and which songs to listen to (e.g., Spotify). Generally, recommendations are generated based on user preferences, item features, historical user-item interactions, and additional information such as specific time points (e.g., top news and breakthrough products). Next, we formally define the recommendation problem.

Recommendation Problem Formulation

Let $\mathcal{U}$ be the set of users, and $\mathcal{I}$ be the set of all possible items that can be recommended. The space of $\mathcal{I}$ can be huge, ranging in hundreds of thousands or even millions of items in various applications. Similarly, the user space can also be very large. Let $\mathtt{R}(u,i)$ be a utility function that measures the relatedness of item i to user u, i.e., $\mathtt{R} : \mathcal{U} \times \mathcal{I} \to K$, where K is a totally ordered set, e.g., non-negative integers or real numbers within a certain range.

For each user $u \in \mathcal{U}$, the goal of a recommender system is to recommend such an item $i^* \in \mathcal{I}$ that maximizes the utility of user u. More formally:

$$\forall u \in \mathcal{U}, i^* = \arg\max_{i \in \mathcal{I}} \mathtt{R}(u,i).$$

Each specific user $u \in \mathcal{U}$ can be described with a *user profile* that includes various user characteristics, such as age, gender, and job in the context of our job recommendation. In the simplest case, the profile can contain only a single (unique) element, i.e., the user ID. Similarly, each item $i \in \mathcal{I}$ is defined with a set of characteristics. For example, in a job recommendation application, where $\mathcal{I}$ is a collection of job postings, each of which can be represented by its ID, job title, job location, job requirements, and mandatory skills. The utility of an item is usually represented by a *rating*, indicating how well a particular item satisfies the preference of a particular user. Furthermore, the utility function $\mathtt{R}(u,i)$ is usually not defined on the entire $\mathcal{U} \times \mathcal{I}$ space, but only on some subset of it (i.e., items the user has interacted with before). Therefore, the central objective of recommender systems is to predict the ratings of non-interacted user-item pairs. Once predicted ratings are generated, items are recommended to the user by selecting the highest rating among all the predicted ratings for that user. Alternatively, we can recommend the items with the highest N rating to the user.

Recommender systems are usually categorized according to the way they perform rating estimation. Traditional recommendation techniques [Adomavicius and Tuzhilin, 2005; Ricci, Rokach, and Shapira, 2010] are mainly divided into three categories: (i) *collaborative filtering*, (ii) *content-based*, and (iii) *hybrid filtering*, which are briefly described below.

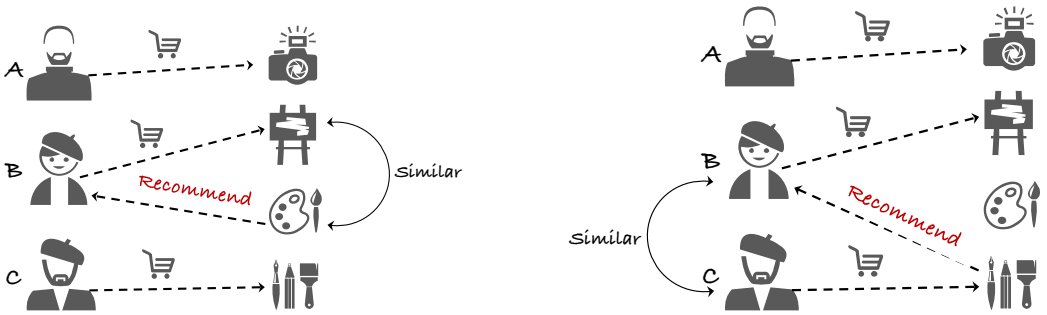
Collaborative Filtering

Collaborative filtering is one of the most successful approaches for building recommender systems. It is based on the assumption that users tend to like things similar to things they liked before and those liked by other users who share similar interests. Therefore, collaborative-filtering-based methods make recommendations by learning from user-item historical interactions, either explicit (e.g., the previous ratings of users) or implicit feedback (e.g., browsing history). The two types of collaborative-filtering-based methods are *memory-based* and *model-based* approaches [Breese, Heckerman, and Kadie, 2013]:

Memory-based Memory-based collaborative filtering uses ratings or interaction records of users to calculate the similarity between users or items. Accordingly, under this category, two subcategories: *item-based collaborative filtering* and *user-based collaborative filtering* are further separated.

- In the *item-based collaborative filtering*, users receive item recommendations similar to items they liked in the past. More formally, the utility $\mathtt{R}(u,i)$ of user u and item i is estimated from the utilities previously assigned by user u to other items, i.e., $\{\mathtt{R}(u,i')\}_{i' \in \mathcal{I}_{\text{liked}}^{(u)}}$. Here, $\mathcal{I}_{\text{liked}}^{(u)} \subseteq \mathcal{I}$ represents the items "liked" by user u. For example, as shown in Figure 3.10a, since *palette* is similar to *canvas* (purchased by the user B), we recommend *palette* to the user B.

- In the *user-based collaborative filtering*, users receive recommendations for items that similar users like. Similar to *Item-based*, the utility $\mathtt{R}(u,i)$ of user u and item i is estimated based on the utilities assigned to item i by users who are "similar" to user u and who have rated item i, i.e., $\{\mathtt{R}(u',i)\}_{u' \in \mathcal{U}_{\text{sim}}^{(u)}}$. Here, $\mathcal{U}_{\text{sim}}^{(u)} \subseteq \mathcal{U}$ denotes users who are "similar" to user u. For example, as shown in Figure 3.10b, since user B is similar to user C (they all like painting), and user C bought *painting tools*, we recommend *painting tools* to the user B.



(a) Item-based collaborative filtering.

(b) User-based collaborative filtering.

FIGURE 3.10: Two types of memory-based collaborative filtering.

Similarities between item-item pairs and user-user pairs can be calculated in different ways, for example, cosine-based and correlation-based metrics such as Pearson [Fkih, 2022]. Memory-based collaborative filtering suffers from two fundamental problems: sparsity and scalability. Interactions between items and users are usually sparse due to the lack of historical data. Therefore, the accuracy of this method is generally poor. As for scalability, memory-based approaches typically cannot handle large numbers of users and projects.

Model-based. In contrast to memory-based methods, model-based algorithms use a collection of ratings to learn a model, which is then used to make rating predictions. The model-building process can be done using Machine Learning (ML) or Data Mining (DM) techniques. These techniques include Bayesian models, clustering models, and Matrix Factorization. The clustering technique is based on the assumption that users in the same cluster have the same interests. Therefore, users are partitioned into clusters. Matrix Factorization attempts to decompose the user-item interaction matrix into low-rank matrices. The main drawbacks of Matrix-Factorization-based methods lie in: (i) Most of them only consider the lower-order interactions, i.e., first- or second-order, while ignoring possible high-order interactions. (ii) They ignore temporal dependencies between behaviors within and across different sessions.

Advantages vs. Disadvantages of Collaborative Filtering A key advantage of collaborative filtering methods is that they are completely independent of machine-readable recommended item representations. However, they suffer from the item cold-start problem.

Content-Based

Another widely used recommender system design approach is content-based. It is primarily based on the comparison of auxiliary information of user and item. A diverse range of auxiliary information such as texts, images, and videos can be considered, but content-based systems mainly focus on textual information, such as reviews written for restaurants and resume forms filled out in job portals. Therefore, the content-based approach originates in Information Retrieval (IR).

More formally, let $\texttt{content}(i)$ be the item information, i.e., a set of attributes that characterizes item i. As mentioned earlier, content-based systems are designed mostly to recommend textual items,

and the content in these systems is usually described by *terms*, i.e., keywords and short phrases. Similarly, let $\texttt{content}(u)$ be the user profile of user u, which contains his/her taste and preferences. This profile is obtained by analyzing some personal information (e.g., gender, age) or the content of items the user has interacted with before. Specifically, given the personal information $\mathcal{P}_u$ and historical interaction item set $\mathcal{I}^{(u)} = \{i_1, \ldots, i_m\}$ of user u, each item i is characterized by $\texttt{content}(i)$. Thus, the user profile $\texttt{content}(u)$ can be calculated from individual item content and personal information $\mathcal{P}_u$ using various techniques such as averaging or summing.

In content-based systems, the utility function $\texttt{R}(u, i)$ is usually defined as:

$$\texttt{R}(u, i) = \texttt{score}\big(\texttt{content}(u), \texttt{content}(i)\big).$$

Here, the matching score between $\texttt{content}(i)$ and $\texttt{content}(u)$ can be calculated by various metrics, such as cosine or Pearson similarity if the two contents belong to the same latent representation space.

According to the utility function, the main difference between the content-based approach and the collaborative-filtering-based approach is the process of building the latent representations of contents (i.e., $\texttt{content}(i)$ and $\texttt{content}(u)$) and the way of calculating the matching score. The best-known traditional approaches to leverage textual representation in IR is Bag-of-Word (BoW) and Term Frequency Inverse Document Frequency (TF-IDF) [Salton and Buckley, 1988]. Topic Modeling, a common technique in text mining, has also been explored for content representation learning [Jin, Y. Zhou, and Mobasher, 2005; C. Wang and Blei, 2011]. A potential drawback of these approaches is that they do not account for word orders and surrounding contexts. More recently, word embeddings have shown excellent results in many NLP tasks and are used to improve the content representation task in order to build better-performing recommender systems [Musto et al., 2015; Musto et al., 2016].

Advantages vs. Disadvantages of Content-Based An advantage of content-based methods over collaborative filtering-based methods is that they can process new items with little or no previous user interactions. Another advantage is that they provide transparency into how recommender systems work, making recommendations easy to explain. However, content-based methods require a wealth of information about items and users.

Hybrid Filtering

In order to achieve high performance and overcome the shortcomings of each traditional recommendation technique, hybrid recommender systems have often been proposed [Burke, 2002; Jin, Y. Zhou, and Mobasher, 2005; Burke, 2007]. They combines the best features of two or more recommendations techniques into a hybrid structure, e.g., combining collaborative-filtering-based techniques with other recommendation techniques to avoid cold-start and sparsity issues.

3.2.2 Sequential Recommendation Models

The recommendation methods mentioned above have demonstrated their effectiveness in research and industry applications. However, these methods typically focus on the long-term preferences of users, do not take into account the short-term behavior or intention in their recommendations, and ignore

the order of behaviors. In practice, (i) historical behaviors in different periods have different effects on the construction of user preferences, and (ii) long-term user behaviors are often unavailable in certain application areas of recommender systems. Consequently, taking temporal behavior into account has become one of the most important and challenging problems in recommender systems [S. Wang, L. Hu, et al., 2019]. To this end, *sequential recommendation* (is also related to *session-based*) systems have recently been proposed. Note that since behaviors within a session are also sequential, in this work, we use *sequential recommendation*, which is a much broader term than *session-based recommendation*, to describe this kind of recommendation problem that models the sequential behavior of users in a specific period. Using the notations described in Table 3.7, we first formally define *session-based recommendation*, and then give a more general definition of *sequential recommendation*.

Sequential Recommendation Formulation

Given a historical behavior sequence ordered by time $\mathcal{H}^u = \{S_1^u, \ldots, S_T^u\}$ of a specific user $u \in \mathcal{U}$, where T is the total number of time steps (events), and each event $S_t^u = \{i_{t,1}, \ldots, i_{t,n_t}\}$[6]$\subseteq \mathcal{I}$ represents the collection of n_t interacted items at a time step t. S_t^u can be regarded as a behavior *session*, e.g., the set of news read or the music listened to by a user in an hour or any period. The behavior sequence is therefore composed of different sessions. In this case, recommender systems that take a session as the basic data organization unit for analyzing the sequential data and making recommendations are *session-based* recommender systems. Similar to [S. Wang, Cao, and Y. Wang, 2019], we formally define *session-based* recommendation as follows.

Definition 3.2.1 Session-Based Recommendation: *Given the session-based historical information, such as part of a session or recent historical sessions, the session-based recommendation aims to predict unknown parts of a session or future sessions based on modeling the complex relations embedded within a session or between sessions.*

Accordingly, *session-based* recommendations can be further divided into two sub-categories: (i) *next-item(s) recommendation*, which recommends a part of the current session, (ii) *next-session (next-basket) recommendation*, which recommends part or entire future session. The formal definitions are given as follows:

Definition 3.2.2 Next-Item(s) Recommendation: *Given the active session of user u at time step t, i.e., $S_t^u = \{i_{t,1}, \ldots, i_{t,n_t}\}$, and assuming that part of S_t^u is known, denoted as $\widetilde{S}_t^u = \{i_{t,1}, \ldots, i_{t,m}\}, m < n_t$, the next-item(s) recommendation task consists in predicting the next item(s) $i_{t,m'}$ in S_t^u conditional on $\widetilde{S}_t^u$, where $m < m' \leq n_t$.*

Definition 3.2.3 Next-Session Recommendation: *Given the collection of sessions of user u before a given time step t, $\{S_1^u, \ldots, S_{t-1}^u\}$, the next-session recommendation task predicts the items that may appear in session S_t^u.*

Next-item(s) recommendation and *next-session (next-basket) recommendation* both use sequential information, i.e., the sequence of items in the active session or the sequence of sessions before the active session. Therefore, as we mentioned before, in this work, we use the broader term "sequential

[6]We omit the superscript of u from the item indices without loss of clarity.

recommendation" rather than *session-based recommendation* to describe the recommendation task that explores the sequential data. Therefore, we formally define *sequential recommendation* as:

Definition 3.2.4 Sequential Recommendation: *The sequential recommendation is defined as recommending the next item(s) based on the historical sequence $\mathcal{H}^u$ of user u.*

Various methods have been proposed to make sequential recommendations based on the historical interactions of users. We first review the traditional methods, then conduct a brief survey on recent Deep Learning (DL)-based sequential recommendation methods and summarize three training strategies. Finally, we introduce commonly used evaluation metrics in sequential recommendations.

Traditional Sequential Recommendation Methods

Conventional (as opposed to DL-based) sequential recommendation methods [Quadrana, Cremonesi, and Jannach, 2018; Ludewig and Jannach, 2018] usually use Markov chain [He and McAuley, 2016], Matrix Factorization [Rendle, Freudenthaler, and Schmidt-Thieme, 2010], and K-Nearest Neighbors [Jannach and Ludewig, 2017] to capture sequential information. In the following, we only introduce the main methods used as baselines in our contribution to sequential recommendations. For better elaboration, we take five users u_A, u_B, u_C, u_D, u_E and their sequences of interaction items at t time step as an example. The following table shows these sequences, where in each entry, a value of 1 indicates that the user interacted with the item (e.g., buy, like, or share), and 0 otherwise. The item set is $\{i_a, i_b, i_c, i_d, i_e, i_f, i_g\}$.

		Item						
		i_a	i_b	i_c	i_d	i_e	i_f	i_g
Session	$S_t^{u_A}$	**1**	0	**1**	0	**1**	0	0
	$S_t^{u_B}$	0	**1**	**1**	**1**	0	0	0
	$S_t^{u_C}$	**1**	0	**1**	**1**	0	**1**	0
	$S_t^{u_D}$	**1**	0	**1**	0	0	**1**	**1**
	$S_t^{u_E}$	**1**	0	0	0	0	0	**1**

- **Association Rule Learning (ARL)** [Agrawal, Imieliński, and Arun Swami, 1993]: It is a classic data mining technique used to find current patterns in data. It uses measures such as the support of an item set (how often two items appear together in the action sequences of any user) and its confidence to learn association rules and their corresponding importance. For example, given another user u_F and his/her interaction sequence $\{i_c\}$, ARL predicts that the next item he/she is most interested in is i_a. Because according to the interaction table, the item pair $\{i_a, i_c\}$ has the highest support score, $\frac{3}{5}$, i.e., in 5 interaction sequences, they appear together 3 times ($S_t^{u_A}$, $S_t^{u_C}$ and $S_t^{u_D}$), and the confidence is equal to $\frac{3}{4}$, i.e., the item i_C appears in 4 sequences, 3 times with i_a. This method is easy to implement and relatively explicable for users. However, determining the appropriate support and confidence thresholds is challenging, and the calculation is very time-consuming.

- **K-Nearest Neighbors (KNN)-Based**: This type of methods includes different schemes: *item-based KNN* and *session-based KNN*.

- **Item-based K-Nearest Neighbors (IKNN)**: The most traditional IKNN, such as the one used in [Hidasi, Karatzoglou, et al., 2015; Hidasi and Karatzoglou, 2018; Ludewig and Jannach, 2018], only considers the last item in a given session and then recommends items that are similar to it, where the similarity is measured by the co-occurrence frequency in other sessions. The item is usually encoded as a binary vector whose size is the total number of sessions, where each element corresponds to a session. If the item is present in this session, the corresponding session position is set to "1". The similarity is then calculated by the cosine similarity between the binary vectors. For example, according to the above table, item i_a can be expressed as $[1,0,1,1,1]$ and i_c as $[1,1,1,1,0]$.
- **Session-based K-Nearest Neighbors (SKNN)** [Lerche, Jannach, and Ludewig, 2016; Jannach and Ludewig, 2017]: SKNN compares the entire current session to all past sessions in the training set instead of only considering the last item in the current session. More specifically, given a session S_t^u, we first apply a suitable session similarity measure $\texttt{sim}(S_t^u, S'), S' \in \mathcal{S}$ to determine the K most similar past sessions $\mathcal{S}_{\text{sim}} \subseteq \mathcal{S}$. The similarity measure can be the Jaccard index or cosine similarity of binary vectors on the item space. Then the recommendation score for each item i is defined by [Bonnin and Jannach, 2014], as:

$$\texttt{score}_{\text{SKNN}}(i, S) = \sum_{S' \in \mathcal{S}_{\text{sim}}} \texttt{sim}(S_t^u, S') \cdot \mathbb{1}_{S'}(i),$$

  where $\mathbb{1}_{S'}(i)$ returns 1 if session S' contains item i and 0 otherwise.

Despite their simplicity, KNN-based methods generally perform surprisingly well, as discussed in [Jannach and Ludewig, 2017; Kamehkhosh, Jannach, and Ludewig, 2017], and they can generate highly explainable recommendations. In addition, since the similarity can be pre-calculated, KNN-based recommender systems can generate recommendations quickly. However, such algorithms generally fail to consider the order dependency among items, i.e., when using the Jaccard index or cosine similarity as a similarity measure, SKNN method does not consider the order of items in the session. To address this issue, some variants of SKNN are proposed in [Ludewig and Jannach, 2018]. For example, Vector multiplication Session-based KNN (V-SKNN) is a variant of SKNN. When computing the similarity, this variant focuses more on recent items in a session. Instead of encoding the session as a binary vector as described above, V-SKNN encodes the current session using a real-valued vector. Only the last item in the session has a value of 1, the weights of the other items are determined using a linear decay function that depends on the position of the item within the session. Therefore, items that appear earlier in the session are given lower weight than the last item.

- **Markov Chain (MC)-Based**: Markov Chain based methods treat sequential data as a random process on discrete random variables. They assume that future user behavior depends only on the last or last few behaviors. For example, [He and McAuley, 2016] adopts a L-high-order MC to make recommendations based on the L previous behaviors. MC-based methods cannot exploit dependencies between behaviors in relatively long sequences.

- **Matrix Factorization (MF)-Based**: Matrix Factorization based approaches first factorize the item co-occurrence matrix or item-to-item transition matrix into a latent representation vector for each item and then uses these latent representations to predict the following items. Such approaches should be distinguished from the MF used in collaborative filtering methods, which usually factorize the user-item interaction matrix (e.g., rating matrix) into latent factors of users and items respectively [Linden, Smith, and York, 2003; Su and Khoshgoftaar, 2009].

Deep Learning Based Sequential Recommendation Methods

Recently, DL-based techniques have been massively used in recommender systems with impressive performance [S. Zhang et al., 2019], especially for the sequential recommendation [Fang, Danning Zhang, et al., 2020]. The general idea of DL-based methods is to use different deep network architectures (e.g., Recurrent Neural Network and Convolutional Neural Network) to explore dependencies between sequential behaviors. The architectures are then trained using a variety of well-designed loss functions that allow the models to make reasonable and accurate predictions about possible follow-up actions. Moreover, in order to improve the training efficiency, some models adopt special training techniques, such as the batch generation method and negative sampling strategy. In the following, before describing some specific sequential recommendation models, we first summarize the main training strategies used in DL-based sequential recommendation models from three aspects according to [Fang, Danning Zhang, et al., 2020], including (i) negative sampling strategy, (ii) mini-batch creation, and (iii) loss function design:

Negative Sampling Strategy Because there may be hundreds, thousands, or even millions of items to recommend, it is difficult to calculate scores for all items at each prediction. The negative sampling strategy is proposed, which selects a small number of items that the user has not interacted with as negative samples to speed up model training. The strategies adopted can usually be divided into *popularity-based sampling* and *uniform sampling*.

- **Popularity-based sampling**: It assumes that the more popular an item is, the more likely users will know about it, i.e., users may not like it if they have not interacted with it before. The sampling of negative samples is proportional to their popularity (support).
- **Uniform sampling**: The sampling is proportional to the uniform distribution.
- **Additional sampling (Popularity-based + Uniform)** : It is a combination of *popularity-based sampling* and *uniform sampling* strategies, which is proposed in [Hidasi and Karatzoglou, 2018]. In *additional sampling*, negative samples are selected with a probability proportional to supp_i^α, where supp_i^α is the support for item i and $\alpha, 0 \leq \alpha \leq 1$ is a parameter. The cases of $\alpha = 0$ and $\alpha = 1$ are equivalent to *uniform sampling* and *popularity-based sampling*, respectively.

Mini-Batch Creation The models are all trained in batches. In order to adapt to the particular data structure of sequential recommendation, some batch generation methods are proposed to improve the training efficiency.

- **Session-parallel mini-batch**: A session-parallel mini-batch training strategy is proposed in GRU4Rec [Hidasi, Karatzoglou, et al., 2015] to adapt to sessions of varying lengths and strive to capture the dynamics of sessions over time. More specifically, as shown in Figure 3.12, the sessions are first ordered chronologically, and then all the first events of the first batch of sessions constitute the inputs of the first mini-batch (i.e., the events $i_{1,1}$, $i_{2,1}$ and $i_{3,1}$). The desired outputs (i.e., $i_{1,2}$, $i_{2,2}$ and $i_{3,2}$) are the second events of the active sessions. If any session of the given sessions ends, the next available session out of the given sessions is placed in the corresponding place to form the mini-batch continually, e.g., $i_{4,1}$ follows $i_{2,2}$.

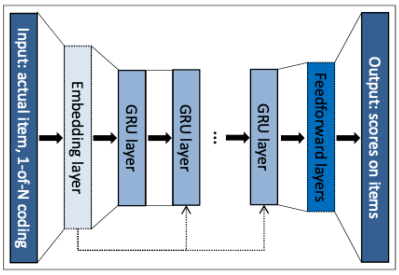


FIGURE 3.11: The GRU4Rec architecture, from [Hidasi, Karatzoglou, et al., 2015].

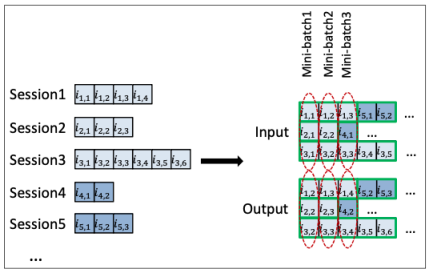


FIGURE 3.12: Session-parallel mini-batch of GRU4Rec, from [Hidasi, Karatzoglou, et al., 2015].

- **User parallel mini-batch**: It is a variant of session-parallel mini-batch [Quadrana, Karatzoglou, et al., 2017] that designs parallel sessions belonging to different users to simulate the evolution of user preferences across sessions.

Loss Function Design The loss function can also greatly affect the model performance. Given a session with $l+1$ interacted items $S_t^u = \{i_1, \ldots, i_l, i_{l+1}\}$, the next-item prediction is to predict i_{l+1} based on the previous items before it, i.e., $\{i_1, \ldots, i_l\}$. The number of negative samples is defined as $|\mathcal{I}^-|$, and the ranking score is denoted as r. The various loss functions are described below:

- **TOP1**: It is a regularized approximation of the relative ranking of positive and negative samples, where the relative ranking is calculated as:

$$\mathcal{L}_{\text{TOP1}} = \frac{1}{|\mathcal{I}^-|} \sum_{i' \in \mathcal{I}^-} \sigma\left(r_{i'} - r_{i_{t+1}}\right) + \sigma\left(r_{i'}^2\right).$$

  The first part is designed to force the scores of negative samples to be around 0, and the second part is used for regularization.

- **Bayesian Personalized Ranking (BPR)**: It is a Matrix Factorization method using pairwise ranking loss. It compares the scores of positive and negative samples, and tries to maximize the probability that the score of the ground-truth item is higher than the negative samples, i.e.,

$\sum_{i' \in \mathcal{I}^-} Pr(r_{i_{t+1}} > r_{i'})$, where $Pr(r_{i_{t+1}} > r_{i'})$ is approximated by $\sigma(r_{i_{t+1}} - r_{i'})$, and use negative log-probability. The final loss function is defined as:

$$\mathcal{L}_{\text{BPR}} = -\frac{1}{|\mathcal{I}^-|} \sum_{i' \in \mathcal{I}^-} \log \sigma \left(r_{i_{t+1}} - r_{i'} \right).$$

As explained in [Fang, Danning Zhang, et al., 2020], as the number of items increases, the gradients of these losses above may suffer from the gradient vanishing problem. In order to address this issue, a new family of listwise loss functions has been proposed, which is based on individual pairwise loss. The idea is to compare the ground-truth score with the most relevant negative sample score, the largest score among the samples, named *ranking-max*. Based on this idea, *TOP1-max* and *BPR-max* losses are formulated as:

- **TOP1-max**: It is pretty straightforward and is given as follows:

$$\mathcal{L}_{\text{TOP1-max}} = \sum_{i' \in \mathcal{I}^-} s_{i'} \left(\sigma \left(r_{i'} - r_{i_{t+1}} \right) + \sigma \left(r_{i'}^2 \right) \right),$$

which can be considered as a weighted version of *TOP1*, weighted by the corresponding softmax score $s_{i'}$. If $r_{i'}$ is much lower than the maximum negative score, $s_{i'}$ will be almost zero.

- **BPR-max**: It tries to maximize the probability:

$$\sum_{i' \in \mathcal{I}^-} Pr(r_{i_{t+1}} > r_{i'} | r_{i'} = r_{\max}) Pr(r_{i'} = r_{\max}),$$

which can be approximated by $\sigma(r_{t+1} - r_{i'})$ and $s_{i'}$ respectively:

$$\mathcal{L}_{\text{BPR-max}} = -\sum_{i' \in \mathcal{I}^-} \log s_{i'} \sigma \left(r_{i_{t+1}} - r_{i'} \right).$$

In addition to the ranking-based loss functions, *Categorical Cross-Entropy* and *Hinge* losses are also applied to sequential recommendations.

- **Categorical Cross-Entropy (CCE)**:

$$\mathcal{L}_{\text{CCE}} = -\log \left(\texttt{softmax}(\boldsymbol{o})_{i_{t+1}} \right),$$

where $\boldsymbol{o}$ is the output of the model. It has computational complexity issues due to the softmax function.

- **Hinge**: It compares the prediction with a pre-defined threshold τ:

$$\mathcal{L}_{\text{Hinge}} = \sum_{i' \in \mathcal{I}_{yes}} \max \left(\tau, 1 - o_{i'} \right) + \gamma \sum_{i'' \in \mathcal{I}_{no}} \max \left(\tau, o_{i''} \right),$$

where $\mathcal{I}_{yes}$ is the recommended item set containing item i_{t+1}, $\mathcal{I}_{no}$ is the recommended item set not containing i_{t+1}, and γ is a parameter that balances the impacts of the two parts. Using

the *Hinge* loss, the recommendation task is transformed into a binary classification problem, where the recommender system determines whether an item should be recommended or not.

Next, we describe some main sequential recommendation approaches according to the architecture type used (e.g., Recurrent Neural Network and Convolutional Neural Network). For an exhaustive survey, readers can refer to [S. Zhang et al., 2019].

RNN-Based Models Recurrent Neural Network (RNN)s with the ability to process sequential data have been used for sequential recommendation and have achieved promising results [Hidasi, Karatzoglou, et al., 2015; Hidasi, Quadrana, et al., 2016; F. Yu et al., 2016]. Compared to traditional models, RNN-based models can capture the dependencies among items within a session or across different sessions well. The basic idea of RNN-based methods is to encode user previous behavior sequences into vectors with various recurrent architectures (e.g., Gated Recurrent Unit (GRU) [Cho et al., 2014] or Long Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997]) and use various loss functions mentioned above to train models.

- **GRU4Rec** [Hidasi, Karatzoglou, et al., 2015]: GRU4Rec is the first work to our knowledge that applies RNN to sequential recommendations. The input is a one-hot encoded representation of the item, and the output is the probabilities for a fixed number of candidate items that will be the next in the session. The core of GRU4Rec is the GRU layer(s) (the GRU architecture is explained in Appendix E.3.2), where the output of each GRU layer is the input of the next GRU layer. A feedforward layer is added between the last GRU layer and the output layer, as illustrated in Figure 3.11. GRU4Rec employs Session-parallel mini-batch (as described above) for efficient model training. The reason for using session-parallel mini-batches is to form sessions with equal length, but the length of active sessions can vary widely. Moreover, it uses a popularity-based negative sampling strategy to select negative samples from other items in the same mini-batch with which the user has not interacted. This strategy assumes that popular items that the user missed are more likely to express dislike than other unknown items (i.e., if the user knows about them, he/she might be interested).

Since GRU4Rec, there has been a series of studies using RNNs for the sequential recommendation. These extended studies [Hidasi, Quadrana, et al., 2016; Tan, X. Xu, and Yong Liu, 2016; J. Li et al., 2017; Quadrana, Karatzoglou, et al., 2017; Hidasi and Karatzoglou, 2018] strive to improve the model performance from the perspective of (i) model training and (ii) designing more advanced model structures for better learning item information or user profile.

- [Hidasi, Quadrana, et al., 2016] extends GRU4Rec by considering additional item information in addition to item IDs (e.g., images and textual descriptions). This information is not obtained end-to-end, i.e., the image features are independently extracted from a pre-trained GoogLeNet implementation [Szegedy et al., 2015] using transfer learning, and the textual descriptions are simply represented by TF-IDF. Specifically, the authors introduce a multi-modal architecture called p-RNNs, which consists of multiple parallel RNNs, each information source (e.g., item ID, image, and textual description) has its own RNN. In addition,the authors propose alternative

training strategies for p-RNNs: *simultaneous*, *alternating*, *residual*, and *interleaving* training. In *simultaneous* training, each parameter of each RNN layer is trained simultaneously. In *alternating* training, RNNs are alternately trained in each period. In *residual* training, RNNs are trained one-by-one with the residual error of the ensemble of previously trained RNNs, and *interleaving* training is performed alternately in each mini-batch.

- [Tan, X. Xu, and Yong Liu, 2016] adopts two techniques to improve the performance of GRU4Rec, including a data augmentation process and a method to account for shifts in the input data distribution. The authors propose two ways to enhance sequences: the first is to treat all prefixes of the original input sessions as new training sequences, and the other is to randomly drop some actions from the training sessions. In addition, they use a lot of outdated data to get a pre-trained model, use this pre-trained model to initialize a new model, and then use the latest data to train the new model. This allows the new model to benefit from large amounts of data while focusing on the latest data.

- [Hidasi and Karatzoglou, 2018] improves GRU4Rec with a new class of loss functions (i.e., BPR-max and TOP1-max) and an improved sampling strategy (a combination of uniform sampling and popularity sampling), which is called Additional sampling, as described above.

CNN-Based Models Convolutional Neural Networks (CNNs) are suitable for capturing dependencies across local information, e.g., the correlation between pixels in a specific part of an image or the dependency between several adjacent words in a sentence. In the sequential recommendation, CNN-based models can capture local features within a session well.

- **Convolutional Sequence Embedding Recommendation Model (Caser)** [Jiaxi Tang and K. Wang, 2018]: Caser is a representative CNN-based method, which consists of three components: *Embedding Look-up*, *Convolutional Layers* and *Fully-connected Layers*, as shown in Figure 3.13.

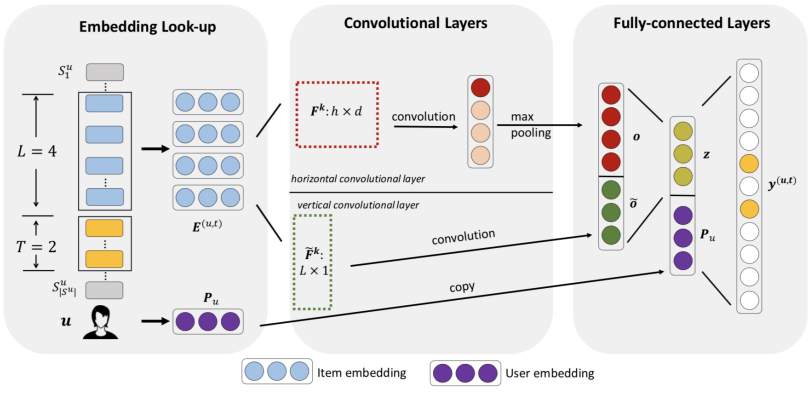


FIGURE 3.13: The Caser architecture, from [Jiaxi Tang and K. Wang, 2018]. Here, the authors use $L = 4$ consecutive items to predict the next $T = 2$ items that user u will interact with.

Specifically, given a sequence of items S^u interacted by user u, the goal of Caser is to predict the next T items based on the L previously interacted items, as shown on the left side of Figure 3.13. It first embeds the user feature and L items into a user embedding $\boldsymbol{P}_u$ (i.e., the purple circle) and an item embedding matrix $\boldsymbol{E}^{(u,t)}$ (i.e., the blue circles) through the *Embedding Look-up* layer. It treats the matrix $\boldsymbol{E}^{(u,t)}$ as an "image" and uses horizontal layers and vertical layers to search for sequential patterns as the local features of this "image", as shown in the middle of Figure 3.13. The outputs of *Convolutional Layers* are fed into *Fully-connected Layers* for more high-level and abstract features. Finally, the final output is used for model training with binary cross-entropy loss.

- **3D-CNN** [Tuan and Phuong, 2017]: 3D-CNN is a similar work to Caser, which combines sequential histories and content features (e.g., item textual description, item category, and item ID) to generate recommendations. It uses a 3-dimensional CNN with character-level encoding to model these data.

Attention-Based Model The attention mechanism is initially proposed in [Bahdanau, Cho, and Bengio, 2014] for the neural machine translation task. The idea is to model the importance of different parts of the input sentence to the output word. Building on this work, *vanilla attention* is proposed, and the attention mechanism is applied as a decoder for RNNs. We provide more details about the attention mechanism in Appendix F.1. The attention mechanism has been widely used in sequential recommendations and can identify more "preference-related" items for users based on their historical experiences. On the other hand, *self-attention*, derived from Transformer [Vaswani et al., 2017], has also been applied to the sequential recommendation. Compared to *vanilla attention*, it does not contain the RNN structure, but performs much better than RNNs-based models. More details on *self-attention* are given in Appendix F.2. The attention mechanism has shown promising potential in improving recommendation performance and interpretability [Kang and McAuley, 2018; J. Li et al., 2017; Q. Liu et al., 2018; Sun et al., 2019]. In the following sections, we summarize some attention-based models according to the type of attention mechanism deployed: (i) *vanilla attention* and (ii) *self-attention*.

- **Vanilla Attention Mechanism**
  - **Neural Attentive Recommendation Machine (NARM)** [J. Li et al., 2017]: NARM is an encoder-decoder framework for sequential recommendation. It considers both the sequential behavior of the user and the main purpose of the current session. Specifically, in the global encoder, a GRU layer is used to model the sequential behavior of users. The local encoder is similar to the global encoder, but an item-level vanilla attention mechanism is combined with a GRU layer to capture the primary purpose of users in the current session. Through the attention mechanism, NARM is able to remove noise from unexpected behaviors, such as accidental actions. In order to learn the model parameters, a cross-entropy loss is used. The loss function can be optimized with the standard mini-batch Stochastic Gradient Descent (SGD).
  - **Short-Term Attention/Memory Priority (STAMP)** [Q. Liu et al., 2018]: STAMP can capture the general interests of the user from the long-term memory of the session context

and take into account the current interests of the user from the short-term memory of the "last action". Therefore, such a session representation can simultaneously capture the long-term general interests of users and their short-term interests. In particular, the general interest is modeled by an attention-based model that considers both the "last action" and the current session.

- **Self-Attention Mechanism**
  - **Self-Attention-based Sequential Recommendation model (SASRec)** [Kang and McAuley, 2018]: SASRec adopts a self-attention layer to balance the short-term intent and long-term preference and seeks to identify items relevant to the next behavior from user historical behavior sequences. Figure 3.14 shows the structure of SASRec.

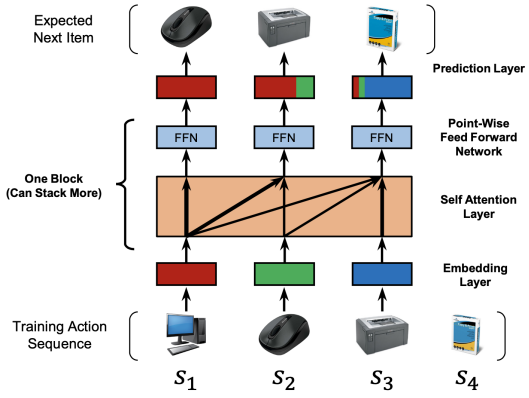


FIGURE 3.14: The SASRec architecture, from [Kang and McAuley, 2018].

Specifically, the first layer of SASRec is an *Embedding layer*, and the input sequence is represented as the concatenation of item embedding and its relative position embedding. The sequence representation is then fed into a stack of self-attention blocks, each with two parts: (i) *Self-Attention Layer* and (ii) *Point-Wise Feed-Forward Network*. However, the training becomes difficult as the network gets "deeper". To alleviate this problem, SASRec uses the residual connection, layer normalization, and dropout on each *Self-Attention Layer* and *Point-Wise Feed-Forward Network*. Another issue to consider is that when recommending the $t+1$-th item, only the previous t items are known. However, the t-th output of *Self-Attention Layer* contains embeddings of the following items. SASRec modifies the attention mechanism by disabling all links between the i-th query and j-th key for $j > i$, which means the corresponding attention weights are set to 0. Finally, the output of the last *Self-attention* block is used for the next-item prediction. The model is trained using pointwise cross-entropy loss.

  - **BERT4Rec** [Sun et al., 2019]: BERT4Rec is another representative self-attention-based method, which introduces a bidirectional self-attention model to model user behavior sequences. Similar to SASRec, the input of the stacked Transformer layer is constructed

by summing the corresponding item and position embeddings. As shown in Figure 3.15, the *Transformer layer* contains two sub-layers: (i) *Multi-Head Self-Attention layer* and (ii) *Position-wise Feed-Forward Network*. The structure of BERT4Rec is illustrated in Figure 3.16.

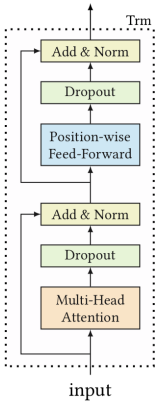


FIGURE 3.15: The Transformer layer used in BERT4Rec, from [Sun et al., 2019].

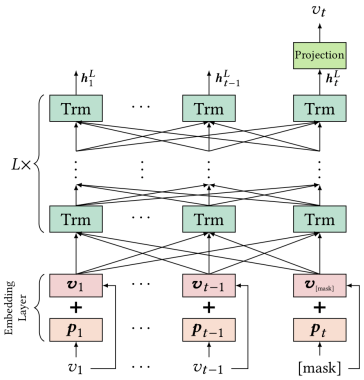


FIGURE 3.16: BERT4Rec architecture, from [Sun et al., 2019].

GNN-Based Models Graph Neural Networks (GNNs) [J. Zhou, G. Cui, Z. Zhang, et al., 2018] can collectively aggregate information from graphs, as described in Section Chapter 3.1. They have also attracted increasing attention in recommender systems due to their effectiveness and excellent performance in many applications [Fang, Danning Zhang, et al., 2020; Shiwen Wu et al., 2022].

- **Session-based Recommendation with Graph Neural Networks (SR-GNN)** [Shu Wu et al., 2019]: SR-GNN is the first work that uses GNN to perform sequential recommendations by capturing more complex relationships between items in the sequence. Each session is represented as a directed graph in this model, which is then processed using a gated GNN [Y. Li et al., 2015] to obtain a session representation. This representation is a combination of the global preference and current interest in this session. Finally, SR-GNN predicts the probability of each item being the next-click in each session. The structure of SR-GNN is showed in Figure 3.17.

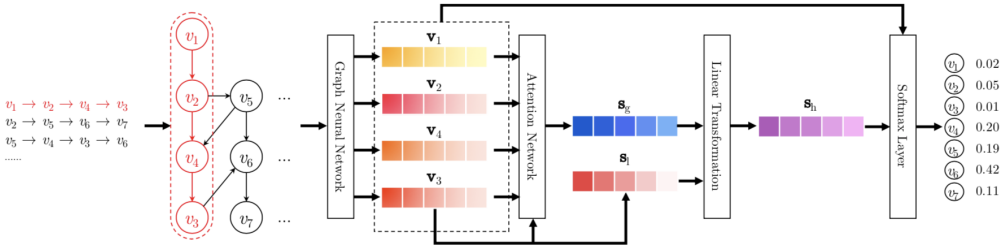


FIGURE 3.17: The SR-GNN architecture, from [Shu Wu et al., 2019].

We summarize the characteristics of the aforementioned deep methods in Table 3.8, which exploit different NN architectures.

TABLE 3.8: Summary of representative DL-based sequential recommendation methods.

Model	Paper	Method
RNN	GRU4Rec [Hidasi, Karatzoglou, et al., 2015]	• session-based mini-batch • popularity-based negative sampling
	p-RNNs [Hidasi, Quadrana, et al., 2016]	• additional item information (images, textual descriptions) • multiple parallel RNNs
	[Tan, X. Xu, and Yong Liu, 2016]	• training data augmentation • data distribution shift (pre-training)
	[Hidasi and Karatzoglou, 2018]	• BPR-max and TOP1-max losses • additional sampling
CNN	Caser [Jiaxi Tang and K. Wang, 2018]	• horizontal and vertical convolutional layers
	3D-CNN [Tuan and Phuong, 2017]	• additional item information (textual description, category) • 3-dimensional CNN with character-level encoding
Attention	NARM [J. Li et al., 2017]	• GRU encoder to model the sequential behavior • GRU encoder with vanilla attention to model the primary purpose in the current session
	STAMP [Q. Liu et al., 2018]	• use attention mechanism to capture both the long-term general interests and the short-term interests
	SASRec [Kang and McAuley, 2018] BERT4Rec [Sun et al., 2019]	• use self-attention to balance the short-term intent and long-term preference • use Transformer
GNN	SR-GNN [Shu Wu et al., 2019]	• model sessions as a directed graph

Evaluation Metrics

Metrics widely used in sequential recommendation include: the accuracy metric Recall@K, also known as Hit Rate (HR@K) [Ludewig and Jannach, 2018], and the two most popular ranking metrics are Mean Reciprocal Rank (MPR@K) [Ludewig and Jannach, 2018] and Normalized Discounted Cumulative Gain (NDCG@K) [Fang, G. Guo, et al., 2019], where K is the cut off value.

- **Recall@K**: It measures the coverage of correctly recommended items based on ground-truth items. If there is only one item in the ground-truth item list, Recall@K is identical to HR@K. In this case, HR@K checks if the true next-item (ground-truth) exists in the top-K ranked items, defined as follows:

$$HR@K = \frac{1}{|\mathcal{S}|} \sum_{S \in \mathcal{S}} \mathbb{1}\left(r_{i^+} \leq K\right),$$

where $\mathcal{S}$ is a sample set of sequences used for evaluation, i^+ is the true next-item of each sequence S, and r is the ranking generated by the recommender system. The symbol $\mathbb{1}$ is an indicator that returns 1 if r_{i^+} is in the top-K recommended items, 0 otherwise.

- **Mean Reciprocal Rank (MRR@K)**: It indicates how well the system ranks the ground-truth items and is sensitive to their position, which assigns higher scores to the top rankings. Intuitively, it is better to rank the ground-truth items higher in practice. MRR@K is defined as follows:

$$MRR@K = \frac{1}{|\mathcal{S}|} \sum_{S \in \mathcal{S}} \begin{cases} \frac{1}{r_{i^+}} & \text{if } r_{i^+} \leq K \\ 0 & \text{otherwise} \end{cases}.$$

- **Normalized Discounted Cumulative Gain (NDCG@K)**: It rewards each ground-truth item based on its position in the list of recommended items, indicating how strongly it is recommended.

$$NDCG@K = \begin{cases} \frac{1}{\log_2(r_{i^+}+1)} & \text{if } r_{i^+} \leq K \\ 0 & \text{otherwise} \end{cases}.$$

Summary of Chapter 3

This chapter serves as the preface of this thesis, providing necessary background knowledge for the subsequent chapters. Specifically, in Section 3.1, we introduced graph embedding and detailed some representative models. These models are used as baselines in two tasks: **Job Title Representation Learning from Graphs** (Chapter 4) and **Skill Representation Learning by Leveraging Hierarchical Graph** (5). Then, in Section 3.2, we introduced the background of the recommender system and mainly detailed some sequential recommendation methods. Similarly, these methods are used as baselines in the task **Next-Application Prediction from Job Application Sequence**(Chapter 6).

Chapter 4

Job Title Representation Learning from Graphs

4.1 Motivation

Job titles (referred to as jobs for short)[1] usually use a few words to describe the position held by talents (i.e., in resumes) or published by employers (i.e., in job postings), combined with responsibilities, functions, and additional information. An illustration of job titles is given in Figure 4.1, where job titles appear in the *Work Experience* section of resumes and at the top of job postings.

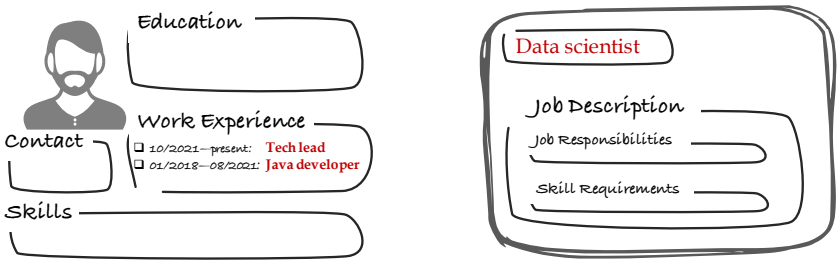


FIGURE 4.1: The scene where the job title appears.

Learning job title representation is conducive to various downstream tasks in the recruitment domain, such as (i) improving talent profile modeling to provide career guidance, (ii) categorizing jobs to facilitate search and recommendation, and (iii) aiding recruitment analysis. However, in practice, learning the proper job title representation is a challenging task for the following reasons:

- **Noisy data**: There is some noise in the job title data due to personal subjective reasons (i.e., spelling errors) or objective reasons (i.e., the resume parser is not perfect). For example, "software engineer" has a spelling error.
- **Messy data**: Job titles are messy because people have different ways of thinking and different understandings of words and ideas (related to **Characteristic 2**). For example, there are too many alternative words for a job position so that the same job position may be expressed by different job titles, e.g., "purchasing clerk" and "buyer", or the same term may refer to different

[1] In this chapter, we use *job title* and *job* interchangeably.

job positions in different contexts, e.g., the term *sandwich* in "registered nurses sandwich rehab" and "sandwich maker".

- **Non-standard naming conventions**: Naming conventions vary greatly from company to company and across industries (related to **Characteristic 3**). They usually define their naming conventions and occupational taxonomies and use these reference materials and standards to write job postings, manage human resource systems and recruit talents. Therefore, there are a lot of subjective and non-standard naming conventions, which brings many heterogeneity issues.

In the literature, job title representation learning, as described in Section 2.3.1, can be categorized into (i) semantic-based and (ii) graph-based. Traditional semantic-based methods mainly focus on improving the semantic representation of job titles, where the job title representation is a simple combination (e.g., mean or sum) of the word semantic representations. For the reasons mentioned above, simply combining the representation of words may increase the possibility of mismatch. Moreover, they ignore the hidden relationships between job titles, e.g., two job titles on the same talent resume may be very similar. Recently, [Dave et al., 2018; Denghui Zhang et al., 2019] learn representations from the perspective of graphs. They create graphs, mainly homogeneous graphs, from job transition trajectories, where nodes represent job titles and edges represent job transitions. Then they design different loss functions to embed the nodes into a low-dimensional space to learn job title representations. We describe the main graph-based approaches for job title representation learning in Section 4.3. However, the generated graphs are usually sparse due to the above reasons, limiting the performance of graph-based methods. Standardizing job titles before generating graphs can alleviate the sparse issue to a certain extent, but it loses some specific semantic information.

To tackle these challenges, in this work, our hypothesis is that such job title representation learning from graphs could benefit from richer additional information about job titles. We propose to enrich the job transition graph with additional information and learn job title representations via enriched graph embedding methods. Specifically, inspired by domain-specific Named Entity tags (i.e., *RES*ponsibility and *FUN*ction) proposed in [Junhua Liu et al., 2019], we treat the job title as a combination of *responsibility*, *functionality*, and *other additional information*. The *responsibility* part describes the responsibility-level of the position, e.g., senior, director and manager, the *functionality* part describes the core function/operation type, e.g., marketing, security and education, and the *additional information* contains some personal-specific information, such as company name and geographic location. Accordingly, words related to *responsibility* and *functionality* are defined as *RES*- and *FUN*-tags, respectively. Along this line, we construct a heterogeneous *Job-Transition-Tag Graph*, containing two types of nodes, i.e., job titles and tags (*RES*-tags and *FUN*-tags), and two types of edges, i.e., job transitions and "has/in" relationships. Then, we apply different graph embedding models on the graph to learn job title representations. More details on graph definition are explained in Section 4.4.2.

Contributions In summary, our main contributions in this chapter are:

- We learn job title representations by simultaneously considering **topological and semantic information**.

- We propose to **enrich** the constructed graphs with recruitment-specific information, i.e., *RES*-tags and *FUN*-tags, thus **improving the effectiveness of job title representation learning** .
- Our work is **the first** to directly apply such information to the job title representation learning task.
- Experiments show that learned job title representations can **improve downstream tasks such as job classification and next-job prediction**.

4.2 Research Scope

Graph embedding methods have demonstrated their effectiveness in learning node representations, as described in Section 3.1. In particular, the career trajectory of talents can be transformed into a job transition graph with job titles as nodes. In this context, graph embedding methods can be an interesting way to improve job title representation learning, i.e., learn node representations from the job transition graph. Therefore, in this chapter, we explore the utility of graph embedding methods in the task of job title representation learning.

This chapter addresses the following Research Questions (RQs):

- **RQ1**: Can the graph structure provide more useful information for job title representation learning?
- **RQ2**: Does additional information resulting in complex heterogeneous graph help learn better job title representations?

For facilitating illustration, we list some important mathematical notations used throughout this chapter in Table 4.1, unless particularly specified.

4.3 Learning from Job-Transition Graph: an Overview

Many existing graph-based methods mainly learn job title representations from homogeneous *Job-Transition Graph*. The typical procedure is to first build a *Job-Transition Graph* from the working history of talents and then learn job title representations from the resulting graph using different approaches, which we introduce below. Formally, we consider a job seeker set $\mathcal{U}$ and their working history set $\mathcal{H} = \{H^u\}_{u \in \mathcal{U}}$, where the working history of each u is represented as a sequence of n work records ordered by time $H^u = \{J_1, \ldots, J_{|H^u|}\}$. The i-th record J_i is denoted by (j_i, p_i, o_i), indicating that u is engaged in a position (titled j_i) during p_i period. o_i represents other information related to this record, like company name and job content. The set of job titles j_i that occurred in $\mathcal{H}$ is denoted as $\mathcal{J}$. Based on $\mathcal{H}$, *Job-Transition Graph* can be constructed according to the following definition:

Definition 4.3.1 Job-Transition Graph: *is a directed homogeneous graph* $G^{jj} = (\mathcal{J}, \mathcal{E}^{jj}, \mathcal{T}_V, \mathcal{T}_E)$ *generated from* $\mathcal{H}$*, where* $\mathcal{J}$ *is the set of job titles occurring in* $\mathcal{H}$ *and the edge* $e^{jj}_{xy} \in \mathcal{E}^{jj}$ *represents the job transition from the former job* j_x *to the next job* j_y*. Here,* $\mathcal{T}_V = \{$*"job title"* $\}$ *and* $\mathcal{T}_E = \{$*"is the previous job of"* $\}$*, both nodes and edges have only one type.*

TABLE 4.1: Mathematical notations used in *Job Title Representation Learning from Graphs* chapter.

Notation	Description
$\mathcal{U}$	job seeker set
$\mathcal{J}$	job title set
$\mathcal{Q}$	tag set
$\mathcal{C}$	company set
$\mathcal{W}$	vocabulary set (words in job titles)
$\mathcal{H} = \{H^u\}_{u \in \mathcal{U}}$	working history set
$H^u = \{J_1, \ldots, J_{\lvert H^u \rvert}\}$	working history of the job seeker u
$J_i = (j_i, p_i, o_i)$	working record indicating the job title j_i and period p_i
G	graph/network
$\mathcal{V}, \mathcal{E}$	node set, edge (link) set
$\mathcal{T}_V, \mathcal{T}_E$	node type set and edge type set
$\texttt{weight}_{xy}$	weight of edge e_{xy}
$\mathcal{Y}$	node category set
G^{jj}	Job-Transition Graph
$\mathcal{E}^{jj}$	job transition set
G_E^{jj}	Enhanced Job-Transition Graph
$\mathcal{E}_E^{jj}$	enhanced edge set
G^{jt}	Job-Tag Graph
$\mathcal{E}^{jt}$	set of edges between a job title and a tag
G^{jtj}	Job-Transition-Tag Graph
$\Phi(\cdot)$	embedding mapping function
$\mathcal{N}(v)$	neighbors of node v

Example 4.3.1 : *Figure 4.2 describes the Job-Transition Graph built from working histories of four talents, where grey circles represent job titles, and black directed edges represent job transitions among job titles. We formally define edges like* $\langle j_x =$ *"purchase agent"*, $j_y =$ *"purchasing manager"*, $T_E =$ *'is the previous job of'*$\rangle$*, where* $T_E \in \mathcal{T}_E$ *is the edge type.*

Talent A automotive technician → automotive shop manager.

Talent B purchase agent → purchasing manager → staff account purchasing manager.

Talent C staff accountant → staff account purchasing manager → purchasing manager.

Talent D customer service → telemarketer → purchasing clerk.

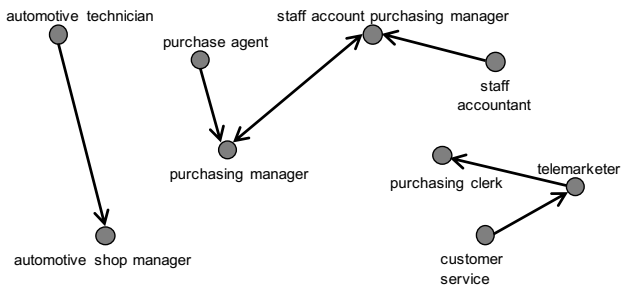


FIGURE 4.2: *Job-Transition Graph* built from working histories of four talents.

Job title representation learning has received much attention in the recruitment field. The learned representations are used in various downstream tasks, such as job recommendation [Dave et al., 2018; M. Liu et al., 2019], job title benchmarking [Denghui Zhang et al., 2019] (i.e., matching job titles with similar expertise levels across various companies, modeled as a link prediction task over the *Job-Transition Graph*), or job mobility prediction [L. Zhang et al., 2021]. These works use different methods, but they are all based on the construction of a *Job-Transition Graph*, defined in Definition 4.3.1.

For example, to better capture the relations between skills and jobs, [Dave et al., 2018] proposes to build three graphs, a homogeneous *Job-Transition Graph* G^{jj} and two other graphs (i.e., job-skill graph and skill co-occurrence graph) based on the jobs and skills present in a set of talent resumes. Then, job title and skill representations are jointly learned from these three graphs. More specifically, the authors capture job transition information from *Job-Transition Graph*, arguing that connected nodes are more similar than non-connected nodes. Given a triple $(j_x, j_y, j_z), j_x, j_y, j_z \in \mathcal{J}$, where the job title j_x is linked to j_y (i.e., $e_{xy} \in \mathcal{E}^{jj}$), while j_x is not linked to j_z (i.e., $e_{xz} \notin \mathcal{E}^{jj}$), the authors want to maximize the probability that the affinity score between j_x and j_y is higher than j_x and j_z. The corresponding loss function is:

$$\mathcal{L} = \min_{\mathbf{\Phi}} - \sum_{(j_x, j_y, j_z) \in \mathcal{D}^{jj}} \ln \sigma \left(\langle \Phi(j_x), \Phi(j_y) \rangle - \langle \Phi(j_x), \Phi(j_z) \rangle \right).$$

$\langle \Phi(j_x), \Phi(j_y) \rangle$ is the affinity score between j_x and j_y, which is the dot product of their representation vectors. The job title representation vector is denoted as $\Phi(\cdot)$, which will be learned during training. $\mathcal{D}^{jj}$ is a set of training triples independently sampled from G^{jj}, and $\sigma(\cdot)$ is a sigmoid function. Similar ideas can be applied to the other two graphs, and since we focus on learning job title representations from the *Job-Transition Graph*, we will not describe operations on the other two graphs.

Job2Vec [Denghui Zhang et al., 2019] proposes to construct a *Job-Transition Graph* $G^{jj} = (\mathcal{V}, \mathcal{E}^{jj})$, where the node $v \in \mathcal{V}$ denotes the job title $j \in \mathcal{J}$ affiliated with the specific company $c \in \mathcal{C}$, i.e., $v = (j, c)$. Then, a collective multi-view (i.e., graph topology, semantic, job transition balance, and job transition duration views) representation learning method is proposed to learn job title representations. Among these views, Job2Vec models the *graph topology view* with a loss function similar to the second-order proximity in LINE [Jian Tang, Qu, et al., 2015], which is formulated as:

$$\mathcal{L}_{topology} = - \sum_{e_{x,y} \in \mathcal{E}^{jj}} \texttt{weight}_{xy} \log \left(\frac{\exp\left(\Phi'(j_y)^T \Phi(j_x)\right)}{\sum_{j \in \mathcal{J}} \exp\left(\Phi'(j)^T \Phi(j_x)\right)} \right),$$

where $\Phi(j_x)$ is the representation vector in the topology-view when j_x is treated as the source node, while when j_x is treated as the "context" of other nodes, its representation is denoted as $\Phi'(j_x)$. The *semantic view* is explored from the shared keywords in job titles, with the idea that job titles with similar keywords should be close to each other in the semantic view representation space. More specifically, the authors first assign to each word w in the job title vocabulary set $\mathcal{W}$, a semantic vector $\boldsymbol{w}$. They then enforce job title representations to be close to each other if they share similar

words, based on the following loss function:

$$\mathcal{L}_{semantic} = \sum_{w_i \in j_x} f_{ix} \log \left(\frac{\exp\left(\boldsymbol{w}_i^T \boldsymbol{j}_x\right)}{\sum_{w \in \mathcal{W}} \exp\left(\boldsymbol{w}^T \boldsymbol{j}_x\right)} \right),$$

where $\boldsymbol{j}_x$ is the semantic representation of job title j_x and f_{ix} is the frequency of the word w_i occurred in the job title j_x. To incorporate more information into job title representations, Job2Vec also considers the number and duration of transitions. The final position representation is obtained by compressing the representations of all views (e.g., $\Phi(j_x)$ and $\boldsymbol{j}_x$) into a denser representation using an encoder-decoder structure.

[L. Zhang et al., 2021] proposes to add company nodes $\mathcal{C}$ in *Job-Transition Graph* to build a heterogeneous graph $G^{jj} = (\mathcal{J} \cup \mathcal{C}, \mathcal{E}^{jj} \cup \mathcal{E}^{cc} \cup \mathcal{E}^{jc})$. Accordingly, the edges contain different node relationships (i.e., the job transition between two company or job title nodes and the belonging relationship between the company node and job title node). Then they use the attentive heterogeneous graph neural network to represent the company and job title nodes. More specifically, the authors define two kinds of aggregators. The first one is an external aggregator, which is used to aggregate information from different types of nodes. For example, given a node v_x (job title or company), the representation $\boldsymbol{a}_{v_x}$ aggregated by external aggregator is expressed as:

$$\boldsymbol{a}_{v_x}^{(l)} = \sum_{v_y \in \mathcal{N}_D(v_x)} \frac{1}{\sqrt{|\mathcal{N}_D(v_y)|}\sqrt{|\mathcal{N}_D(v_x)|}} \boldsymbol{W}^{(l)} \boldsymbol{H}_{v_x}^{(l)},$$

where $\mathcal{N}_D(v_x)$ is the set of neighbors for v_x with different types, $\boldsymbol{W}^{(l)}$ is a weight matrix in (l)-layer, and $\boldsymbol{H}_{v_x}^{(l)}$ is the hidden representation of v_x in (l)-layer. Note that in this part, only the company-job edges are considered. The second aggregator is an internal aggregator that aggregates information from nodes of the same type, i.e., job-job and company-company. Similar to Job2Vec, [L. Zhang et al., 2021] further aggregates the job transition features (i.e., the number of job transitions and the average working duration) into the learned representation via a transition-aware attention mechanism. The representation $\boldsymbol{b}_{v_x}$ aggregated by internal aggregator is expressed as:

$$\boldsymbol{b}_{v_x}^{(l)} = \sum_{v_y \in \mathcal{N}_I(v_x)} \frac{\exp\left(\boldsymbol{\mu}_1 \cdot \left(\boldsymbol{W}_1 \boldsymbol{H}_{v_x}^{(l)} + \boldsymbol{W}_2 \boldsymbol{H}_{v_y}^{(l)} + \boldsymbol{W}_3 \boldsymbol{r}_{xy}\right)\right)}{\sum_{z \in \mathcal{N}_I(v_x)} \exp\left(\boldsymbol{\mu}_2 \cdot \left(\boldsymbol{W}_1 \boldsymbol{H}_{v_x}^{(l)} + \boldsymbol{W}_2 \boldsymbol{H}_{v_y}^{(l)} + \boldsymbol{W}_3 \boldsymbol{r}_{xy}\right)\right)} \boldsymbol{H}_{v_x}^{(l)}$$

$\mathcal{N}_I(v_x)$ is the set of neighbors for v_x with the same type, $\boldsymbol{W}_*$ are trainable weight matrices, and $\boldsymbol{\mu}_*$ are the attention vectors. $\boldsymbol{r}_{xy}$ is the transition feature of edge e_{xy}, which is modeled using the number of job transitions and the average working duration from v_x to v_y. The final job title representation is obtained by fusing the external representation $\boldsymbol{a}_{v_x}$ and internal representation $\boldsymbol{b}_{v_x}$.

JAMES [Yamashita et al., 2022] proposes a multi-aspect co-attention mechanism to fuse three independent embeddings of the job title, namely, topological, semantic, and syntactic embeddings. More specifically, the authors first learn the topological embedding by applying hyperbolic graph representation learning on the constructed job transition graph and obtain the semantic embedding using the pre-trained BERT. Because the syntactic embedding is learned for the final task of this

work (i.e., mapping user-created job titles to predefined and standard job titles), we do not provide details. Finally, similar to [Denghui Zhang et al., 2019; L. Zhang et al., 2021], the three independent embeddings are fused together to obtain the final job title representation.

As mentioned in Section 4.1, the job title and job transition data are noisy and messy. Therefore, *Job-Transition Graph* may be extremely sparse, as emphasized in the Job2Vec approach [Denghui Zhang et al., 2019]. In order to alleviate this issue, a simple and straightforward method is to standardize job titles and then construct a normalized and denser graph based on the standardized job titles. For example, before constructing its *Job-Transition Graph*, [Dave et al., 2018] normalizes job titles by using Carotene, a private job title classification system [Javed, Q. Luo, et al., 2015]. [Denghui Zhang et al., 2019] aggregates job titles by filtering out low-frequency words, and [L. Zhang et al., 2021] unifies job titles according to the Industrial and Professional Occupation Dataset (IPOD) [Junhua Liu et al., 2019] dataset, which contains 192k job titles belonging to 56k LinkedIn users, and each word of these job titles is manually associated with a Named Entity (NE) tag indicating its level of seniority, field of work, and location. Another method is to consider semantic information in addition to graph topology information. For instance, [Denghui Zhang et al., 2019] enforces job title representations to be close to each other if they share similar words. However, these mentioned methods either ignore the semantic information contained in job titles [Dave et al., 2018; L. Zhang et al., 2021] or separate the semantic information from the topology structure [Denghui Zhang et al., 2019]. In addition, the standardization of job titles may lose some specific information.

For better comparison, we summarize the methods explained above in Table 4.2.

TABLE 4.2: A summary of the above graph-based job title representation learning methods. *HetG* denotes the heterogeneous graph, while *HomG* denotes the homogeneous graph.

Paper	Graph	Scheme	Normalization	Topology	Semantic	Others
[Dave et al., 2018]	Job-Transition Graph (HomG), job-skill graph (HetG) and skill co-occurrence graph (HomG)	joint learning	use Carotene	ranking-based loss	not consider	-
[Denghui Zhang et al., 2019]	Job-Transition Graph (HomG)	multi-view	filter out low-frequency words	second-order proximity	conditional probability	number and duration of transitions
[L. Zhang et al., 2021]	Job-Transition Graph (HetG, with company nodes)	multi-view	based on IPOD	attentive heterogeneous GNN	not consider	number and duration of transitions
[Yamashita et al., 2022]	Job-Transition Graph (HomG)	multi-view	-	hyperbolic graph embedding	BERT	syntactic information

As can be seen from the table, all methods use normalized job titles to build graphs, and these methods use different approaches to learn job title representations from the graph topology. Only [Denghui Zhang et al., 2019] models the semantic information, but semantics is learned separately and then fused with topological information. Our proposed scheme differs from these methods in that we learn topological and semantic information simultaneously. We incorporate job title semantic information into the graph topology when constructing graphs. Furthermore, our method does not need to normalize job titles before constructing graphs, thus reducing the information loss. In the next section, we will explain the inspiration for our learning scheme and the details of our method.

4.4 Our Method: Integrating Job Knowledge to Enrich Representations

4.4.1 Job Title Composition

Generally speaking, a job title usually consists of three parts [Junhua Liu et al., 2019; Denghui Zhang et al., 2019]:

- **Responsibility**: describes the roles and responsibilities of a position from different levels, such as:
  - managerial level (e.g., director, manager, and lead),
  - seniority level (e.g., vice, assistant, and associate),
  - operational level (e.g., engineer, accountant, and technician).
- **Functionality**: describes the business functions of a position from various dimensions, such as:
  - work department (e.g., sales, marketing, and operations),
  - work scope (e.g., enterprise, project, and national),
  - work content (e.g., data, automotive, and security).
- **Additional Information**: contains personal-specific information, such as company name and geographic location.

These three distinct parts are the essence of the job title and provide important information about the position. For example, job titles with the same responsibility or functionality are more likely to describe the same level of ability/authority or belong to the same industry. However, such useful information is rarely directly used to learn job title representations, except [Denghui Zhang et al., 2019; L. Zhang et al., 2021] that use it as auxiliary information to normalize job titles in working histories and thus build graphs from these normalized job titles. Unlike these works, we add these words related to responsibility or functionality directly to the graph, which can be seen as a kind of semantic enrichment. As a result, the *Job-Transition Graph* we construct becomes a heterogeneous graph with different types of nodes and edges, which can alleviate the graph sparsity problem and provide additional information for the task of job title representation learning. Furthermore, we simultaneously learn graph topology and semantic information from the same graph instead of combining different views learned in separate graphs like [Denghui Zhang et al., 2019]. A detailed description of graph construction is given in the next section. Extensive experiments have proved that the added words can provide richer information, thereby improving the quality of job title representation.

4.4.2 Methodology

In order to address the sparsity issue of *Job-Transition Graph* mentioned above, we consider adding more information when generating graphs, i.e., words related to the job responsibility or functionality,

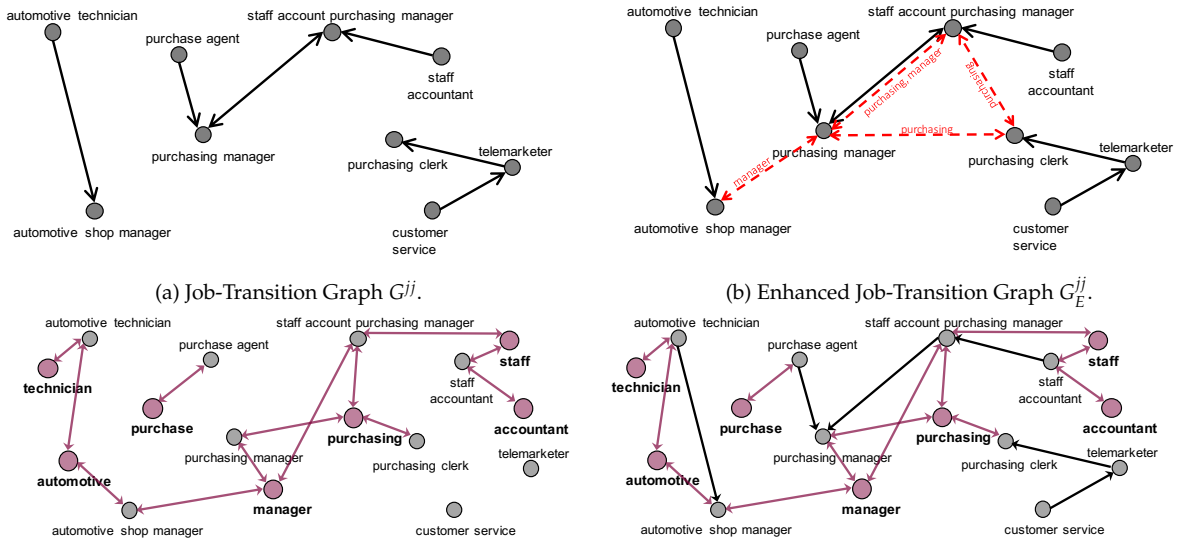
driven by the composition of job titles. Along this line, we formally denote these words as tags, forming a tag set $\mathcal{Q}$, and we define various types of graphs as follows:

Definition 4.4.1 Enhanced Job-Transition Graph: *is based on G^{jj} with additional enhanced edges. It is defined as $G_E^{jj} = (\mathcal{J}, \mathcal{E}^{jj} \cup \mathcal{E}_E^{jj}, \mathcal{T}_V, \mathcal{T}_E)$, where $\mathcal{E}_E^{jj}$ is a set of enhanced edges. More specifically, if j_x and j_y share a word w, then we add a bidirectional edge between them, i.e., e_{xy}^{jj} and e_{yx}^{jj}. The common word w can belong to $\mathcal{Q}$, or other predefined vocabulary sets.*

As shown in Figure 4.3b, red dashed lines represent additional enhanced edges, e.g., "purchasing manager" shares the tag "purchasing" with "purchasing clerk", so we add edges between them. Note that the *Enhanced Job-Transition Graph* can be either homogeneous or heterogeneous, depending on how we define the edge type of the red dashed line. If the edge type is expressed as "share similar tag", it is heterogeneous, with one node type (i.e., job title) and two types of edges (i.e., job transition and "share similar tag" relationship). While if homogeneous, it is similar to the *Job-Transition Graph*, with directed job transition edges and bi-directional enhanced edges. In this work, we consider that *Enhanced Job-Transition Graph* is homogeneous. The heterogeneous case is left for future work.

Definition 4.4.2 Job-Tag Graph: *is a heterogeneous graph $G^{jt} = (\mathcal{J} \cup \mathcal{Q}, \mathcal{E}^{jt}, \mathcal{T}_V, \mathcal{T}_E)$, with job titles and tags, two node types, i.e., $\mathcal{T}_V = \{$"job title", "tag"$\}$. $\mathcal{E}^{jt}$ is a set of bidirectional edges between a job title and a tag, representing the "has/in" relationship, i.e., $\mathcal{T}_E = \{$"has/in"$\}$.*

An example of *Job-Tag Graph* is given in Figure 4.3c, where the grey/purple circles represent job titles/tags, respectively. Edges are bidirectional to indicate that tags are in titles, e.g., job title "automotive technician" has a tag "automotive", so the bidirectional edge e^{jt} (the purple line) means that "automotive technician" *has* the tag "automotive", and "automotive" is *in* "automotive technician".



(a) Job-Transition Graph G^{jj}.

(b) Enhanced Job-Transition Graph G_E^{jj}.

(c) Job-Tag Graph G^{jt}.

(d) Job-Transition-Tag Graph G^{jtj}.

FIGURE 4.3: Examples of four types of graphs (better to view in color), where small grey circles represent job titles, and purple circles are tags. The black lines represent job transitions, red dotted lines represent additional enhanced edges added when job titles share a word, and purple lines represent "has/in" relationships between a job title and a tag.

In order to aggregate more information, we further combine *Job-Transition Graph* and *Job-Tag Graph* to build *Job-Transition-Tag Graph*:[2]

Definition 4.4.3 Job-Transition-Tag Graph: *is a heterogeneous graph* $G^{jtj} = (\mathcal{J} \cup \mathcal{Q}, \mathcal{E}^{jj} \cup \mathcal{E}^{jt}, \mathcal{T}_V, \mathcal{T}_E)$, *with two node types, i.e.,* $\mathcal{T}_V = \{$*"job title"*, *"tag"*$\}$, *and two edge types, i.e.,* $\mathcal{T}_E = \{$*"is the previous job of"*, *"has/in"*$\}$.

Our method then consists of learning job title representations (i.e., node embeddings) from these graphs using the following hypothesis:

1. Job seekers generally work in a specific occupation, so the job titles in the same resume should have strong correlations, e.g., "telemarketer" and "purchasing clerk" given in the same job seeker profile, indicating that the job seeker works in the sales field. Such information is expressed in the graph as job transitions. That is, the transition between "similar" jobs is more likely to occur than non-similar jobs. More precisely, this means that the representation learning scheme should respect the topology of the graph, i.e., adjacent job title nodes in the graph should be close in the embedding space.

2. Job titles with the same tag describe similar job functions or responsibilities. For example, "**automotive** service technician" and "**automotive** shop manager" are two job titles in the automotive domain, and "automotive shop **manager**" and "purchasing **manager**" are two titles for manager-level positions. This hypothesis implies that we want to preserve the semantic information between jobs in the learning scheme. More precisely, the idea is to embed the relationship between job titles linked to the same tag node into the embedding space, and the modeling of the relationship is guided by meta-paths. The definition of meta-path is given in Section 3.1.11.

Inspired by the achievements of graph embedding models in the node representation learning problem [Hamilton, Ying, and Leskovec, 2017a], we benchmark different graph embedding models and adapt them to these different types of graphs in order to learn job representations from these graphs. These methods include homogeneous unsupervised and supervised methods, e.g., Node2Vec, GCN and GAT, as well as heterogeneous methods that consider node and edge types, e.g., metapath2vec, RGCN and HAN. For unsupervised methods, the learning objective is to maximize the different proximity (i.e., first-, second-, or high-order) of nodes preserved in the embedding space, while for supervised methods, job title representations are learned through a job title classification objective or a link prediction objective. A detailed description of these methods is given in Section 3.1.3 and in Section 4.5.3.

4.5 Application on Two Real Recruitment Datasets

Node classification and link prediction are two tasks that are widely used to evaluate the quality of node representation. Therefore, this section empirically evaluates our proposed graphs on two real-world networks with (i) a node classification task (i.e., *job title classification*) and (ii) a link prediction

[2] In further work, we consider trying different combinations, for example, combining *Enhanced Job-Transition Graph* with *Job-Tag Graph*, that is, adding enhanced edges $\mathcal{E}_E^{jj}$ to *Job-Transition-Tag Graph*.

task (i.e., *next-job prediction*). We first describe the datasets used and experimental settings. Then we present the quantitative results and analyze our graphs in more detail.

4.5.1 Datasets

In this work, we employ a public dataset *CareerBuilder12* (*CB12*) and a private dataset *Randstad* to evaluate our proposed graphs.

CareerBuilder12 It is an open dataset from an open Kaggle competition[3] provided by the online recruitment site CareerBuilder.[4] It contains a collection of working experiences represented by sequences of job titles. In total, we have 375,531 experience records (talents) and 657,153 job titles. In the pre-processing process, we tokenized titles into tokens, then removed stopwords, numbers, and punctuation. Then we use an online third-party API *O*Net-SOC AutoCoder*[5] to assign a Standard Occupation Classification (SOC) code 2018 to each job title. The SOC taxonomy consists of four levels, *Major Group* (9), *Sub-Major Group* (25), *Minor Group* (90) and *Unit Group* (369), an example is shown in Figure 4.4. For a detailed introduction to SOC, see Appendix B.2.2. Given a job title, the original repository assigns a 3-digit *Minor Group* code to the job title. For more details on the labeling process and AutoCoder, please refer to Appendix B.1.1 and Appendix B.3.1. In the end, 22,590 labeled job titles were obtained.

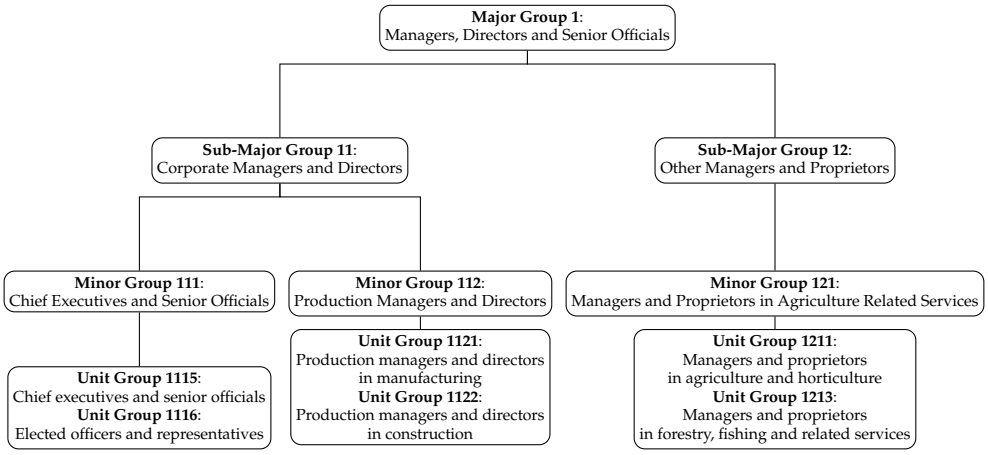


FIGURE 4.4: An example of SOC structure.

Randstad It is a private resume dataset provided by the talent pool of Randstad company[6], where each resume is in French and is parsed into sections, e.g., *Personal EducationHistory* and *EmploymentHistory*, etc., as shown in Figure 4.5. More statistics for this resume dataset are given in

[3] https://www.kaggle.com/c/job-recommendation

[4] https://www.careerbuilder.com/

[5] http://www.onetsocautocoder.com/plus/onetmatch

[6] https://www.randstad.com/

Appendix B.1.2. Graphs are built from *EmploymentHistory* section, which consists of multiple *EmploymentItem*s, each *EmploymentItem* containing a *JobTitle*, and its corresponding occupation labels (i.e., *JobCode*, *JobGroup* and *JobClass*).

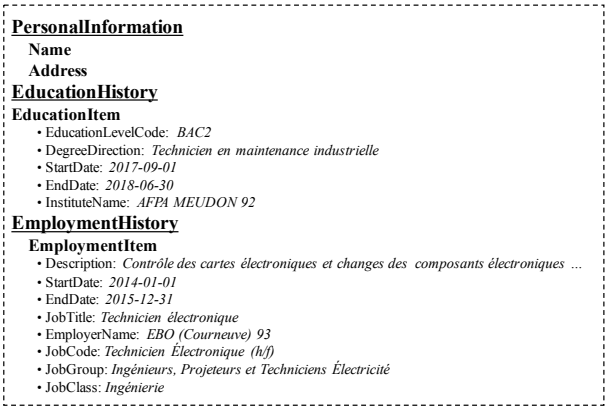


FIGURE 4.5: An example of parsed resume in Randstad dataset.

The taxonomy used in *Randstad* has a three-level hierarchy, where *JobCode*s are leaf classes, and each internal (*JobGroup*)/root class (*JobClass*) is an aggregation of all its descendant classes. An example[7] is illustrated in Figure 4.6.

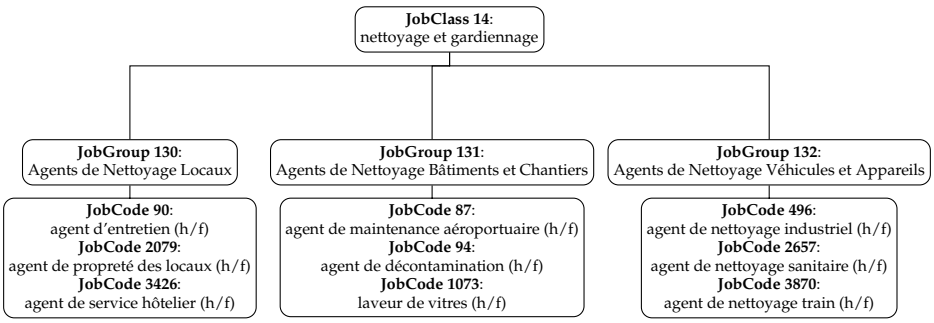


FIGURE 4.6: An example of the taxonomy used in Randstad dataset.

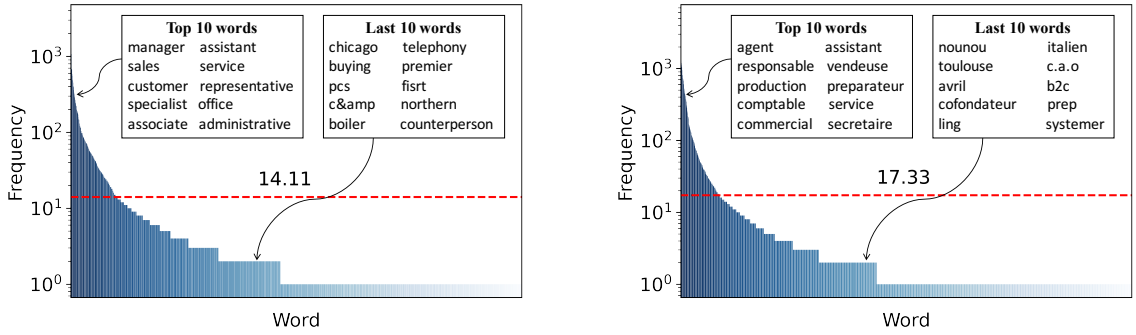
In this experiment, we categorized job titles to the root level, i.e., *Major Group*/*JobClass*. Due to the highly imbalanced label distribution, we further filtered out the job titles with low-frequency labels for both datasets. Specifically, we filtered out *Minor Group*s/*JobGroup*s that have a frequency lower than 200 for *CB12*/*Randstad* datasets. Also, for graph construction, we remove working histories with less than two work records, i.e., $|H^u| < 2$.

4.5.2 Tag Generation

We use the Top-200 tokens that appear most frequently in job titles and belong to IPOD [Junhua Liu et al., 2019] as tags for both datasets. Note that IPOD is in English, while our Randstad dataset is

[7] We randomly select three *JobGroup*s under the same *JobClass*, and for each *JobGroup*, three *JobCode*s are displayed.

in French, so we manually translate the IPOD tags into French.[8] More specifically, we first tokenize titles into tokens and remove stopwords, numbers, and punctuation. The word frequency distribution of words in two datasets are shown respectively in Figure 4.7, which are subject to the long-tail distribution, similar to the observation in [Denghui Zhang et al., 2019].



(a) CB12.

(b) Randstad.

FIGURE 4.7: Word frequency distribution, where the red line represents the average value. *Top10* words are sorted by frequency, and *Last10* are randomly selected from the words with a frequency of 2.

Most words appear only once, i.e., 53.55% of words only appear one time in *CB12* dataset, and this ratio is 56.55% in *Randstad* dataset. Figure 4.7 further shows the top ten and last ten frequent words in each dataset. Obviously, high-frequency words like "manager" and "sales" describe the responsibility or functionality of the job title, while low-frequency words are usually noise or person-specific words. Based on the domain-specific Named Entity tags (i.e., *RES*ponsibility, *FUN*ction) proposed in IPOD [Junhua Liu et al., 2019], we then select the Top-200 tokens that appear most frequently and appear in the IPOD Named Entity tag set as tags for each dataset.[9]

Moreover, we assign the one-hot encoding of the corresponding title for each job title node as the node feature. The vocabulary set $\mathcal{W}$ is obtained by filtering words with a frequency of 1 from the tokenized job titles. Afterward, various graphs can be constructed using the corresponding work history, and tag sets $\mathcal{Q}$. Statistics for datasets and different graphs are summarized in Table 4.3. We can observe that the generated *Job-Transition Graph*s (i.e., $|\mathcal{V}|$ and $|\mathcal{E}^{jj}|$) are sparse. We present details of some parts of these graphs in Appendix A.1.

4.5.3 Experimental Settings

We consider the following graph embedding approaches. They are naturally divided into *Homogeneous* and *Heterogeneous* according to the type of input graph and can be further categorized into *Unsupervised* and *Semi-Supervised* according to whether node labels are provided for learning. Section 3.1

[8] The French version of the tag set is provided in Appendix B.1.3.

[9] Note that the Randstad dataset uses translated IPOD labels. Two complete tag sets are listed in Appendix B.1.3.

TABLE 4.3: Statistics of datasets and corresponding graphs, #C represents the number of categories, and $|\mathcal{W}|$ represents the vocabulary size for node one-hot encoding.

	#C	$\lvert\mathcal{W}\rvert$	$\lvert\mathcal{J}\rvert$	$\lvert\mathcal{Q}\rvert$	$\lvert\mathcal{E}^{jj}\rvert$	$\lvert\mathcal{E}_E^{jj}\rvert$	$\lvert\mathcal{E}^{jt}\rvert$
CB12	16	1,682	9,216	200	20,640	6,477,819	22,149
Randstad	18	2,303	12,864	200	36,722	6,663,267	22,897

describes these approaches in detail in Section. Here, we only present how they are implemented and/or adapted to our enriched graphs.

- **Homogeneous & Unsupervised**
  - *Node2Vec* ($N2V$) [Grover and Leskovec, 2016]: It is an extension of DeepWalk with a biased random walk process for neighborhood exploration. For the graph G^{jj}, we use the original flexible biased random walk strategy, which explores neighborhoods in BFS and DFS fashion, as explained in Section 3.2 and the original paper [Grover and Leskovec, 2016]. More specifically, considering a random walk that traverses from node v_x to node v_y and now resides at node v_y, the standard Node2Vec calculates the transition probability from node v_y to the next node v_z using the following formula:

$$Pr(v_{w_{(i+1)}} = v_z | v_{w_{(i)}} = v_y) = \frac{\alpha(x,z) \cdot \texttt{weight}_{yz}}{Z}, \quad \text{if } e_{xy} \in \mathcal{E}^{jj},$$

    where Z is the normalizing constant, $v_{w_{(i+1)}}$ is the $i+1$-th node in the walk and $\texttt{weight}_{yz}$ is the weight of edge e_{yz}. In Node2Vec, $\alpha(x,z)$ is controlled by a return parameter p and an in-out parameter q, i.e., p controls the probability of returning v_x after visiting v_y, and q controls the probability of exploring undiscovered parts of the graph. More detailed calculation formulas are given in Section 3.1.

    For G^{jj}, we set $p = 0.25$ and $q = 0.25$, which are set according to the best performance, while for G_E^{jj}, random walks are sampled from $\mathcal{E}^{jj} \cup \mathcal{E}_E^{jj}$ with $p = 0.25$ and $q = 1$.

- **Homogeneous & Semi-supervised**
  - *GCN* [Kipf and Welling, 2016]: It is a semi-supervised GNN that generalizes the convolutional operation to HomGs.
  - *GAT* [Veličković, Cucurull, et al., 2017]: It uses a self-attention strategy to learn the importance between a node and its neighbors.

  For these two methods, node features are modeled as one-hot encodings whose dimension is the vocabulary size, as listed in Table 4.3.

- **Heterogeneous & Unsupervised**

  - *metapath2vec* ($M2V$) [Dong, Chawla, and Ananthram Swami, 2017]: It performs meta-path-guided random walk and utilizes Skip-Gram to embed HetGs. We test all meta-paths (i.e., "Job-Job" and "Job-Tag-Job") in G^{jtj} and report the best performance.

- **Heterogeneous & Semi-supervised**

  - *RGCN* [Schlichtkrull et al., 2018]: It is an extension of GCN on HetGs, introducing relation-specific transformations based on the type of edges. For G^{jj}, G^{jj}_E and G^{jt}, there is only one edge type, and in G^{jtj}, we have two relation transformations.
  - *HAN* [X. Wang, Ji, et al., 2019]: It proposes a hierarchical attention mechanism, i.e., node-level and semantic-level for HetGs.

In addition to the comparison between network embedding methods, we will also compare the representation learned through graphs with the representation obtained by semantic-based methods.

- **Semantic-based**

  - *Word2Vec* ($W2V$) [Q. Le and Mikolov, 2014]: The representation of a job title is obtained by averaging word vectors in it. We use word vectors trained on Google News[10] for *CB12*, and a pre-trained French embedding model [Fauconnier, 2015] for *Randstad*.
  - *BERT* [Devlin et al., 2018]: The job title representations are obtained by using the bert-as-service package [Xiao, 2018], a sentence encoding service for mapping variable-length sentences to fixed-length vectors. We default to using the pre-trained BERT models provided by the package, i.e., BERT-Base-Uncased is used for *CB12*, and BERT-Base-Multilingual-Cased (New) for *Randstad*.[11]

Our implementation is based on the PyTorch version of the DGL package [Minjie Wang et al., 2019]. To ensure fairness, we keep the same data split for all methods, and we set the dimension of node embedding to 128 for all the above methods, except for $W2V$. We then propose to evaluate the quality of learned representations through two downstream tasks: (i) the *job title classification* task and the *next-job prediction* task.

For *job title classification* task: we classify job titles into root categories in this work, i.e., *Major Group* for *CB12* and *JobClass* for *Randstad*. We randomly split the data into training/validation/test sets with a ratio of 60%/20%/20%. For unsupervised methods, node representations are learned from the entire dataset. Then, a classifier is trained on the training and validation sets simultaneously. In our experiments, we use logistic regression as the classifier applied to node representations. Semi-supervised models are trained on the training set using a multi-class classification loss function (i.e., categorical cross entropy).

For *next-job prediction* task: we treat it as a link prediction task on *Job-Transition Graph* to predict whether there is an edge (transition) between two nodes (job titles). We keep the same split ratio on positive/negative edges to generate training/validation/test sets, where negative edges are randomly

[10] https://code.google.com/archive/p/word2vec/

[11] They were the state-of-the-art models at the time of these experiments. Testing more recent models is a direction for our future work. We also report results obtained using another language model, trained on the Randstad data in Appendix A.1.

picked from unconnected node pairs (i.e., the same size as positive edges). Similar to the *job title classification* task, for unsupervised methods, node representations are learned from the entire dataset, and the logistic regression is used as a link classifier applied to edge representations. Here, we treat link prediction as a binary classification. Edges in the training and validation sets are used to train the classifier. Edge features are represented by applying binary operators [Grover and Leskovec, 2016] on pairs of node representations, and then the best operator is chosen based on the validation set, while the dot product is used for all semi-supervised methods. Semi-supervised models are trained on the training set using a binary-class classification loss function (i.e., binary cross entropy).

In both tasks, semi-supervised models are trained on the training set, the parameters are optimized on the validation set, and the final performance is evaluated on the test set. Models are optimized with the Adam [Kingma and Ba, 2014] with a learning rate of 1e-3, and we apply L_2 regularization with a value 5e-4. We use an early stop with a patience of 100, i.e., if the validation loss does not decrease in 100 consecutive epochs, we stop training. For models applying the attention mechanism, the dropout rate of attention is set to 0.2. For random-walk-based methods, including $N2V$ and $M2V$, we set the window size to 5, walk length to 10, walks per node to 50, and the number of negative samples to 5. We repeat each prediction experiment ten times and report the average performance scores (i.e., Macro-F1 and Micro-F1 for *job title classification* and AUC for *next-job prediction*).

Macro-F1 and Micro-F1 are standard metrics for multi-class classification, which are defined as different aggregation methods for F1 scores. Specifically, Macro-F1 is the unweighted mean of the F1 scores $\texttt{F1}_i$ of different categories $Y_i \in \mathcal{Y}$, given in the following given by the following formula:

$$\texttt{Macro-F1} = \frac{\texttt{F1}_1 + \cdots + \texttt{F1}_{|\mathcal{Y}|}}{|\mathcal{Y}|},$$

where $|\mathcal{Y}|$ is the total number of categories. The F1 score is calculated as:[12]

$$\texttt{F1} = 2\frac{\texttt{Precision} \times \texttt{Recall}}{\texttt{Precision} + \texttt{Recall}},$$

where $\texttt{Precision} = \frac{\texttt{True Positive}}{\texttt{True Positive + False Positive}}$ is used to measure the probability that the classifier is correct when it makes a positive prediction, i.e., the number of true positive results divided by the number of all positive results, and $\texttt{Recall} = \frac{\texttt{True Positive}}{\texttt{True Positive + False Negative}}$ is to measure how many positive results are correctly predicted, i.e., the number of true positive results divided by the number of all samples that should be identified as positive. The calculation of Micro-F1 is similar to the F1 score, but instead of calculating for each individual category, it calculates the sum of True Positives, False Positives, and False Negatives of all categories, respectively, and then substitutes these three values into the F1 score equation. The larger the value of Macro-F1 and Micro-F1, the better the classification performance. AUC is the "Area under the ROC Curve", and the ROC Curve (Receiver Operating Characteristic Curve) is a curve showing the performance of a binary classification model at all classification thresholds. This curve plots the True Positive Rate and False Positive Rate. One way to interpret AUC is the probability that the model ranks a random positive example higher than a random negative example. If AUC is equal to 0.5, it proves that the model is useless.

[12] We omit the superscript of i without loss of clarity.

4.6 Results

In this section, we present our experimental results. For all tables in this section, scores in bold are the best in each metric, and scores in the underline are the second best.

4.6.1 Job Title Classification

TABLE 4.4: Job title classification results (Macro-F1/Micro-F1). The score in bold is the best among all methods applied to all graphs, and the scores underlined are the best in all graphs of each method. For *M2V*, we report the best results obtained by the meta-path *Job-Tag-Job*. Note that, *N2V*, *GCN* and *GAT* are methods for homogeneous graphs, so we do not report their results for G^{jt} and G^{jtj}.

		N2V	GCN	GAT	M2V	RGCN	HAN	W2V	BERT
CB12	G^{jj}	0.206/0.360	0.576/0.688	0.568/0.664	0.154/0.334	0.524/0.637	0.670/0.747	0.713/0.767	0.688/0.719
	G^{jj}_E	0.599/0.714	0.628/0.720	0.692/0.759	0.571/0.688	0.591/0.701	0.698/0.781		
	G^{jt}	-	-	-	0.588/0.692	0.687/0.752	0.703/0.766		
	G^{jtj}	-	-	-	0.588/0.692	0.703/0.766	**0.742/0.797**		
Randstad	G^{jj}	0.201/0.304	0.520/0.616	0.529/0.593	0.166/0.282	0.388/0.536	0.592/0.665	0.595/0.671	0.580/0.609
	G^{jj}_E	0.523/0.623	0.484/0.621	0.607/0.677	0.469/0.585	0.452/0.580	0.607/0.689		
	G^{jt}	-	-	-	0.590/0.665	0.552/0.643	0.572/0.663		
	G^{jtj}	-	-	-	0.590/0.665	0.600/0.678	**0.641/0.708**		

Table 4.4 summarizes the best results of all methods on different graphs. We have the following observations:

(i) Among all graphs, all models generally have the lowest scores on G^{jj} because this graph is often sparse and can only provide limited information, as mentioned earlier.

(ii) All models perform better on G^{jj}_E (except Macro-F1 of *GCN*) than G^{jj}, which shows that the enhanced edges provide additional information. One interpretation is that enhanced edges add semantic information, i.e., two job titles are more likely to be similar if they share the same word, which in our case, is tackled under a graph perspective. Hence, it is able to address the sparsity problem while adding semantic information to the representation learning.

(iii) The heterogeneous models perform well on our proposed G^{jtj}, which indicates that the added tag nodes can effectively improve the quality of representation. Specifically, G^{jtj} carries both semantic information (i.e., provided by the "has/in" edge) and pairwise topological information (i.e., provided by the job transition relation), which makes it perform better than other graphs that only contain one information. Note that we did not apply homogeneous methods to G^{jtj}, but the results on G^{jj}_E prove that the information given by tags is useful. To make our results more complete, we will attempt to adapt homogeneous methods to heterogeneous graphs, such as a variant of Node2Vec [Valentini et al., 2021]. The full results can be found in Appendix A.1. For *M2V*, we report the best results obtained by the meta-path *Job-Tag-Job*.

(iv) The models with attention mechanisms outperform the models without attention, demonstrating that the attention mechanism is good at capturing important information from noisy graphs.

(v) Graph-based methods perform better than semantic-based methods, which justifies our argument (i.e., due to the messy data, simply combining the representation of words may increase the likelihood of mismatches) is correct.

4.6.2 Next-Job Prediction

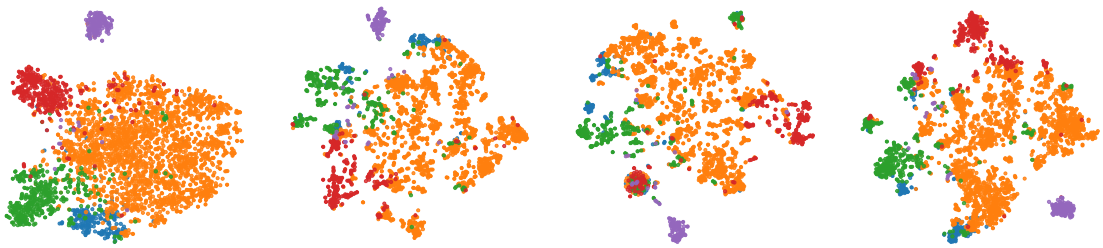
We further evaluate the learning scheme using *next-job prediction*, which can be viewed as a link prediction task to predict whether a position will be recommended as the next-job. For unsupervised methods, edge features are represented by applying binary operators [Grover and Leskovec, 2016] on node pairs, and then the best binary operator is selected based on the validation set, while the dot product is used for semi-supervised methods. The results on *CB12* given in Table 4.5 show the promising results of our proposed graphs. Like *job title classification*, the scores of all network embedding methods, i.e., *N2V*, *GAT*, *M2V* and *HAN* better on G_E^{jj} compared to G^{jj}, and the heterogeneous models perform best on G^{jtj}. Such results further demonstrate the effectiveness of our method for constructing graphs, whether adding additional information based on tags (i.e., G_E^{jj}) or directly adding tags to graphs (i.e., G^{jt} and G^{jtj}). *BERT* using the Hadamard operator performs best, followed by *HAN* on G^{jtj} with a slight difference of 0.007. However, when we use the dot product used in *HAN* to obtain edge features for *BERT*, the AUC of *BERT* drops sharply to 0.477, while *W2V* only drops a little to 0.763. We will analyze these results more deeply in future work. Overall, link prediction results also demonstrate the effectiveness of our proposed graphs.

TABLE 4.5: Next-job prediction results (AUC) on *CB12*. The bold score is the best among all methods, and the underlined score is the second-best.

	N2V	**GAT**	**M2V**	**HAN (Dot)**	**W2V (Dot)**	**BERT (Dot)**	**W2V (Hadamard)**	**BERT (Hadamard)**
G^{jj}	0.564	0.704	0.548	0.685	0.763	0.477	0.777	**0.840**
G_E^{jj}	0.692	0.789	0.593	0.792				
G^{jt}	-	-	0.604	0.768				
G^{jtj}	-	-	0.604	<u>0.833</u>				

4.6.3 Visualization

For a more intuitive comparison, we visualize the learned representations for an example of the data in Figure 4.8.



(a) HAN ($\mathcal{G}^{jj}$). (b) HAN ($\mathcal{G}^{jj}_{E}$). (c) HAN ($\mathcal{G}^{jt}$). (d) HAN ($\mathcal{G}^{jtj}$).

FIGURE 4.8: Visualization of representations (CB12). *Healthcare support* (green), *Healthcare practitioners and technical* (blue), *Architecture and engineering* (purple), *Office and administrative support* (orange) and *Transportation and material handling* (red).

Specifically, we select five types of job titles for illustration purposes, including two similar occupations: *Healthcare support* and *Healthcare practitioners and technical*, as well as three non-similar occupations: *Architecture and engineering*, *Construction and extraction* and *Transportation and material handling*. We use t-SNE [Van der Maaten and Hinton, 2008] to reduce the representation dimension. Each color corresponds to an occupation category. Overall, the representations learned by *HAN* on all graphs are clustered into groups. However, when considering tags, representations are easier to be subdivided further in each category. For example, as shown in Figure 4.8d, the administrative occupation (i.e., orange) can obviously be further divided into three sub-clusters, which proves that adding tag nodes can help capture more detailed information and make the learned representation more informative. This detailed information helps further categorize occupations, as we only classify job titles into the root category (i.e., *Major Group*) in this work. We further provide visualizations for other occupations in Appendix A.1.

4.7 Conclusion and Perspectives

In this chapter, we study the problem of job title representation learning from the graph perspective. We first propose to enrich *Job-Transition Graph* commonly used in job title representation learning tasks by adding tag-related information or directly adding tag nodes. Then we learn job title representations through network embedding methods. This enhanced method can alleviate the sparsity problem, thereby improving the quality of learned representations, as demonstrated in the experimental results of *job title classification* and *next-job prediction*. Based on the experimental results, we can now answer the Research Questions (RQs) proposed in Section 4.2.

- **RQ1**: Can the graph structure provide more useful information for job title representation learning?
  Answer: *Yes, according to the performance comparison between semantic-based methods and methods using graphs, we can conclude that using graphs can help job title representation learning.*

- **RQ2**: Does additional information resulting in complex heterogeneous graph help learn better job title representations?
  Answer: *Yes, either adding tag-related information (i.e., Enhanced Job-Transition Graph) or adding tags directly to graphs (i.e., Job-Tag Graph and Job-Transition-Tag Graph) bring better results than the standard Job-Transition Graph. Such results demonstrate that the additional information carries useful information for job title representation learning.*

Perspectives When generating graphs, we did not consider edge weights, such as the number of job transitions between two job titles and the number of the same tags shared by two job titles. We think these weights can provide more information about job title similarity, i.e., the job title j_x is more likely to be more similar to j_y if they have a job transition edge with weight ten ($\texttt{weight}_{xy} = 10$) than j_z, which has only one transition from j_x ($\texttt{weight}_{xz} = 1$). Therefore, one research direction in future work is to consider edge weights when constructing graphs. In the process of graph construction, we combine *Job-Transition Graph* with *Job-Tag Graph* to build *Job-Transition-Tag Graph*. In future work, we will try different combinations, such as combining *Enhanced Job-Transition Graph* with *Job-Tag Graph*. Another research direction is to improve the tag generation approach and consider tags related to "responsibility" and "functionality" separately. In the *job title classification* task, we classify job titles to the root level, which is the coarsest category in the occupation taxonomy, as illustrated in Figure 4.4. In the future, we will classify job titles into different occupational levels to explore whether tags can provide more profound information. We also would like to explore why the Hadamard operator and dot product lead to different link prediction results for *BERT*, as shown in Section 4.6.2. Finally, to make our experiments more complete and make the results more convincing, we will compare representations obtained by simultaneously considering topological and semantic information with those obtained by simply aggregating these two parts of information, which are learned independently.

Chapter 5

Skill Representation Learning by Leveraging Hierarchical Graph

5.1 Motivation

Skills play a vital role in the labor market and are the backbone of recruitment, i.e., occupations, job postings, and talents are accompanied by a set of skills, and whether a talent is suitable for a position is first measured from skill fit. Consequently, learning good skill representation is of prime importance and can improve multiple tasks in recruitment portals. Most existing works focus on skill extraction [Kivimäki et al., 2013; M. Zhao et al., 2015; Khaouja et al., 2019; Gugnani and Misra, 2020] and skill relevance or popularity analysis [W. Zhou et al., 2016; Börner et al., 2018; T. Xu et al., 2018], as summarized in Section 2.2. To the best of our knowledge, in the literature, only a few works have addressed the problem of skill representation learning. [Dave et al., 2018] proposes to jointly learn skill and job representations from three graphs: job transition, job-skill, and job co-occurrence graphs. In [Nigam et al., 2020], the authors propose to encode skills utilizing a BERT [Devlin et al., 2018] model trained from scratch on a manually annotated skills dataset.

From the study of these previous works, we can derive interesting findings that motivate the work of this chapter. The first observation is that, like job title representation learning (Chapter 4), skill representation learning can benefit greatly from advances in graph embeddings. In particular, the *skill co-occurrence graph* constructed from job descriptions or resumes is important information for skill representation learning using graphs, as done in [Dave et al., 2018]. Furthermore, as highlighted by [Nigam et al., 2020], another characteristic of skills is that they are organized into competency groups, i.e., groups of similar skills required to do a job successfully. In graphs, these capability groups can be analogized to communities, opening up a research direction for skill representation learning, i.e., using community-preserving graph embedding methods. Finally, in the world of recruitment, skills also tend to be in a tree-like structure, representing a hierarchical relationship, called a *skill taxonomy*. These taxonomies can come from standard open ontologies in the recruitment field, such as ESCO (Appendix B.2.4) and ROME (Appendix B.2.5), or private company references, such as Randsatd. While this hierarchical information is well-structured and informative, it has rarely been studied directly and has never been used for skill representation learning to the best of our knowledge. Therefore, in this task, we exploit this hierarchical information, i.e., the *skill taxonomy* to learn better skill representations, along with the *skill co-occurrence graph*.

Contributions In summary, our main contributions in this chapter include the following:

- We present an overview of the community and hierarchy preserving graph embedding models.
- We propose to learn skill representations from **co-occurrence relations**, and enhance the representations by exploiting the common predefined reference, i.e., **skill taxonomy**. Our work is **the first** attempt to exploit this hierarchical information directly for skill representation learning.
- We propose to benchmark different representative community and hierarchical community preserving methods for skill representation learning on two real-world datasets using an occupation classification.
- Experimental results show that the skill representations learned by cooperating co-occurrence relations and hierarchical information can **improve occupation classification** task.

5.2 Research Scope

From the methodological point of view, the work of this chapter is related to works that integrate the community structure into network embeddings [X. Wang, P. Cui, et al., 2017; T. Zhang et al., 2020]. The main idea is that the representation of vertices within a community should be more similar than vertices belonging to different communities. Not only vertices form communities, but communities are usually organized in a hierarchy, i.e., a tree-structured hierarchy. This has motivated some typical methods [Du et al., 2018; Long et al., 2019; Bhowmick et al., 2020] that try to preserve hierarchical community structures within vector spaces. We describe these methods in detail below. Our objective is to adapt these methods to the problem of skill representation learning and answer the following Research Questions (RQs):

- **RQ1**: Can the graph structure provide more useful information for skill representation learning than the semantic representation?
- **RQ2**: Does the hierarchical information (i.e., skill taxonomy) contribute positively to skill representation learning?

5.3 Our objective: Preserving Pairwise Proximity and Community Hierarchy

In this section, we first formulate our problem, review some representative community and hierarchical community preserving graph embedding models, and then compare these methods to inspire our future perspective, a novel hierarchical community preserving graph embedding for skill representation learning. For facilitating illustration, we list some important mathematical notations used throughout this chapter in Table 5.1, unless particularly specified.

TABLE 5.1: Mathematical notations used in *Skill Representation Learning by Leveraging Hierarchical Graphs* chapter.

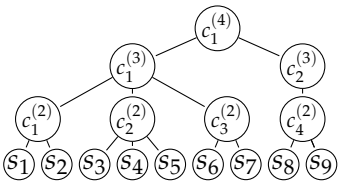
Notation	Description
G	graph
$\mathcal{V}, \mathcal{E}$	node set, edge (link) set
$\boldsymbol{A}$	adjacency matrix
v_x, e_{xy}	x-th node, edge between node pair (v_x, v_y)
$\texttt{weight}_{xy}$	weight of edge e_{xy}
$\texttt{deg}(v)$	degree of node v
$\Phi(\cdot)$	embedding mapping function
$\mathcal{N}(v)$	neighbors of node v
$\mathcal{O}$	occupation set
$\mathcal{S}$	skill set
G^{ss}	skill co-occurrence graph
$\mathcal{E}^{ss}$	skill co-occurrence edge set
$\Phi(\cdot)$	embedding mapping function
$\mathcal{N}(v)$	neighbors of node v
$\boldsymbol{X}$	feature matrix of skills
$\mathcal{H}$	skill taxonomy or hierarchical community tree of G
L	depth of $\mathcal{H}$, or number of layers in $\mathcal{H}$
C	community-node set in $\mathcal{H}$
$C^{(l)}$	the set of community-node at l-level ($C^{(l)} = \{c_i^{(l)}\}_{i \in \{1,\dots,m^l\}}$)
$\mathcal{U}$	node set of $\mathcal{H}$ (i.e., $\mathcal{U} = \mathcal{S} \cup C$)
$\mathcal{C}$	the community, is a group of nodes
$\texttt{Ch}(u)$	child nodes of node $u \in \mathcal{U}$
$\texttt{Pa}(u)$	parent node of node $u \in \mathcal{U}$
Q	modularity

5.3.1 Problem Formulation

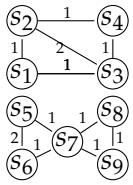
Benefiting from the fact that each occupation/job posting/talent can be represented as a set of skills they require or possess, we first build a *skill co-occurrence graph* to reflect the pairwise similarity between skills. More specifically, given a set of occupations $\mathcal{O}$ and a set of skills $\mathcal{S}$, where each occupation $o \in \mathcal{O}$ is pre-associated with a set of skills, i.e., $o = \{s_1, \dots, s_{|o|}\} \subset \mathcal{S}$. Then the *skill co-occurrence graph* $G^{ss} = (\mathcal{S}, \mathcal{E}^{ss}, \boldsymbol{X}, \boldsymbol{W})$, an undirected homogeneous graph, which is constructed according to $\mathcal{O}$, with the skill set $\mathcal{S}$ as nodes, and each edge $e_{xy} \in \mathcal{E}^{ss}$ indicates that s_x and s_y appear in the same occupation. $\boldsymbol{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{|\mathcal{S}|}\}$ is the node feature matrix where $\boldsymbol{x}_i \in \mathbb{R}^F$ is the attribute vector associated with s_i, here the dimension F is the vocabulary size of the words in skill names, and $\texttt{weight}_{xy}$ is the weight of edge e_{xy}, indicating the number of times s_x and s_y appear together. Note that, for simplicity, here, we only discuss the occupation set. It can be replaced by a set of jobs or talents, where skills are extracted from job postings or resumes using the skills extraction tool [Kivimäki et al., 2013; M. Zhao et al., 2015].

We give an example with five occupations and their required skills, and the corresponding *skill co-occurrence graph* is shown in Figure 5.1.

$o_1 = \{s_1, s_2, s_3\}$
$o_2 = \{s_2, s_3, s_4\}$
$o_3 = \{s_5, s_6\}$
$o_4 = \{s_5, s_6, s_7\}$
$o_5 = \{s_7, s_8, s_9\}$



(A) Skill taxonomy $\mathcal{H}$.



(B) Skill co-occurrence graph G^{ss}.

FIGURE 5.1: An example of *skill taxonomy* $\mathcal{H}$ and *skill co-occurrence graph* G^s.

Moreover, we also assume that skills are organized in a *skill taxonomy*. These taxonomies, such as ESCO (Appendix B.2.4)[1], are available for standardizing the recruitment industry. We denote it as $\mathcal{H}$, where leaf nodes represent "skills", internal and root nodes are "skill categories", denoted as C, so nodes $\mathcal{U}$ in $\mathcal{H}$ consist of two types: skill nodes $\mathcal{S}$ and category-nodes C, i.e., $\mathcal{U} = \mathcal{S} \cup C$. From high-level to low-level, category-nodes describe more fine-grained skill types. More formally, we define the category-nodes at l-level as $C^{(l)} = \{c_i^{(l)}\}_{i \in \{1,\dots,m^l\}}, l \in \{2, \cdots, L\}$, where $c_i^{(l)}$ is a category-node and m^l is the number of category-nodes at l-level. Note that $C^{(L)} = \{c_1^{(L)}\}$ is the root node. An example of such a *skill taxonomy* with a depth of four (i.e., $L = 4$) is given in Figure 5.1a, where the first three high-level layers (i.e., $C^{(4)}$, $C^{(3)}$ and $C^{(2)}$) are "skill categories", and leaf nodes (i.e., $\{s_i\}_{i \in \{1,\dots,9\}}$) are "skills". We further use $\texttt{Ch}(c_i^{(l)}) \subseteq C^{(l-1)}$ to denote the child node set of the node $c_i^{(l)}$, and $\texttt{Pa}(c_i^{(l)}) \in C^{(l+1)}$ to denote the parent node of $c_i^{(l)}$.

Using the above notations, we formally define the **skill representation learning** problem as follows:

Given a set of occupations $\mathcal{O}$, where each occupation can be represented as a set of required skills, the corresponding *skill co-occurrence graph* G^{ss}, and a given *skill taxonomy* $\mathcal{H}$, we want to

Learn a skill embedding mapping $\Phi: \mathcal{V} \to \mathbb{R}^d$ from skill graphs to d-dimensional skill representations, which preserves the pairwise proximity, community structure, and hierarchical community structure.

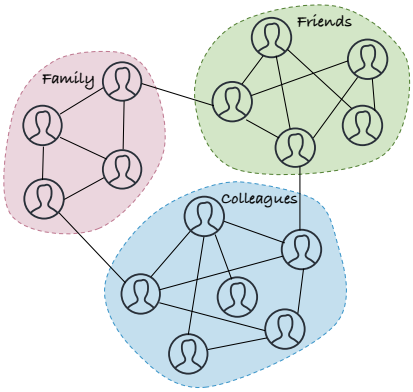
In the next section, we give an overview of related graph embedding methods: (i) community preserving graph embeddings and (ii) hierarchical community structure preserving graph embeddings.

5.3.2 Review of Community Preserving Graph Embedding Models

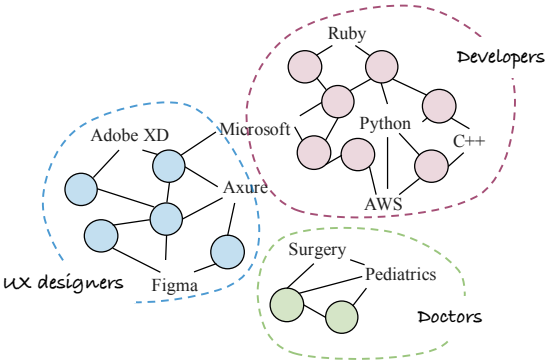
In addition to the general proximity between pairs of nodes, e.g., the first- and second-order proximity (Definition 3.1.5 and Definition 3.1.6), an interesting property of real-world networks is often the community structure, i.e., the nodes of the network can be grouped into (potentially overlapping) node sets of different sizes. For example, as shown in Figure 5.2, in a social network, communities can represent different relationships, such as friends, colleagues, and family members, and in a recruitment

[1] https://esco.ec.europa.eu/en

scenario, communities in a skill co-occurrence graph can represent different skills required for a specific profession, such as skills for developers, UX designers, and doctors.



(a) A social network with three communities: friends, colleagues and family.



(b) A skill network with three communities: different skills of developers, UX designers, and doctors.

FIGURE 5.2: Examples of communities.

Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph, where $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$ is the set of nodes, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, each edge is represented by a node pair $e_{xy} = (v_x, v_y)$, $x, y \in [1, |\mathcal{V}|]$. the community is formally defined as:

Definition 5.3.1 Community: *A community $\mathcal{C}$ refers to a group of nodes $\mathcal{C} \subseteq \mathcal{V}$ such that intra-group connections are denser than inter-group ones [Meng Wang et al., 2015]. A graph can be decomposed into several disjoint (possibly overlapping) communities $\{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$, with $\mathcal{V} = \bigcup_{i=0}^{m} \mathcal{C}_i$.*

Definition 5.3.2 Modularity: *The modularity is a measure of graph structure and is often used in community detection tasks to measure the quality of the community partitioning of a graph. It measures the difference between the actual number of edges within communities and the expected number of edges (i.e., if the edges were randomly placed).The general expression for modularity is:*

$$Q = \frac{1}{2|\mathcal{E}|} \sum_{v_i, v_j \in \mathcal{V}} \sum_{\mathcal{C} \in \{\mathcal{C}_i, \ldots, \mathcal{C}_m\}} \left(\boldsymbol{A}_{ij} - \frac{\texttt{deg}(v_i)\,\texttt{deg}(v_j)}{2|\mathcal{E}|} \right) \mathbb{I}_{i\mathcal{C}} \mathbb{I}_{j\mathcal{C}},$$

where $\boldsymbol{A}$ is the adjacency matrix, $\texttt{deg}(v)$ is the degree of node v and $\mathbb{I}_{i\mathcal{C}}$ is an indicator, i.e., $\mathbb{I}_{i\mathcal{C}} = 1$ if node v_i is assigned to community $\mathcal{C}$, otherwise $\mathbb{I}_{i\mathcal{C}} = 0$.

Modularity values for unweighted and undirected graphs are in the $[-1/2, 1]$ range. A graph with a high modularity score has many connections within a community but few connections out to other communities. There are other measures of graph structure,[2] here we mainly focus on modularity.

In the following, we introduce some methods that preserve community information in the embedding space. In particular, we present M-NMF, an approach based on Matrix Factorization, and CommDGI, a Graph Neural Network (GNN) approach.

[2] http://braph.org/braph-1-0/manual/graph-measures/

- **Modularized Nonnegative Matrix Factorization (M-NMF)** [X. Wang, P. Cui, et al., 2017]: M-NMF preserves both the pairwise node similarity and the community structure into network embedding. In particular, for the pairwise node similarity, M-NMF incorporates the first- and second-order proximities of nodes (Definition 3.1.5 and Definition 3.1.6) to learn the representations using Matrix Factorization. For the community structure, the communities are detected by a modularity constraint term. Then, these two terms can be jointly optimized by using an auxiliary community representation matrix to connect the two terms through a consensus relationship between node representations and the community structure of the network.

- **Community Deep Graph Infomax (CommDGI)** [T. Zhang et al., 2020]: CommDGI is an extension of Deep Graph Infomax (DGI) to handle community detection.

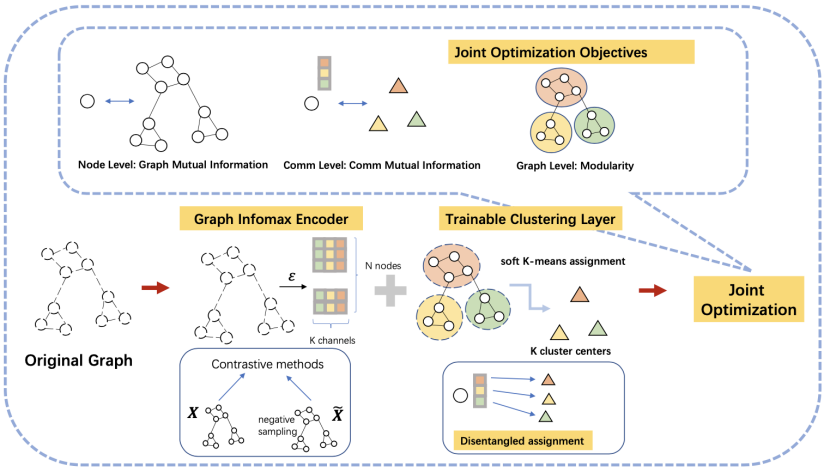


FIGURE 5.3: The framework of CommDGI, from [T. Zhang et al., 2020]. Given a graph, CommDGI learns hidden node representations via a *Graph Infomax Encoder* and clusters these node representations using a *Trainable Clustering Layer*, which is optimized together with the *Graph Infomax Encoder*.

As shown at the top of Figure 5.3, CommDGI maximizes together two kinds of mutual information:

  - **Graph mutual information**: is calculated between nodes and the graph. Similar to DGI, CommDGI uses a GCN to encode node representations and trains the GCN with contrastive loss $\mathcal{L}_{graph}$ (Equation 3.2).
  - **Community mutual information**: is calculated between nodes and the community (sub-graph). CommDGI includes a clustering layer (*Trainable Clustering Layer* in Figure 5.3) implementing a differentiable K-means clustering which produces a soft assignment of the nodes to K clusters in the embedding space. The cluster centers $\boldsymbol{\mu}_C$ are then optimized by:

$$\boldsymbol{\mu}_C = \frac{\sum_i r_{iC} \boldsymbol{h}_i}{\sum_i r_{iC}}, \forall C \in \{C_1, \ldots, C_m\},$$

where $\boldsymbol{h}_i$ is the representation of node v_i, and $r_{i\mathcal{C}}$ is the probability that v_i is assigned to cluster $\mathcal{C}$, which is calculated by:

$$r_{i\mathcal{C}} = \frac{\exp\left(-\delta\|\boldsymbol{h}_i - \boldsymbol{\mu}_{\mathcal{C}}\|\right)}{\sum_{\mathcal{C}'\in\{\mathcal{C}_1,\ldots,\mathcal{C}_m\}}\exp\left(-\delta\|\boldsymbol{h}_i - \boldsymbol{\mu}_{\mathcal{C}'}\|\right)}, \forall \mathcal{C} \in \{\mathcal{C}_1,\ldots,\mathcal{C}_m\},$$

where δ is an inverse temperature hyperparameter that defines the difficulty of using the clustering process.

The community mutual information objective is thus defined as:

$$\mathcal{L}_{community} = \frac{1}{|\mathcal{V}|}\sum_{i=1}^{|\mathcal{V}|}\sum_{\mathcal{C}\in\{\mathcal{C}_1,\ldots,\mathcal{C}_m\}}\mathbb{E}_{(\mathbf{H},\boldsymbol{\mu})}[\log \mathtt{D}(r_{i\mathcal{C}}\boldsymbol{h}_i, \boldsymbol{\mu}_{\mathcal{C}})].$$

$\mathtt{D}\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *discriminator*, and $\mathtt{D}(r_{i\mathcal{C}}\boldsymbol{h}_i, \boldsymbol{\mu}_{\mathcal{C}})$ represents the probability score assigned to a node-cluster pair $(r_{i\mathcal{C}}\boldsymbol{h}_i, \boldsymbol{\mu}_{\mathcal{C}})$.

CommDGI further applies the modularity objective into the clustering layer as:

$$\mathcal{L}_{modularity} = \frac{1}{|\mathcal{E}|}\mathrm{Tr}\left[\boldsymbol{R}^T\left(\boldsymbol{A}_{ij} - \frac{\mathtt{deg}(v_i)\mathtt{deg}(v_j)}{2|\mathcal{E}|}\right)\boldsymbol{R}\right].$$

where $\boldsymbol{R} = \{r_{i\mathcal{C}}\}$ is the cluster assignment matrix.

5.3.3 Review of Hierarchical Community Structure Preserving Graph Embedding Models

Not only do vertices form communities, but the communities are usually organized in a hierarchy, i.e., a tree-structured hierarchy. Smaller communities at lower levels come together to form larger communities at higher levels. Before we start introducing methods of network embedding that preserve this hierarchy, we first formally define the hierarchical network as:

Definition 5.3.3 Hierarchical Network: *A graph $G = (\mathcal{V}, \mathcal{E})$ is called a hierarchical network if it can be decomposed into different communities, and these communities exist at different levels of granularity. At the higher (coarser) level, there are a few large communities; at the lower (finer) level, there are many small communities. Formally, the hierarchical community tree of G is denoted as $\mathcal{H}$, the depth is L, for example $L = 4$ in Figure 5.4. The community set at the l-th layer is denoted $\mathcal{C}^{(l)} = \{\mathcal{C}_1^{(l)},\ldots,\mathcal{C}_{m^l}^{(l)}\}$, $\mathcal{C}_i^{(l)} \subseteq \mathcal{V}$. For example, in Figure 5.4, there are three communities at 3-level, i.e., $\mathcal{C}_1^{(3)}$, $\mathcal{C}_2^{(3)}$ and $\mathcal{C}_3^{(3)}$, where $\mathcal{C}_1^{(3)} = \{v_1, v_2, v_3, v_4, v_5\}$. In particular, $|\mathcal{C}^{(L)}| = 1$ and $\mathcal{C}_1^{(L)}$ contains all nodes of G, i.e., $\mathcal{C}_1^{(L)} = \mathcal{V}$.*

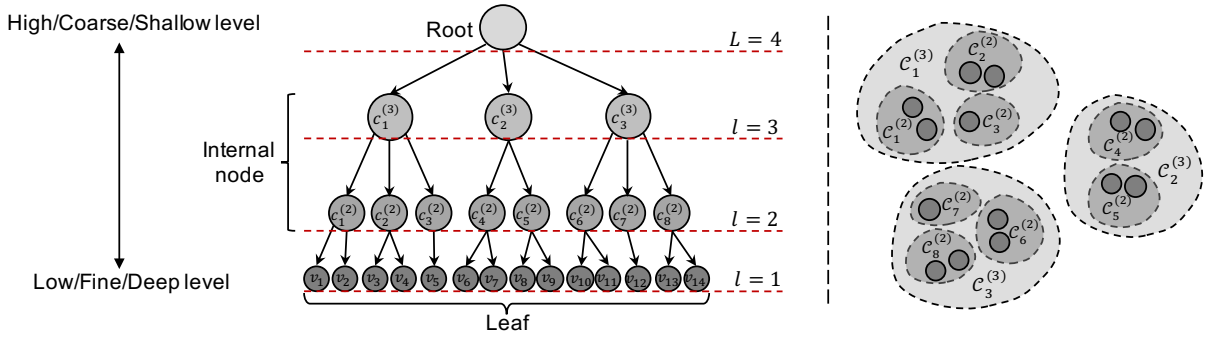


FIGURE 5.4: A formal illustration of a 4-level hierarchical graph/tree $\mathcal{H}$, and the communities.

Moreover, the node set of the tree is denoted as $\mathcal{U}$, i.e., $\mathcal{V} \subseteq \mathcal{U}$. For a node $u \in \mathcal{U}$, $\mathcal{Ch}(u)$ and $\mathcal{Pa}(u)$ denotes the set of child nodes and the parent node of node u, respectively. We further denote the nodes other than the leaf nodes $\mathcal{V}$ in the tree $\mathcal{H}$ as community-nodes, i.e., $c_i^{(l)}$ is the community-node of the community $\mathcal{C}_i^{(l)}$. Note that the community-node here is the same as the category-node in the skill taxonomy described in Section 5.3.1.

Hierarchical community structure preserving graph embedding methods encode rich community and hierarchical structure information into a vector space, making it much easier to analyze the graph at different scales. Some representative works include:

- **LouvainNE** [Bhowmick et al., 2020]: LouvainNE first employs the Louvain algorithm [Blondel et al., 2008] to recursively coarsen a large graph into smaller meta-graphs (communities) and construct the hierarchy of meta-graphs. For example, in Figure 5.5, the meta-graphs are S_1 with nodes $\{O, G, N, L, M, P, J, E, D, A, Z\}$ and S_{12} with nodes $\{E, D, A, Z\}$.

  It then obtains representations of nodes in the original graph at different levels of the hierarchy and aggregates these representations to learn the final node embedding.

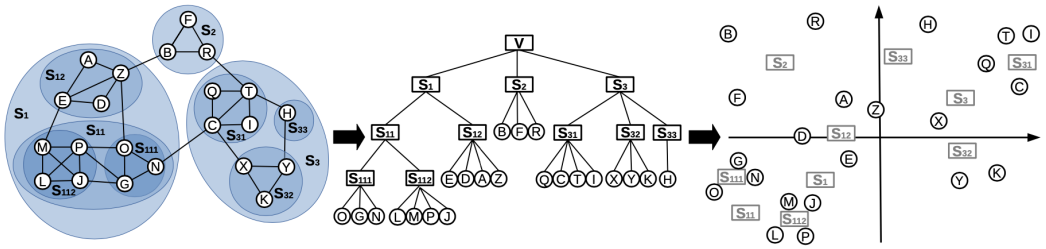


FIGURE 5.5: The embedding generation process of LouvainNE: from graph to hierarchical tree to embedding, from [Bhowmick et al., 2020].

More specifically, LouvainNE proposes two techniques to learn the embedding vector for each node except the root node:

– **Standard embedding**: given any tree-node, including the root but not the leaves, LouvainNE constructs a weighted undirected meta-graph where the nodes are children of the considered tree-node and the weighted edge between two children is calculated based on the number of edges between them in the original graph. For example, the weighted edge between children S_1 and S_2, as shown in Figure 5.5, is given by:

$$\texttt{weight}_{S_1,S_2} = \frac{|\mathcal{E}_{S_1,S_2}|}{|S_1| \cdot |S_2|},$$

where $\mathcal{E}_{S_1,S_2}$ is the edge set between the node sets S_1 and S_2 in the original graph. Once the meta-graphs are obtained, a standard graph embedding algorithm, e.g., DeepWalk or Node2Vec is used to obtain embeddings of each one of these meta-graphs independently.

– **Stochastic embedding**: generates a random vector for each tree-node from a standard normal distribution with zero mean and unit variance.

After obtaining the node embedding of each meta-graph, the final node embedding $\boldsymbol{v}$ of each node in the original graph $v \in G$ is given as:

$$\boldsymbol{v} = \sum_{l=1}^{L-1} \alpha^l \boldsymbol{v}^l,$$

where $\boldsymbol{v}^l$ is the embedding of leaf v in the l-th level meta-graph. For example, as shown in Figure 5.5, the embedding of node O is calculated as $\boldsymbol{v}_O = \alpha_1 \boldsymbol{v}_O^{S_{111}} + \alpha_2 \boldsymbol{v}_O^{S_{11}} + \alpha_3 \boldsymbol{v}_O^{S_1}$, where $\boldsymbol{v}_O^{S_{111}}$ is the node representation of O in the sub-graph S_{111}.

- **Galaxy Network Embedding (GNE)** [Du et al., 2018]: Inspired by the galaxy hierarchy, GNE proposes an optimization problem with spherical constraints to preserve the hierarchical community structure. Communities are embedded into a low-dimensional spherical surface, the center of which represents the parent community they belong to, as shown in Figure 5.6.

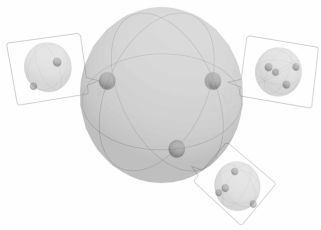


FIGURE 5.6: Spherical galaxy model, from [Du et al., 2018].

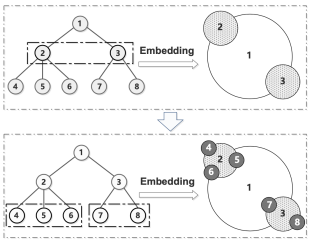


FIGURE 5.7: The structure of GNE, from [Du et al., 2018].

More specifically, the optimization task is constrained by:

– **Local community structure** is preserved through the pairwise node similarity in the same community. For $c_i^{(l)}$ and $c_j^{(l)}$ two community-nodes derived from the same parent

$c_k^{(l+1)}$, $c_i^{(l)}, c_j^{(l)} \in \texttt{Ch}(c_k^{(l+1)})$ the pairwise similarity between them is preserved by the second-order proximity of LINE:

$$\min_{\Phi,\Phi'} \mathcal{O}_k^{(l)} = -\sum_{c_i^{(l)},c_j^{(l)} \in \texttt{Ch}\left(c_k^{(l+1)}\right)} S_{i,j}^{(l)} \log Pr\left(\Phi'\left(c_j^{(l)}\right) | \Phi\left(c_i^{(l)}\right)\right),$$

where $\Phi(\cdot)\colon \mathcal{V} \to \mathbb{R}^d$ and $\Phi'(\cdot)\colon \mathcal{V} \to \mathbb{R}^d$ are two embedding mapping functions, which embed each node $v \in \mathcal{V}$ into a d-dimensional latent space, and similar to LINE, $\Phi(v)$ is the embedding vector of v when it is treated as the source node and $\Phi'(v)$ is the vector when it is treated as the "context" of other nodes. $S_{i,j}^{l}$ is the community proximity of communities $\mathcal{C}_i^{(l)}$ and $\mathcal{C}_j^{(l)}$ (i.e., their community-nodes are $c_i^{(l)}$ and $c_j^{(l)}$), calculated as follows:

$$S_{i,j}^{(l)} = \frac{1}{|\mathcal{C}_j^{(l)}| \cdot |\mathcal{C}_j^{(l)}|} \sum_{v_x \in \mathcal{C}_i^{(l)}} \sum_{v_y \in \mathcal{C}_j^{(l)}} \frac{\boldsymbol{A}_x^T \boldsymbol{A}_y}{\sqrt{\|\boldsymbol{A}_x^T\|_1 \cdot \|\boldsymbol{A}_y^T\|_1}},$$

which means the average common neighbor similarity between $\mathcal{C}_i^{(l)}$ and $\mathcal{C}_j^{(l)}$, where $\boldsymbol{A}_x$ is the x-th column of the adjacency matrix of G.

– **Global hierarchical structure** is preserved by *horizontal* and *vertical* constraints:

  * *Horizontal constraint*: nodes in the same community are more similar than nodes belonging to different communities. For all community-nodes $c_i^{(l)}$, $c_j^{(l)}$ and $c_k^{(l)}$ at l-level, with $\texttt{Pa}(c_i^{(l)}) = \texttt{Pa}(c_j^{(l)})$ and $\texttt{Pa}(c_i^{(l)}) \neq \texttt{Pa}(c_k^{(l)})$, they obey the following constraint:
  $$\|\Phi(c_i^{(l)}) - \Phi(c_j^{(l)})\| < \|\Phi(c_i^{(l)}) - \Phi(c_k^{(l)})\|,$$
  where $\|\cdot\|$ means 2-norm.
  * *Vertical constraint*: the cohesion of communities at shallow-level should be less than that of communities at deep-level. The community cohesion in GNE is calculated by the average representation distance between the sub-communities in it. Therefore, the parent-child relationship can be described as: for all community-nodes $c_i^{(l+1)}$ at $(l+1)$-level, all $c_j^{(l)} \in \texttt{Ch}(c_i^{(l+1)})$ and all $c_k^{(l-1)} \in \texttt{Ch}(c_j^{(l)})$, they obey the following constraint:
  $$\|\Phi(c_i^{(l+1)}) - \Phi(c_j^{(l)})\| > \|\Phi(c_j^{(l)}) - \Phi(c_k^{(l-1)})\|.$$

The final optimization formulation is:

$$\min_{\boldsymbol{\Phi},\boldsymbol{\Phi}'} \mathcal{O}_i^{(l)}\left(\Phi, \Phi'\right)$$
$$\text{s.t.} \quad \forall c_j^{(l)} \in \texttt{Ch}(c_i^{(l+1)}), \quad \|\Phi(c_i^{(l+1)}) - \Phi(c_j^{(l)})\|_2 = r_i^{(l+1)},$$

where $r_i^{(l+1)}$ denotes the sphere radius of community $\mathcal{C}_i^{(l+1)}$, and all children nodes of the community-node $c_i^{(l+1)}$ are embedded on a sphere with center $c_i^{(l+1)}$ and radius $r_i^{(l+1)}$. The

learning process recurses from top to bottom in the hierarchy. The whole embedding procedure is illustrated in Figure 5.7.

- **Subspace Network Embedding (SpaceNE)** [Long et al., 2019]: SpaceNE is inspired by the fact that subspaces within Euclidean space inherently follow a hierarchy, as shown in Figure 5.8. It preserves the *pairwise proximity* and the *proximity between hierarchical communities*, including structural information within and among communities. More specifically, node representations can be learned via the following scheme:

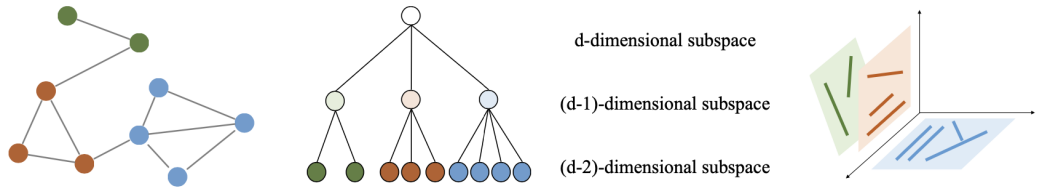


FIGURE 5.8: The correspondence between the community hierarchy and the subspace hierarchy, from [Long et al., 2019].

  - SpaceNE models the pairwise proximity between nodes in a similar way to DeepWalk:

$$\min_{\mathbf{\Phi}} \mathcal{L}_1 = \sum_{v_x \in \mathcal{V}} \sum_{v_y \in \mathcal{N}(v_x)} \log \sigma\left(\|\Phi(v_x) - \Phi(v_y)\|_2\right) + k \cdot \mathbb{E}_{v_z \propto} \left[\log \sigma\left(-\|\Phi(v_x) - \Phi(v_z)\|_2\right)\right],$$

    where $\mathcal{N}(v_x)$ denotes the neighbors of v_x in graph G, $\sigma(\cdot)$ is the sigmoid function, and k is the number of negative nodes v_z, i.e., $e_{xz} \notin \mathcal{E}$.

  - SpaceNE models the proximity of hierarchical communities by considering the following two types of information:

    * *Structural information within each community*: nodes from the same community should be closer than nodes from different communities, so nodes from the same community are projected into the same subspace. For each community-node $c_i^{(l)}$, the constraint is given as:

$$\texttt{rank}(\boldsymbol{U}_i^{(l)}) \leq d^{(l)},$$

      where $d^{(l)}$ is the dimension of the subspace at layer (l), and $\boldsymbol{U}_i^{(l)}$ is a matrix where each row is the embedding vector of each node belonging to the community $\mathcal{C}_i^{(l)}$. Note that each node has a new vector under the basis vector of the corresponding subspace (community).

    * *Structural information among communities*: minimize the difference between subspace similarity and community similarity for every two communities $\mathcal{C}_i^{(l)}$ and $\mathcal{C}_j^{(l)}$, namely,

$$\min \mathcal{L}_2^{(l)} = \|\mathbf{\Delta}^{(l)} - \mathbf{\Gamma}^{(l)}\|_F,$$

where $\mathbf{\Delta}^{(l)}$ is a matrix, and each entry $\mathbf{\Delta}^{(l)}_{i,j}$ is the similarity of two communities $\mathcal{C}^{(l)}_i$ and $\mathcal{C}^{(l)}_j$ in the original graph, the calculation method of $\mathbf{\Delta}^{(l)}$ can be customized according to the data set. Similarly, $\mathbf{\Gamma}^{(l)}_{i,j}$ is the similarity of two subspaces. $\|\cdot\|_F$ is the Frobenius norm.

Graph Neural Networks (GNNs) (see Section 3.1 for more description of GNNs) can also be extended to preserve hierarchical community structures into the embedding space. For example,

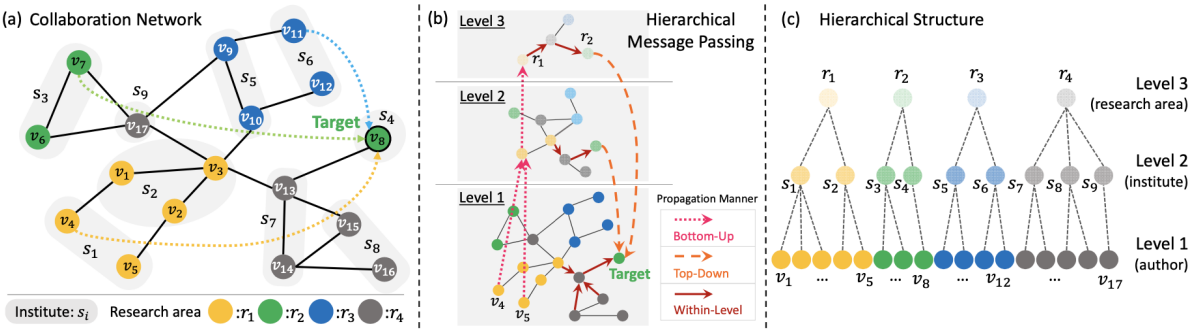


FIGURE 5.9: Elaboration of hierarchical message passing: (a) a collaboration network, (b) the hierarchical message-passing mechanism based on (a) and (c), and (c) the identified hierarchical structure, from [Zhong, C.-T. Li, and Pang, 2020].

- **Hierarchical Community-aware Graph Neural Network (HC-GNN)** [Zhong, C.-T. Li, and Pang, 2020]: HC-GNN is the first implementation of the *Hierarchical Message-passing Graph Neural Network* framework that detects and exploits the hierarchical community structure for message passing. The learning process consists of four stages:
  - Generate the hierarchical structure of the original graph G using the Louvain algorithm [Blondel et al., 2008].
  - Construct l-level graph based on its lower-level (i.e., $l-1$) graph, as illustrated in the middle of Figure 5.9. To this end, the hierarchical structure of G is represented as a list of graphs $\{G_1, \ldots, G_L\}$, where G_1 represents the original graph G.
  - Hierarchical message propagation consists of three parts:
    * *Bottom-up Propagation*: using node embeddings in G_{l-1} to update node embeddings in G_l in the hierarchy $\{G_1, \ldots, G_L\}$:

$$\boldsymbol{a}^{(k)}_{c^{(l)}_i} = \frac{1}{|\texttt{Ch}(c^{(l)}_i)|+1}\left(\sum_{c^{(l-1)}_j \in \texttt{Ch}(c^{(l)}_i)} \boldsymbol{h}^{(k-1)}_{c^{(l-1)}_j} + \boldsymbol{h}^{(k-1)}_{c^{(l)}_i}\right),$$

where $\boldsymbol{h}^{(k-1)}_{c^{(l)}_i}$ is the node embedding of $c^{(l)}_i$ generated by $(k-1)$ GNN layer, and $\boldsymbol{a}^{(k)}_{c^{(l)}_i}$ is the updated embedding of $c^{(l)}_i$.

* *Within-level Propagation*: propagating information in each graph $G_l \in \{G_1, \ldots, G_L\}$. More specifically, for each node $v_x \in G_l$, the within-level propagation is formulated as:

$$\boldsymbol{b}_{v_x}^{(k)} = \boldsymbol{U} \cdot \texttt{MEAN}(\boldsymbol{a}_{v_y}^{(k)}), \forall v_y \in \mathcal{N}(v_x) \cup \{v_x\}.$$

  $\boldsymbol{U}$ is a learnable matrix, `MEAN` is an element-wise mean pooling.

* *Top-down Propagation*: adopting the graph attention mechanism to learn the contributions of different levels during information integration. More specifically, given a node v_x of the original graph G, its final node embedding is obtained by

$$\boldsymbol{h}_{v_x}^{(k)} = \texttt{ReLU}(\boldsymbol{V} \cdot \texttt{MEAN}\{\alpha_{cx}\boldsymbol{b}_c^{(k)}\}), \forall c \in \mathcal{P}(v_x) \cup \{v_x\},$$

  where α_{cx} is the trainable attention score between node v_x and community-node $c \in \mathcal{P}(v_x)$ or itself, and $\mathcal{P}(v_x)$ represents the set of community-nodes at different levels to which node v_x belongs. For example, in Figure 5.9, $\mathcal{P}(v_1) = \{s_1, r_1\}$.

– **Train the model** with a node classification objective.

We summarize, here, in Table 5.2, the representative works that preserve *community structure* and *hierarchical community structure* introduced in the above two sections.

TABLE 5.2: Comparison of different community and hierarchical community structure preserving graph embedding models. Here, *Top2Bottom* and *Bottom2Top* denote methods for aggregating hierarchical information from top to bottom and bottom to top, respectively, and *Learning Scheme* includes both supervised and unsupervised.

	Method	Input	Top2Bottom	Bottom2Top	Node or Edge feature	Pairwise proximity	Learning Scheme
Community	M-NMF	graph	-	-	**no**	first- and second-order	unsupervised
	CommDGI	graph	-	-	**yes**	GNN (GCN)	unsupervised
Hierarchy	LouvainNE	graph (hierarchical community tree is generated by Louvain algo. from the graph)	**no**	**no**	**no**	standard graph embedding, e.g., DeepWalk, LINE and Node2Vec	unsupervised
	GNE	graph + its ready-made hierarchical community tree	vertical constraint	**no**	**no**	second-order proximity similar to LINE	unsupervised
	SpaceNE	graph + its ready-made hierarchical community tree	subspace projection	**no**	**no**	first-order proximity similar to DeepWalk	unsupervised
	HC-GNN	graph (hierarchical community tree is generated by Louvain algo. from the graph)	graph attention	mean	**yes**	GNN (GAT,GraphSAGE)	supervised (node classification)

Here, we point out two limitations that they usually have:

(i) most of the methods ignore node features (i.e., the semantic information of skills in our scenario), which are important in the recruitment field,

(ii) they [Du et al., 2018; Long et al., 2019] only consider one direction information, i.e., top-to-bottom, and ignore bottom-to-top information, which can provide more pairwise proximity.

Benefiting from the ability of GNNs to encode node features and graph topology efficiently, HC-GNN [Zhong, C.-T. Li, and Pang, 2020] applies GCN [Kipf and Welling, 2016] and GAT [Veličković, Cucurull, et al., 2017] to the within-level and the top-down propagation schemes, respectively, after using a mean bottom-up propagation scheme. These schemes allow messages to propagate between nodes across different levels and between nodes within the same hierarchy. However, HC-GNN aggregates information from bottom to top using a simple mean operation, and we argue that the order of the three propagations can be changed for better performance, which will be studied in future work. **These limitations inspire us to propose a new model in the future that will consider both top-bottom and bottom-top information transfer, and this model takes node features into the learned representation**.

5.4 Benchmark Graph Embeddings for Skill Representation Learning

5.4.1 Datasets

In this work, we conduct experiments on two real-world datasets. Both datasets have an occupation set $\mathcal{O}$ and a known skill taxonomy $\mathcal{H}$. They are described below, and detailed descriptions can be found in Appendix B.2.4 and Appendix B.2.5.

ESCO It is a multilingual European classification of Skills, Competences, and Occupations.[3] It provides descriptions of occupations and associated skills. In this work, we use the English version. We further consider one of the two sub-trees of the *ESCO* skill taxonomy (i.e., "skills" and "knowledge"), named *ESCO_K*, where the nodes are all knowledge-related, as shown in Figure 5.10.

ROME It is the Operational Directory of Trades and Jobs used to categorize and identify trades based on associated skills.[4] It has an inventory of the names of trades, jobs, knowledge, and know-how.

We generate *skill co-occurrence graph*s from the corresponding occupation set $\mathcal{O}$, with statistics summarized in Table 5.3.

[3] https://ec.europa.eu/esco/portal

[4] https://data.europa.eu/data/datasets/58da857388ee384902e505f5?locale=en

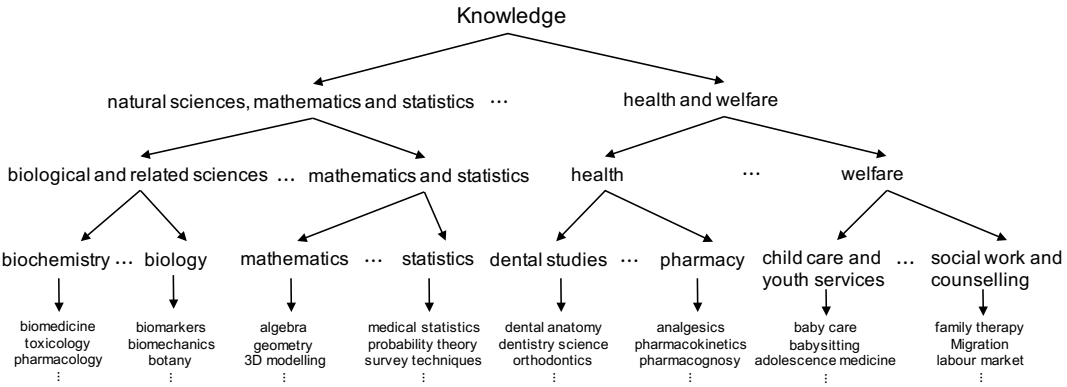


FIGURE 5.10: A knowledge sub-tree of *skill taxonomy* in ESCO.

TABLE 5.3: Dataset statistics. $|\mathcal{N}_{avg}|$ is the average number of neighbors per node, F is the vocabulary size for node features. *Hierarchy* represents the number of nodes in each layer from root to leaf, and these nodes are either "skill categories" or "skills". *#O_C* is the number of occupation categories, and $\#S_{avg}$ is the average number of skills required for each occupation.

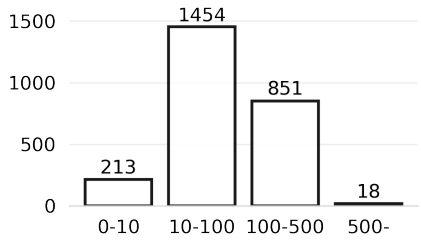
Dataset	G^s				G^o		$\mathcal{T}$		#O_C	$\#S_{avg}$
	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{E}^{ss}\rvert$	$\lvert\mathcal{N}_{avg}\rvert$	F	$\lvert\mathcal{O}\rvert$	$\lvert\mathcal{E}^{oo}\rvert$	$\lvert\mathcal{S}\rvert+\lvert\mathcal{C}\rvert$	Hierarchy		
ESCO_K	2,536	116,218	91.63	2,290	1,639	1,342,341	2,646	$\{1, 9, 25, 75, 2536\}$	10	10.80
ESCO	11,368	1,191,122	209.56	5,682	1,701	1,445,850	11,846	$\{1, 2, 17, 99, 359, 11368\}$	10	37.82
ROME	11,428	587,213	102.77	8,424	532	141,246	11,839	$\{1, 53, 357, 11428\}$	14	49.93

5.4.2 Dataset Analysis

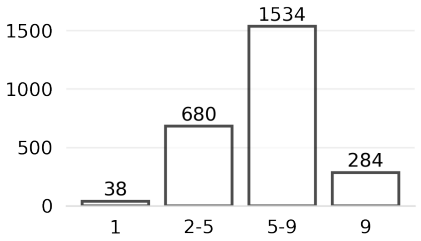
This section provides more analysis on these datasets from different aspects. We analyze the neighborhood of each node in the *skill co-occurrence graph*, including the number of neighboring nodes and the number of types of these neighboring nodes (according to the *skill taxonomy*). The statistics are shown in Table 5.4, and the detailed distribution of *ESCO_K* is given in Figure 5.11. The detailed distributions of *ESCO* and *ROME* are given in Figure B.7 and Figure B.8.

TABLE 5.4: Average number of neighbors and average number of neighbor types at different level (i.e., different skill category granularity) per node in *skill co-occurrence graph*.

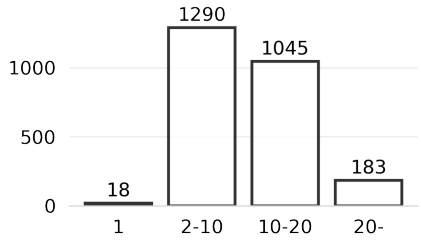
	Dataset							
	ESCO_K			**ESCO**			**ROME**	
$\lvert\mathcal{N}(s)\rvert$	91.63			209.56			102.77	
Level l ($\lvert C^{(l)}\rvert$)	2(75)	3(25)	4(9)	2(359)	3(99)	4(17)	2(357)	3(53)
Avg	19.45	10.38	5.83	68.84	37.43	11.47	32.21	14.40



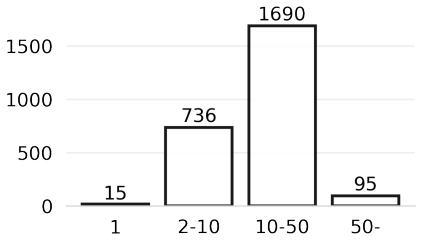
(A) Neighbor number.



(B) 4-level skill categories (9).



(C) 3-level skill categories (25).



(D) 2-level skill categories (75).

FIGURE 5.11: Neighborhood statistics for *ESCO_K*. In (a), the x-axis represents the size of the neighborhood $|\mathcal{N}(s)|$, and in (b), (c) and (d), the x-axis represents the number of neighbor types (i.e., "skill categories"). The y-axis of all subfigures represents the corresponding number of nodes.

According to Table 5.3 and Table 5.4, ESCO is denser, and the skills are more versatile, required by most occupations. Furthermore, based on the *skill taxonomy*, we group skills according to the "skill category" they belong to, resulting in different levels of communities. For example, the (4)-th level of *skill taxonomy* in *ESCO_K* has 9 "skill categories", so there will be 9 skill communities $|C^{(4)}| = 9$. We show the modularity of each *skill co-occurrence graph* in Table 5.5. We further use the commonly used hierarchical clustering algorithm Louvain [Blondel et al., 2008] to discover the hierarchical community structure and show the resulting modularity in Table 5.6.

TABLE 5.5: Graph modularity Q based on skill taxonomy $\mathcal{H}$.

Dataset	Level l ($\|C^{(l)}\|$)	Q
ESCO_K	2 (75)	0.237
	3 (25)	0.333
	4 (9)	0.363
ESCO	2 (359)	0.048
	3 (99)	0.097
	4 (17)	0.072
ROME	2 (357)	0.104
	3 (53)	0.206

TABLE 5.6: Graph modularity Q based on Louvain.

Dataset	Level l ($\|C^{(l)}\|$)	Q
ESCO_K	2 (44)	0.552
	3 (9)	0.580
ESCO	2 (63)	0.537
	3 (13)	0.557
	4 (11)	0.558
ROME	2 (164)	0.439
	3 (18)	0.495
	4 (14)	0.496

The modularity score based on the *skill taxonomy* is generally lower than the score calculated by the Louvain algorithm, especially for *ESCO* dataset.

TABLE 5.7: Top 10 common skills for *ESCO* and *ROME*.

ESCO	ROME
create solutions to problems (S1)	coordonner l'activité d'une équipe (00031)
use different communication channels (S1)	outils bureautiques (00007)
manage staff (S4)	management (00031)
keep records of work progress (S2)	gestion administrative (00007)
have computer literacy (S5)	gestion comptable (00008)
quality standards (04)	techniques pédagogiques (00017)
troubleshoot (S1)	suivre l'état des stocks (00029)
manage budgets (S4)	utiliser un engin nécessitant une habilitation (00016)
identify customer's needs (S1)	techniques commerciales (00007)
mechanics (07)	définir des besoins en approvisionnement (00029)

Common skills in *ESCO* and *ROME* are listed in the table 5.7. Not surprisingly, the skills in high demand are "soft skills", such as communication and problem-solving skills.

5.4.3 Experimental Settings

The baselines used for skill representation learning can be classified into the following types:

- **Traditional methods** do not use graph structures to obtain skill representations or occupation representations.
  - *One-Hot*: It encodes occupation representation as binary vectors, with dimension being the number of skills in the skill set. Taking the occupations shown in Figure 5.1 as an example, there are nine skills $\{s_i\}_{i=1}^{9}$, so the representation of occupation $o_1 = \{s_1, s_2, s_3\}$ is encoded as $[1,1,1,0,0,0,0,0,0]$.
  - TF-IDF [Salton and Buckley, 1988]: It treats occupations as documents and skills as words and then calculates a score for each skill using the standard TF-IDF method. It can be seen as a weighted version of *One-Hot*.
  - *Doc2Vec* [Q. Le and Mikolov, 2014]: It also treats occupations as documents and skills as words and trains occupation representations using distributed memory.
  - *Semantic*: skill representations are obtained by averaging the pre-trained word vectors in skills.
- **Local neighborhood preserving methods** learn skill embedding vectors using the homogeneous *skill co-occurrence graph* G^{ss} defined in 5.3.1. We select the representative methods described in Section 3.1, which can be categorized into *unsupervised* or *semi-supervised* learning methods.
  - **Unsupervised**: no labeled nodes are provided for node (i.e., skill) representation learning.
    - *DeepWalk*: It simulates uniform random walks on the graph to capture the first-order proximity, and the generated random walks are used to update node representations using the Skip-Gram approach with a hierarchical softmax.

    - *LINE*: It learns node representations by explicitly modeling the first- and second-order proximity.
    - *Node2Vec*: It is an extension of *DeepWalk* with a biased random walk process that can balance Depth First Search (DFS) and Breadth First Search (BFS) for neighborhood exploration.
    - *DGI*: is an unsupervised GNN that maximizes the mutual information between the graph representation and node representations.
  - **Semi-supervised**: labeled nodes are provided for node (i.e., skill) representation learning. Here we use skill classification.
    - *GCN*: generalizes the convolutional operation to graphs.
    - *GAT*: uses a self-attention to learn the importance between a node and its neighbors.

- **Community structure preserving methods** learn from the *skill co-occurrence graph* G^{ss} by preserving node proximity and community structure. We consider the two methods described in Section 5.3.2 in our benchmark.
  - *M-NMF*: preserves community structure via matrix factorization.
  - *CommDGI*: jointly maximizes the graph mutual information, community mutual information, and modularity.

- **Hierarchical community structure preserving methods** learn from *skill co-occurrence graph* G^{ss} and *skill taxonomy* $\mathcal{H}$ by preserving node proximity and hierarchical community structure.
  - *LouvainNE*: the final skill representation is a weighted sum of skill representations in different sub-communities.
  - *GNE*: preserves community hierarchy through spherical projection.
  - *SpaceNE*: introduces the subspace idea to preserve the hierarchy.
  - *LouvainNE_O*: unlike *LouvainNE*, which uses Louvain to discover hierarchical communities, *LouvainNE_O* uses the skill taxonomy structure directly, and skill representations at different levels are learned via DeepWalk.

We evaluate the quality of learned skill representations by an *occupation classification* task. We set the dimension of skill representation to 100 for all methods except traditional methods. Semi-supervised methods, including *GCN* and *GAT*, which learn skill representations using a skill classification loss (8:1:1 nodes for train/validation/test), where the "skill category" is set according to the skill taxonomy. We report the results obtained with the best "skill category" settings, i.e., 75/359/357 for *ESCO_K*/*ESCO*/*ROME*, respectively. For unsupervised methods which need to specify the number of clusters, e.g., *M-NMF*, *CommDGI*, the number of clusters is chosen also according to the taxonomy, and the best results are reported in Table 5.9. For detailed hyperparameter settings, see Appendix A.1.

5.5 Results

In this section, we present our experimental results. For all tables in this section, scores in bold are the best in each metric, and scores in the underline are the second best.

5.5.1 Occupation Classification

In this experiment, we evaluate the learned skill representations using an occupation classification task. Since an occupation can be represented as a set of skills, we express the occupation representation $\boldsymbol{o}$ in terms of the corresponding skill representations:

$$\boldsymbol{o} = \texttt{AGGREGATE}(\{\ \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{|o|}\}).$$

Here $\texttt{AGGREGATE}(\cdot)$ is an aggregation operator, which is used to aggregate skill representation $\boldsymbol{s}$ to form occupation representations. Occupation classification is then performed on these aggregated occupation representations. Aggregators can be max pooling, mean pooling, or other trainable operations. In this experiment, the occupation representation is the average of the corresponding skill representations. The detailed occupation categories in the two datasets are listed in Table 5.8. There are nine categories for the *ESCO* dataset, and *ROME* has 14 categories.

TABLE 5.8: The detailed occupation categories.

Dataset	Occupation Category
ESCO	(1) Technicians and associate professionals, (2) Service and sales workers, (3) Elementary occupations, (4) Professionals, (5) Managers, (6) Craft and related trades workers, (7) Skilled agricultural, forestry and fishery workers, (8) Plant and machine operators and assemblers, (9) Clerical support workers, (10) Armed forces occupations
ROME	(1) Agriculture et Pêche, Espaces naturels et Espaces verts, Soins aux animaux, (2) Arts et Façonnage d'ouvrages d'art, (3) Banque, Assurance, Immobilier, (4) Commerce, Vente et Grande distribution, (5) Communication, Média et Multimédia, (6) Construction, Bâtiment et Travaux publics, (7) Hôtellerie-Restauration, Tourisme, Loisirs et Animation, (8) Industrie, (9) Installation et Maintenance, (10) Santé, (11) Services à la personne et à la collectivité, (12) Spectacle, (13) Support à l'entreprise, (14) Transport et Logistique

Table 5.9 summarizes the best results (Macro-F1/Micro-F1) of all methods on occupation classification. We have the following observations:

(i) Methods exploiting graph structure generally perform better than traditional methods, indicating that graphs provide more effective information about pairwise similarity of skills.

(ii) Compared to unsupervised methods, semi-supervised methods learn skill representations through a skill classification loss, resulting in better occupation classification results. However, the choice of skill category does affect the result, which will be discussed in Section 5.5.2.

(iii) *CommDGI* outperforms *DGI*, suggesting that preserving community structure is meaningful for skill representation learning.

(iv) *SpaceNE* shows its advantages on *ROME* as it preserves the community hierarchy. While, on *ESCO* its performance is even worse than *Node2Vec* used to learn the initial embedding. One possible reason is that *ROME* has a deep skill taxonomy, i.e., $L = 6$. *SpaceNE* has difficulty capturing hierarchical information as the depth of skill taxonomy increases. Likewise, *GNE* performs poorly due to the "curse" of depth, as explained in [Long et al., 2019].

TABLE 5.9: Occupation classification results (Macro-F1/Micro-F1).

	Method	ESCO_K	ESCO	ROME
Traditional	One-Hot	0.513/0.567	0.640/0.665	0.806/0.814
	TF-IDF	0.451/0.550	0.511/0.611	0.422/0.536
	Doc2Vec	0.344/0.437	0.561/0.580	0.733/0.752
	Semantic (AVG+AVG)	0.497/0.545	0.560/0.611	0.772/0.795
	Semantic (AVG+SUM)	0.530/0.545	0.613/0.608	0.828/0.842
	Semantic (SUM+AVG)	0.519/0.547	0.625/0.639	0.795/0.808
	Semantic (SUM+SUM)	0.526/0.533	0.601/0.593	0.803/0.812
Local neighbor	DeepWalk (AVG)	0.505/0.538	0.627/0.646	0.847/0.859
	DeepWalk (SUM)	0.514/0.520	0.607/0.615	0.820/0.831
	LINE(1st+2st) (AVG)	0.490/0.532	0.598/0.601	0.837/0.845
	LINE(1st+2st) (SUM)	0.517/0.529	0.602/0.618	0.822/0.830
	Node2Vec (AVG)	0.514/0.555	0.630/0.646	0.831/0.845
	Node2Vec (SUM)	0.516/0.529	0.633/0.642	0.834/0.842
	GCN (AVG)	0.482/0.519	0.638/0.651	0.845/0.860
	GCN (SUM)	0.521/0.532	0.638/0.651	0.845/0.860
	GAT	0.527/0.555	**0.648**/**0.659**	0.850/0.866
	DGI	0.485/0.521	0.569/0.587	0.721/0.725
Com	M-NMF	0.338/0.409	0.116/0.310	0.107/0.303
	CommDGI	0.492/0.533	0.580/0.609	0.735/0.740
H_Comm	GNE	0.503/0.503	0.605/0.620	0.858/0.872
	SpaceNE	**0.549**/**0.567**	0.626/0.633	**0.895**/**0.888**
	LouvainNE	0.340/0.393	0.422/0.460	0.581/0.626
	LouvainNE_O	0.532/0.537	0.640/0.648	0.806/0.822

5.5.2 Skill Category Granularity

Semi-supervised methods such as *GCN* and *GAT* learn skill representations through a skill classification task, where the skill categories are selected based on the *skill taxonomy*. Due to the inherent structural nature of the taxonomy, lower-level skill categories describe more fine-grained skill characteristics. Therefore, skill representations trained with more detailed skill categories usually contain more specific information. The experimental results shown in Table 5.10 confirm this observation.

TABLE 5.10: Different skill category granularity.

	Method							
	GCN				GAT			
ESCO_K	9	25	75		9	25	75	
	0.433/0.503	0.500/0.535	**0.523/0.551**		0.436/0.510	0.515/0.522	**0.527/0.555**	
ESCO	2	17	99	359	2	17	99	359
	0.576/0.601	0.596/0.607	0.621/0.633	**0.638/0.651**	0.591/0.612	0.622/0.634	0.636/0.645	**0.648/0.659**

5.6 Conclusion and Perspectives

In this chapter, we study the problem of skill representation learning by exploiting hierarchical community information. Based on the fact that each occupation is equipped with a set of skills required to work in that occupation, we first construct a *skills co-occurrence graph* from different occupations.

Since skills are connected according to their co-occurrence relationships in occupations, the *skills co-occurrence graph* reflects their pairwise proximities. Furthermore, in the field of recruitment, skills also tend to be organized in a tree-like structure, representing a hierarchical relationship, called a *skill taxonomy*. The skills can be grouped according to different skill categories, which are further organized into hierarchies, so *skill taxonomy* reflects a hierarchical community structure. To this end, we propose to learn skill representations from these two graphs by preserving both the pairwise proximity and hierarchical community structure. We apply different community and hierarchical community structure preserving graph embedding methods for skill representation learning to verify the importance of the information carried in *skill taxonomy*. Based on the experimental results, we can now answer the Research Questions (RQs) proposed in Section 5.2.

- **RQ1**: Can the graph structure provide more useful information for skill representation learning than the semantic representation?
  Answer: *Yes, according to the performance comparison between the Semantic method and methods using graphs, we can conclude that using graphs can help skill representation learning.*

- **RQ2**: Does the hierarchical information (i.e., skill taxonomy) contribute positively to skill representation learning?
  Answer: *Yes, methods that learn representations from skills co-occurrence graph and skill taxonomy, such as GNE and SpaceNE, generally outperform other methods that do not consider the hierarchical community structure. However, they suffer from the depth curse when skill taxonomy has many layers.*

Perspectives In this chapter, we adapt some existing graph embedding methods that preserve the hierarchical community structure to learn skill representations. As summarized in Table 5.2, these methods usually suffer from two major limitations: (i) ignore node features and (ii) only aggregate hierarchical information from top to bottom. These limitations inspire us to propose a new model in the future that will consider both top-bottom and bottom-top information transfer, and this model incorporates node features into the learned representation. Additionally, we will also address the depth curse present in *GNE* and *SpaceNE*.

Chapter 6

Next-Application Prediction from Job Application Sequences

6.1 Motivation

As discussed in the introduction (Chapter 1), the widespread use of online recruitment services has recently led to an information explosion in the job market (i.e., over 3 million jobs are posted on LinkedIn in the U.S. every month).[1] Therefore, there is an urgent need for an accurate, effective, meaningful, and transparent job recommendation system. In response to this situation, many works have been proposed to build effective job recommender systems [Tripathi, Agarwal, and Vashishtha, 2016]. Like many other domains, Deep Learning (DL)-based recommendation models have been extensively studied in the recruitment domain. Nevertheless, while mostly studied, there are still important issues with current approaches, which we summarize below:

- **Issue 1**: The first issue is related to intrinsically dynamic and constantly evolving Job-Person interactions, which can be built from job application records or working histories. Identically, we assume that temporal relations between application records tend to point toward **sequential/session-based recommendations**.[2] Although various session-based methods have been proposed in other domains, such as news, E-commerce, and movies, session-based job recommendation is less studied. A thorough search of the relevant literature yielded only one related work [Lacic et al., 2020], which encodes sessions using different autoencoder architectures and then recommends the next job posting in a K-nearest-neighbor manner. Moreover, another important aspect of the Job-Person interaction matrix is its sparsity, as most job seekers apply for only a few positions from the entire repository or work in a specific occupation.

- **Issue 2**: The second issue is related to the sensitivity and privacy of data in the recruitment domain. From the perspective of content analysis methods, it is not easy to create an unbiased and evolving labeled dataset due to the sensitivity and privacy of the data in the recruitment domain [Qin et al., 2018; Zhu et al., 2018]. Therefore, it **prevents the use of fully supervised machine learning methods and calls for other learning schemes**.

[1] https://economicgraph.linkedin.com/resources/linkedin-workforce-report-february-2021

[2] In this paper, we describe recommendation task to explore the sequential data by a broader term "sequential". Thus "session" and "sequential" are interchangeable.

- **Issue 3**: The third issue is related to the **personalization of recommendations**. Indeed, two job seekers can apply for the same job with different motivations. Consequently, the important information in a job and in its application is specific to the job seeker. Thus, weighing the jobs differently in the modeling tasks of job seeker careers is essential. General-purpose approaches are often unable to capture such specific information.

- **Issue 4**: The fourth issue is due to the **evolving nature** of the recruitment domain. In fact, in addition to constantly updated E-recruitment websites (i.e., a large number of job postings are added or removed daily, and hundreds of candidate profiles are created or updated), the domain is still evolving (i.e., new occupations, formations, and skills), thus intensifying the cold-start problem. Therefore, building a static method that cannot be adjusted quickly is impractical as the data constantly changes. We thus need to avoid retraining a deep model for each change.

- **Issue 5**: The last issue is related to the diversity of preference influencing factors in the recruitment domain. Many factors may influence job seeker preferences for jobs, such as job content, analogous to article content in news recommendations, which is one of the most important factors. Besides job content, based on the uniqueness of the recruiting field, the personal context information of job seekers is also crucial. Among them, the geographic location of job seekers is a non-negligible factor, which often has a substantial impact on the application decision and the recommendation result, i.e., job seekers are more likely to apply for jobs close to their current location. It raises two opposite goals. On the one hand, it is essential to **consider and capture this personal context information when modeling personal preference**. On the other hand, when making recommendations, the recommender system also requires distinguishing this information from more core content, i.e., job content.

In order to solve these issues, in this chapter, we study the job recommendation problem modeled as a next job application prediction task. We propose a hybrid Personalized-Attention Next-Application Prediction model (PANAP) to answer the previous issues partially. Specifically, for Issue 1, we model the next job application problem as a sequential recommendation problem and then compare our proposed model with different session-based recommendation methods to analyze the dynamics in the recruitment domain. Our work is the first attempt at applying deep end-to-end networks to the next-application prediction task. Our model is composed of three independent modules. (i) The first module learns job representations from textual job content and available metadata in an unsupervised way to answer Issue 2. Text content plays an important role in describing job postings, and the way the job content is encoded affects the model performance. Therefore, we explore the utility of different content encoding methods. (ii) The second module contains a personalized-attention mechanism to learn the job seeker representation that can solve the problem of personalized recommendations mentioned in Issue 3. Different from the classical attention mechanism [Meng et al., 2019]), the personalized-attention mechanism uses job-seeker-specific query vectors, which will be explained in Section 6.3.2. (iii) The third module predicts the next application a job seeker will apply for. In order to answer Issue 4, this module is inspired by the Deep Structured Semantic Model (DSSM) [Huang et al., 2013] with a training loss based on representation similarities. To account for the "location" factor when making recommendations (i.e., Issue 5), our model adds location-related metadata, e.g.,

city and state, when modeling job and job seeker representations. Furthermore, we propose a specific negative sampling strategy that considers the geographic location of the job and job seekers.

Contributions In summary, our model is motivated by the above five issues, which address the job recommendation from the direction of next-application prediction. Its three modules answer these issues, i.e., alleviating reliance on annotated data, improving the personalization of recommendations, adapting to continuous and rapid item updates, and considering the specific preference influencing factor, respectively. Our contributions include the following:

- We propose a new session-based model, named PANAP, which is used for *Next-Application Prediction* task. It integrates the **personalized-attention mechanism to improve the recommendation accuracy**.
- We propose a **novel location-based sampling strategy** that prioritizes jobs in the same state as the job seeker as negative samples. Experiments have proved that this strategy can **improve the quality of recommendations**.
- We conduct an extensive experimental study on the CareerBuilder12 dataset to investigate the effect of leveraging **different metadata types**, **different job posting sections**, and **different text content encoders** on model performance.

6.2 Research Scope

We model the next-application problem as a sequential recommendation problem, which has been studied recently in other domains, such as news or product recommendation [F. Yu et al., 2016; J. Li et al., 2017; Souza Pereira Moreira, Ferreira, and Cunha, 2018; Fang, Danning Zhang, et al., 2020]. Recurrent Neural Networks (RNNs) have been widely studied for this problem since they have demonstrated their effectiveness in processing sequential data, one representative work is GRU4Rec [Hidasi, Karatzoglou, et al., 2015; Hidasi, Quadrana, et al., 2016; Hidasi and Karatzoglou, 2018]. The attention mechanism [Bahdanau, Cho, and Bengio, 2014] has shown promising potential in improvements of accuracy and interpretability, like the vanilla attention-based [J. Li et al., 2017; Q. Liu et al., 2018] and self-attention based one [Sun et al., 2019; Kang and McAuley, 2018]. Inspired by the success of session-based methods in other applications, we apply some representative models in the next application prediction problem to study the effectiveness of session-based models in the recruitment domain.

This chapter addresses the following Research Questions (RQ):

- **RQ1**: In the recruitment field, do we really need the sequential recommendation method?
- **RQ2**: Can the personalized-attention mechanism better capture the personal career preference?
- **RQ3**: Are job textual information and context information (e.g., geographical location and educational background) important in job recommendation?
- **RQ4**: How to take into account context information in the recommendation process?

6.3 Proposed Method: the PANAP Framework

In this section, we first formulate the *Next-Application Problem*, and then we describe in detail our proposed model, called Personalized-Attention Next-Application Prediction (PANAP).For facilitating illustration, we list some important mathematical notations used throughout this chapter in Table 6.1, unless particularly specified.

TABLE 6.1: Mathematical notations used in *Next-Application Prediction from Application Sequences* chapter.

Notation	Description
$\mathcal{U}$	job seeker set
$\mathcal{J}$	job posting set
ID	identifier of job seeker or job posting
meta	metadata attributes of job seeker or job posting
$\mathcal{H}_u$	job application record of job seeker u

6.3.1 Problem Formulation

Let $\mathcal{U}$ be a set of job seekers and $\mathcal{J}$ be a set of job postings (referred to as jobs for short). A job $j \in \mathcal{J}$ can be represented as a tuple, i.e., $j = (\texttt{ID}_j, \{w_1^j, \ldots, w_m^j\}, \texttt{meta}_j)$. $\texttt{ID}_j$ is the unique identifier of job j. $\{w_1^j, \ldots, w_m^j\}$ is a sequence of m words, representing the textual content of job j. It can be any text information about the job, such as job title and job description. $\texttt{meta}_j$ are associated metadata attributes, such as the job city. Similarly, each job seeker $u \in \mathcal{U}$ can be summarized as a tuple $u = (\texttt{ID}_u, \mathcal{H}_u, \texttt{meta}_u)$ with an identifier $\texttt{ID}_u$, a professional profile $\mathcal{H}_u$, and metadata attributes $\texttt{meta}_u$, like the education background or the geographical information. Specifically, the professional profile $\mathcal{H}_u$ can be represented either as his/her working experience or his/her job application record. In this work, we target at the job application record, so we denote $\mathcal{H}_u$ as a job posting sequence (session) ordered by time $\mathcal{H}_u = \{j_1, \ldots, j_n\}$[3], where each j_i is a specific job in $\mathcal{J}$. With the above notations, the *Next-Application Prediction* problem can be formally defined as follows:

Given a set of available job positions $\mathcal{J}$ and a set of job seeker $\mathcal{U}$, for each specific job seeker $u \in \mathcal{U}$ who has a job application sequence $\mathcal{H}_u = \{j_1, \ldots, j_t\}$ at one specific time step t, we want to

Predict the next most likely job j_{t+1} that the job seeker u might apply for, which means maximizing the likelihood of the conditional probability $p(j_{t+1} = j^+ | \mathcal{H}_u, \mathcal{J})$ of the true next-applied job j^+, given the application sequence $\mathcal{H}_u$. An illustration is given in Figure 6.1.

[3]We omit the superscript of u without loss of clarity.

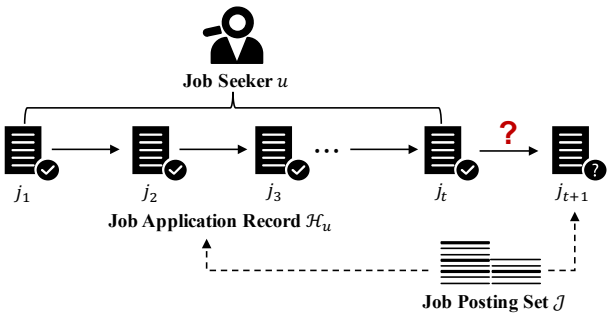


FIGURE 6.1: An illustration of the *Next-Application Prediction* problem. Given the job application record $\mathcal{H}_u = \{j_1, j_2, j_3, \ldots, j_t\}$ of job seeker u, a subset of the job posting set $\mathcal{J}$, the purpose is to predict the next job j_{t+1} he/she will apply for.

6.3.2 Proposed Model

An overview of our model, called Personalized-Attention Next-Application Prediction (PANAP), is given in Figure 6.2. It consists of three modules: (i) *Job Content Representation* is used for job representation learning, (ii) *Job Seeker Representation* is used for job seeker characterization, and (iii) *Next-Application Predictor* is used for next-application prediction. Next, we detail our model by describing these different modules.

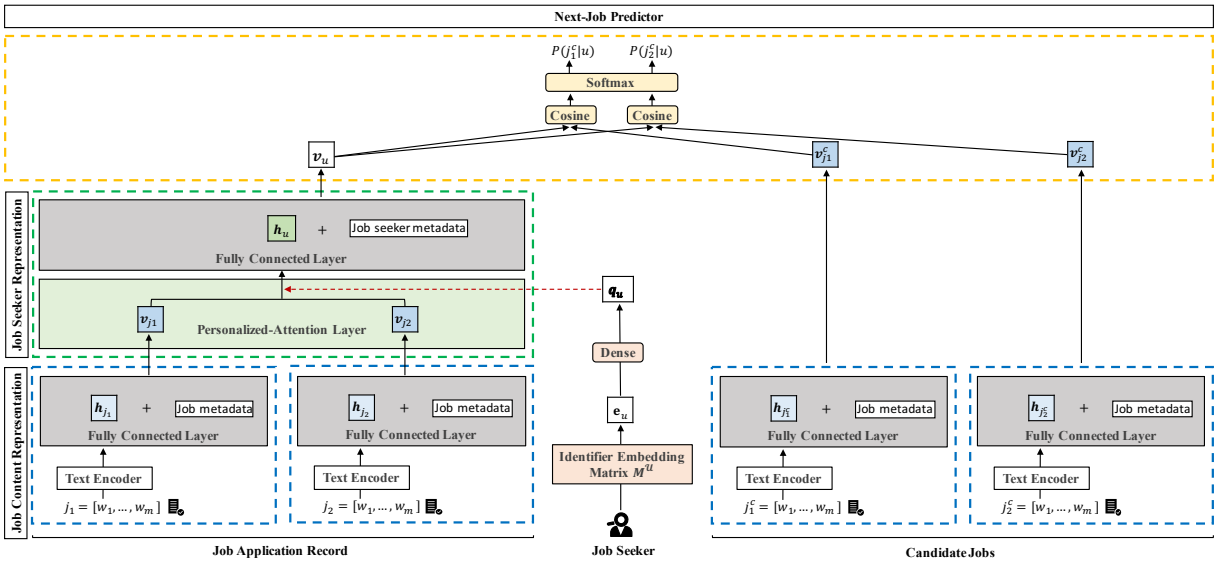


FIGURE 6.2: The proposed PANAP framework consists of three parts: (i) *Job Content Representation* (i.e., blue dashed box) is used for job content representation learning. (ii) *Job Seeker Representation* (i.e., green dashed box) uses the personalized-attention mechanism to characterize the career preference of job seekers based on the Job Application Record. (iii) *Next-Application Predictor* (i.e., yellow dashed box) utilizes a ranking loss based on the representation similarity of job and job seeker to train the model.

Job Content Representation

The first module, named JCR (Job Content Representation), is responsible for learning a distributed representation of each job $j \in \mathcal{J}$, where $j = (\texttt{ID}_j, \{w_1^j, \ldots, w_m^j\}, \texttt{meta}_j)$ as defined in Section 6.3.1. Specifically, for job j, a d-dimensional textual representation $\boldsymbol{h}_j \in \mathbb{R}^d$ is learned from its textual content $\{w_1^j, \ldots, w_m^j\}$ with a text encoder:

$$\boldsymbol{h}_j = \texttt{text\_encoder}\left(\{w_1^j, \ldots, w_m^j\}\right),$$

where the text encoder can be some unsupervised textual embedding approaches (e.g., Word2Vec [Mikolov, K. Chen, et al., 2013] and Doc2Vec [Q. Le and Mikolov, 2014]) or pre-trained models (e.g., BERT [Devlin et al., 2018]). The choice and design of the text encoder is important and can have an impact on the recommendation accuracy. We study this impact experimentally in Section 6.5.6. As explained in the motivation, we want to tackle the fact that the data in the recruitment domain often lack labels and that the recruitment domain is an evolving field, as pointed out in Issue 2 and Issue 4. Therefore, using textual information to represent jobs allows new jobs can be added easily and immediately to the database, thereby alleviating the job cold-start problem (Issue 4). Moreover, the textual job representation $\boldsymbol{h}_j$ is trained separately in an unsupervised way or generated by a pre-trained language model that can address Issue 2.

The metadata attributes $\texttt{meta}_j$ (e.g., city and state) are, respectively, embedded into vectors (e.g., $\boldsymbol{v}_j^{\texttt{city}}$ and $\boldsymbol{v}_j^{\texttt{state}}$) through different trainable embedding matrices (e.g., $\boldsymbol{M}^{\texttt{city}}$ and $\boldsymbol{M}^{\texttt{state}}$), which will be jointly learned during the training process of the model. The vector $\boldsymbol{v}_j^{\texttt{meta}}$ is then obtained by concatenation of all vectors:

$$\boldsymbol{v}_j^{\texttt{meta}} = \left[\boldsymbol{v}_j^{\texttt{city}} \oplus \boldsymbol{v}_j^{\texttt{country}} \oplus \ldots\right],$$

where the symbol $\oplus$ represents the concatenation operator. Then the textual job representation $\boldsymbol{h}_j$ and job metadata vector $\boldsymbol{v}_j^{\texttt{meta}}$ are combined by using a sequence of Fully Connected (FC) layers to produce a fixed $d^{\mathcal{J}}$-dimensional *Job Content Vector* $\boldsymbol{v}_j$. The formula is as follows:

$$\boldsymbol{v}_j = \texttt{FCs}\left([\boldsymbol{h}_j \oplus \boldsymbol{v}_j^{\texttt{meta}}]\right).$$

Job Seeker Representation

The second module, named JSR (Job Seeker Representation), is responsible for learning the distributed representation of each job seeker. It requires three inputs: the identifier $\texttt{ID}_u$ of job seeker u, his/her historical applied job sequence $\mathcal{H}_u = \{j_1, \ldots, j_n\}$,[4] and associated metadata attributes $\texttt{meta}_u$. In addition, $\mathcal{H}_u$ can be transformed as a sequence of *Job Content Vectors* $\{\boldsymbol{v}_{j_1}, \ldots, \boldsymbol{v}_{j_n}\}$, where each $\boldsymbol{v}_{j_i} \in \mathbb{R}^{d^{\mathcal{J}}}$ is obtained from the JCR module. Since the same job may have different informativeness for different job seekers, it contributes differently to characterize career profiles of job seekers. Then, we learn the job seeker representation with a personalized-attention mechanism proposed in [C. Wu et al., 2019]. We first embed the $\texttt{ID}_u$ of job seeker u into a vector $\boldsymbol{e}_u$ using an identifier embedding

[4] We omit the superscript of u without loss of clarity.

matrix $\mathbf{M}^{\mathcal{U}} \in \mathbb{R}^{|\mathcal{U}| \times d^s}$, where d^s denotes the dimension of identifier embedding. Then $\boldsymbol{e}_u$ is passed to a dense layer parameterized with $\boldsymbol{W}^q \in \mathbb{R}^{d^{\mathcal{U}} \times d^q}$ and $\boldsymbol{b}^q \in \mathbb{R}^{d^q}$ to form the preference query vector $\boldsymbol{q}_u = \texttt{ReLU}(\boldsymbol{W}^q \times \boldsymbol{e}_u + \boldsymbol{b}^q)$, where d^q is the query dimension.

The importance score of i-th job for job seeker u is calculated as follows:

$$\alpha_i = \frac{\exp\left(\boldsymbol{v}_{j_i}^T \boldsymbol{p}_u\right)}{\sum_{i'=1}^{n} \exp\left(\boldsymbol{v}_{j_{i'}}^T \boldsymbol{p}_u\right)},$$

where $\boldsymbol{p}_u = \texttt{tanh}\left(\boldsymbol{W}^a \times \boldsymbol{q}_u + \boldsymbol{b}^a\right)$, with projection parameters $\boldsymbol{W}^a \in \mathbb{R}^{d^q \times d^{\mathcal{J}}}$ and $\boldsymbol{b}^a \in \mathbb{R}^{d^{\mathcal{J}}}$, $d^{\mathcal{J}}$ is the dimension of *Job Content Vector*. Note that the attention scores might differ for the same job, as the query vector depends on the ID of the job seeker. This allows forming a weighted career preference representation $\boldsymbol{h}_u$ for job seeker u:

$$\boldsymbol{h}_u = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_{j_i}.$$

Similar to the JCR module, the metadata vector of job seeker u is represented as follows:

$$\boldsymbol{v}_u^{\texttt{meta}} = \left[\boldsymbol{v}_u^{\texttt{city}} \oplus \boldsymbol{v}_u^{\texttt{education}} \oplus \dots\right].$$

Then $\boldsymbol{v}_u^{\texttt{meta}}$ and $\boldsymbol{h}_u$ are combined by using a sequence of FC layers to produce the final *Job Seeker Vector*:

$$\boldsymbol{v}_u = \texttt{FCs}\left([\boldsymbol{h}_u \oplus \boldsymbol{v}_u^{\texttt{meta}}]\right),$$

where *Job Seeker Vector* $\boldsymbol{v}_u$ has a dimension of $d^{\mathcal{U}}$, which is equal to the dimension of *Job Content Vector* $\boldsymbol{v}_j$ (i.e., $d^{\mathcal{U}} = d^{\mathcal{J}}$).

Next-Application Predictor

In classical deep recommendation architectures, the output is usually a probability vector whose dimension is the number of available items. In our scenario of job recommendation, due to the evolving nature of the recruitment domain (Issue 4), such models are not tractable in practice. Thus this dynamic scenario needs an approach that does not need to retrain the whole network at each change. As a consequence, inspired by the highly dynamic news recommendation scenario, we use the ranking loss proposed in [Souza Pereira Moreira, Ferreira, and Cunha, 2018] to train the predictor. The principle of this loss is to train the predictor to maximize the similarity between user preferences and positive samples, while minimizing similarities with negative samples.

The idea of the ranking loss comes from DSSM [Huang et al., 2013], which is an effective document ranking model and has been leveraged for the recommendation. It uses a Deep Neural Network (DNN) structure to model a pair of text strings into a continuous semantic space and then measures the semantic similarity between the two strings. In our case, we measure the similarity between jobs and job seekers. More precisely, given the application history $\mathcal{H}_u = \{j_1, \dots, j_n\}$ of job seeker u, we formulate the above representation generating processes as $\boldsymbol{v}_u = \texttt{JSR}(\texttt{ID}_u, \mathcal{H}_u, \texttt{meta}_u; \Theta_1)$, where each $j_i \in \mathcal{H}_u$ can be embedded into a *Job Content Vector* $\boldsymbol{v}_{j_i}$ through $\texttt{JCR}(j_i; \Theta_2)$, Θ_1 and Θ_2 are model parameters. $\boldsymbol{v}_u$ and $\boldsymbol{v}_{j_i}$ are vectors of the same dimension, and we thus can define the relevance

score between job seeker u and job j by the cosine similarity of their representations:

$$\texttt{sim}(u, j_i) = \cos(\boldsymbol{v}_u, \boldsymbol{v}_{j_i}) = \frac{\boldsymbol{v}_u \cdot \boldsymbol{v}_{j_i}}{\|\boldsymbol{v}_u\| \|\boldsymbol{v}_{j_i}\|}.$$

Due to the huge number of job postings in the database, it is impractical to compare the job seeker u with all jobs one by one. A common strategy is to sample a set of jobs that u did not apply during his/her active session as "negative" samples, denoted as $\mathcal{J}_u^-$. Thus, the prediction problem can be treated as a $|\mathcal{J}_u^-| + 1$-class classification problem ($|\mathcal{J}_u^-|$ negative jobs and a positive j_u^+), the goal is to maximize the probability that the true next-application j_u^+ is predicted while minimizing the probability of negative jobs $j^- \in \mathcal{J}_u^-$. The probability of $j \in \mathcal{J}_u^- \cup j_u^+$ being the next job for u is formulated as follows:

$$Pr(j|u) = \frac{\exp\left(\texttt{sim}(u, j)\right)}{\sum_{j' \in j_u^+ \cup \mathcal{J}_u^-} \exp\left(\texttt{sim}(u, j')\right)},$$

The recommender should learn to maximize the similarity between the content vector $\boldsymbol{v}_{j_u^+}$ of positive job j_u^+ and the job seeker preference vector $\boldsymbol{v}_u$, while minimizing similarities with job vector $\boldsymbol{v}_{j^-}$ in the negative set $\mathcal{J}_u^-$. Thus, the loss function is defined by

$$\mathcal{L} = -\log \prod_{(u,j^+)} Pr\left(j^+ | \texttt{ID}_u, \mathcal{H}_u, \texttt{meta}_u; \Theta_1, \Theta_2\right).$$

With this loss function, a newly published job posting can be immediately recommended as soon as its representation is encoded, thus solving Issue 4. Furthermore, we consider metadata attributes of jobs and job seekers in JCR and JSR, respectively, when modeling representations. These representations contain more context information, which will help improve the recommendation quality (Issue 5).

Negative Sampling Strategies

Since our method is trained and evaluated with negative samples as described in Section 6.3.2, the negative sample set $\mathcal{J}^-$[5] significantly influences the model performance. A well-used sampling strategy is the mini-batch based sampling proposed in [Hidasi, Karatzoglou, et al., 2015], which treats the items from the other training/evaluation sessions in the same mini-batch as negative samples. [Hidasi and Karatzoglou, 2018] extends the mini-batch based strategy by adding additional samples (based on unity or based on popularity). In the scenario of news recommendation, [Souza Pereira Moreira, Ferreira, and Cunha, 2018; Gabriel De Souza, Jannach, and Da Cunha, 2019] uniformly sample additional samples from a global buffer of the N most recently interacted items. For some applications, such as news, music, and video, popularity is also an essential factor that influences user choice besides personal preference. Therefore, sampling based on popularity is a good sampling strategy, as described in Section 3.2.2. However, in the recruitment field, unlike these applications, the choices of job seekers are more influenced by their personal contexts, such as the geographic location factor. When job seekers make a choice, they usually first need to consider the job location. They are more likely to apply for jobs in their cities or other cities not far from their current locations (i.e., cities in the same state). We will prove this observation in Section 6.4.1.

[5] We omit the superscript of u without loss of clarity.

In order to solve this problem, our PANAP method cooperates with the "location" metadata attributes (e.g., city, state, and country) of jobs and job seekers to model their representations, respectively. As a consequence, these representations contain "location" information. In addition, considering the "location" when generating negative samples can enable our model to learn useful information for more meaningful recommendations. As a consequence, we propose and evaluate different sampling strategies in Section 6.4.2.

6.4 Experiments

6.4.1 Datasets

We employ *CareerBuilder12*, an open dataset from Kaggle competition,[6] to evaluate our method.[7] Its statistics are given in Table 6.2. It consists of job applications for almost 13 weeks. In this dataset, job seeker has five metadata: *City*, *State*, *Country*, *Degree*, and *Major*. Job metadata are *City*, *State* and *Country*. The textual job content includes a *Job Title*, a *Job Description* and some *Job Requirements*. From this initial dataset, we created two datasets: (i) *CB12_s* like in [Lacic et al., 2020], in which sessions are created via a time-based split of 30 min inactivity threshold, and we discarded sessions with less than two applications for next job prediction purpose. (ii) *CB12_l* uses all application records during 13 weeks to model the career profile of each job seeker. Thus, *CB12_l* has longer sequences that enable us to evaluate the effectiveness of our method. We further split the last 14 days for testing and the remaining sessions for training. Like [Lacic et al., 2020], we filter job postings in the test set that do not belong to the training set as this enables a better comparison with the approaches, which can only recommend items that have been used to train the model.

TABLE 6.2: Statistics of datasets, $|\mathcal{S}|$ is the total number of sessions, and $|\mathcal{A}|$ is the total number of applications. $|\mathcal{H}_u|_{\text{avg}}$ is the average application number in sessions. $|Metadata|$ contains the cardinality of each metadata attribute.

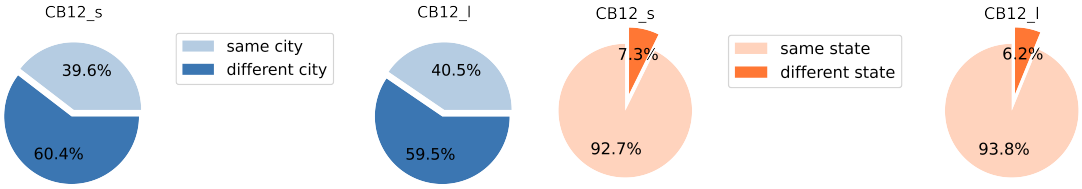
Dataset	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{J}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{H}_u\rvert_{\text{avg}}$	$\lvert Metadata\rvert$ City/State/Country/Degree/Major
CB12_s	111,785	207,972	165,027	638,469	3.87	8,226/122/33/7/21,224
CB12_l	137,642	239,581	137,642	772,305	5.61	8,856/130/37/7/25,201

As we described in Section 6.3.2, the "location" is an essential factor that needs to be considered in the job recommendation. In order to further prove this observation, in Figure 6.3, we illustrate the relationship between locations of job seekers and applied jobs in *CareerBuilder12* datasets. As shown in Figure 6.3b, most job seekers (92.7% in CB12_s and 93.8% in CB12_l) have applied for jobs in their states. Among these people, 81.3% and 79.7% of job seekers only consider jobs in their own states. Figure 6.3a shows that only 39.6% and 40.5% of job seekers apply for jobs in their own

[6] https://www.kaggle.com/c/job-recommendation

[7] Another similar dataset in the recruitment domain is the RecSys17 dataset provided by XING after the RecSys Challenge 2017 [Abel et al., 2017], but it is no longer available.

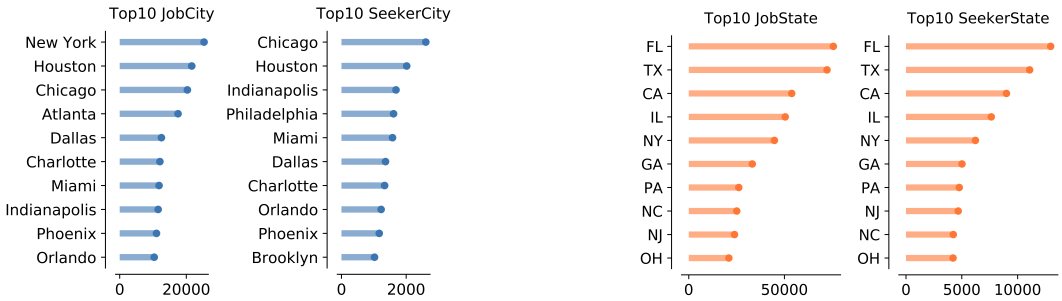
cities because job opportunities in their cities are usually limited. As explained above, people are also applying for jobs in other cities in the same state. Therefore, the "location" (the job site or the current location of the job seeker) is an essential factor to be considered in the job recommendation.



(a) City (b) State

FIGURE 6.3: The relationship between job seeker location and the applied job location in *CareerBuilder12* datasets. Most job seekers (92.7% in *CB12_s* and 93.8% in *CB12_l*) have applied for jobs in their states. Of these, 81.3% and 79.7% of job seekers only considered jobs in their own states.

In addition, we list the Top10 *SeekerCity*s and *SeekerState*s in Figure 6.4a and Figure 6.4b, respectively, along with the Top10 *JobCity*s and *JobState*s with the most applications by job seekers. From the figures, we can see that *JobState*/*JobCity* and *SeekerState*/*SeekerCity* have a large overlap, which also proves that "location" affects the choices of job seekers.



(a) Top10 City. (b) Top10 State.

FIGURE 6.4: The Top10 city (i.e., *JobCity* and *SeekerCity*) and Top10 state (i.e., *JobState* and *SeekerState*) in *CB12_s* dataset.

6.4.2 Experimental Settings

Since PANAP consists of three modules, we explain the experimental settings in three parts.

To obtain job representation $\boldsymbol{h}_j$ in JCR, we first tokenize job postings as words then remove stop words, punctuation, and numbers. In this experiment, $\boldsymbol{h}_j$ were obtained by Doc2Vec with dimension $d = 300$ via the distributed memory. We also compared different text encoders in Section 6.5.6.

In JSR, the dimension d^s of identifier embedding $\boldsymbol{e}_u$ and the query dimension d^q were set to 100, and the dimension $d^{\mathcal{U}}$ and $d^{\mathcal{J}}$ were set to 300. We applied the dropout technique with a rate of 0.2 to each layer and L2 regularization with rate 1e-4 to parameter weights. Metadata attributes with

low cardinality (i.e., ≤ 10) were one-hot encoded, and high cardinality attributes were represented as trainable embeddings. We used a sequence of two FC layers in our settings, with Leaky ReLU [Maas, Hannun, and Ng, 2013] and tanh activation functions to combine metadata vectors, respectively.

PANAP is trained and evaluated with a 256 mini-batch size, and we use the Adam [Kingma and Ba, 2014] optimizer with a learning rate of 5e-4. Our model is trained and evaluated with negative samples, so we used 15 negative training samples for training and 50 for evaluation, chosen according to the best performance of the model. We further discuss the impact of the number of negative samples on model performance in Section 6.5.5. In our experiments, we first explore a sampling strategy inspired by [Souza Pereira Moreira, Ferreira, and Cunha, 2018; Hidasi and Karatzoglou, 2018], and then propose an improved mini-batch based sampling strategy for the job recommendation scenario. This improved strategy takes into account the current geographic location of the job seeker and the job site when sampling negative jobs. The two strategies are as follows:

- **Strategy 1 (S1)**-*mini-batch + additional samples*: It is proposed in [Souza Pereira Moreira, Ferreira, and Cunha, 2018; Gabriel De Souza, Jannach, and Da Cunha, 2019] for news recommendation. We adapt this strategy to the context of job recommendation discussed in this thesis, i.e., when there are not enough negative samples within the mini-batch, additional samples are added with a uniform sampling strategy from a global buffer of the N most recently applied jobs. Jobs are then uniformly sampled from the "candidate set" (i.e., jobs within the mini-batch and additional jobs).

- **Strategy 2 (S2)**-*mini-batch + additional samples + location-based*: Different from Strategy 1, in Strategy 2, we first select the jobs in the same state as the job seeker from the "candidate set" as negative samples.[8] When there are not enough negative samples in the current state, we sample jobs in other states from the "candidate set".

We evaluated PANAP by comparing it with the following baselines, details of each method can be found in Section 3.2.2.

- *POP*: It is a simple and often strong baseline that recommends the most popular (i.e., most applied) job postings.

- *Association Rule Learning (ARL)*: It is a simplified version of association rule [Agrawal, Imieliński, and Arun Swami, 1993] with a maximum rule size of two.

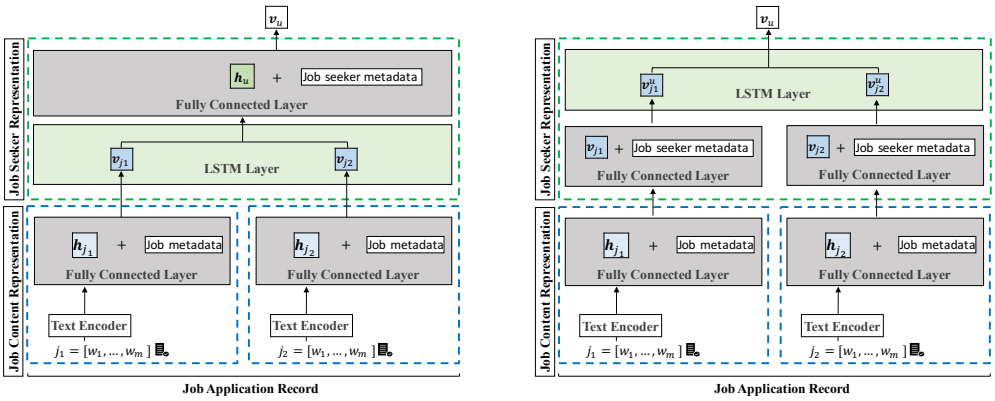
- *Content Similarity (CS)*: It recommends similar jobs based on the cosine similarity between representations of each applied job and the N most recently applied jobs in the global buffer.

- *Item-based K-Nearest Neighbors (IKNN)*: It recommends jobs that are similar to the last applied job during the current session as in [Hidasi, Karatzoglou, et al., 2015].

- *Session-based K-Nearest Neighbors (SKNN)* [Jannach and Ludewig, 2017]: It compares the entire current session with the past sessions in the training dataset, rather than considering only the last job in the current session.

[8] As shown in Figure 6.3 people are more likely to apply for jobs where they are (i.e., the same city or the same state). Since job opportunities in their cities are usually limited, as shown in Figure 6.3a, here we sample jobs in the same state.

- *Vector multiplication Session-based KNN (V-SKNN)* [Ludewig and Jannach, 2018]: It is a variant of *SKNN* that emphasizes jobs more recently interacted within the current session, when computing the similarities with past sessions. For this, a linear decay function is used that depends on the position of the job within the session.
- *VAE_Comb*: It is the best-performing model in the CareerBuilder12 dataset proposed in [Lacic et al., 2020], which uses a variational autoencoder to encode the job application session and the job content, and then recommends the next-job based on the resulting session representation using a K-Nearest Neighbor manner.
- *GRU4Rec*: It is a model that was specifically designed for sequential recommendation scenarios. For this experiment, we use the most recent version of *GRU4Rec* [Hidasi and Karatzoglou, 2018].
- *BERT4Rec* [Sun et al., 2019]: It introduces the bi-directional self-attention model to model user behavior sequences.
- Two variants of PANAP:
  - *LSTM*: It replaces the personalized-attention layer with a LSTM layer to verify the effectiveness of personalized attention, as shown in Figure 6.5a,
  - *PLSTM*: It adds job seeker metadata to each job representation to generate *Personalized job embedding* before the LSTM layer, similar to [Souza Pereira Moreira, Ferreira, and Cunha, 2018], in order to verify the validity of the job seeker metadata and choose an appropriate position to add the metadata, as shown in Figure 6.5b.



(a) LSTM (b) PLSTM

FIGURE 6.5: Two variants of PANAP.

The evaluation metrics used in this work are the accuracy metric Hit Rate (HR@5), and two popular ranking metrics, Mean Reciprocal Rank (MRR@5) and Normalized Discounted Cumulative Gain (NDCG@5). We quickly review these metrics in Table 6.3, for more details, see Section 3.2.2.

TABLE 6.3: A quick review of the evaluation metrics used in *Next-Application Prediction*.

Metric	Idea
HR	the more ground-truth items included in the recommended items (i.e., the larger the value of HR), the better the model
MRR	the higher the ranking of ground-truth items among the recommended items (i.e., the larger the value of MRR), the better the model
NDCG	the higher the ranking of ground-truth items among the recommended items, the more rewards they get (i.e., the larger the value of NDCG), and the better the model

6.5 Results

In this section, we present our experimental results. For all tables shown in this section, scores in bold are the best in each metric, and score in the underline are the second best.

6.5.1 Next-Application Prediction

Table 6.4 summarizes the best results (HR@5/MRR@5/NDCG@5) for next-application prediction among all methods. According to the results, we have the following observations:

(i) Among all models, POP and CS give the lowerest scores, as they neither model the personalized preference nor consider the sequential information. Although POP is often a strong baseline in certain domains, i.e., news and movies, career preferences are less affected by popularity factors in the recruitment domain.

(ii) Overall, DL-based methods consistently outperform traditional methods, demonstrating that Neural Networks are good at modeling sequential information, and the self-attention mechanism can improve accuracy.

(iii) Our proposed method $PANAP(S1)$, $PANAP(S2)$ and its variants $LSTM(S2)$ and $PLSTM(S2)$ perform best among all baselines, which indicates the job content and metadata can effectively improve the recommendation performance. We detail this point in Section 6.5.3.

(iv) $PANAP(S1)$ and $PANAP(S2)$ outperform $LSTM(S2)$ and $PLSTM(S2)$, which use LSTM to model the sequential information. One possible reason is that career preferences are less dynamic in the recruitment domain than in other domains, and application sequences are relatively short (i.e., 3.87 and 5.61 on average for both sets). Thus, the advantage of RNN can not be well demonstrated. This result also demonstrates the advantage of personalized-attention as different jobs might have different importance for career preference modeling, and selecting the more critical jobs is useful for achieving better recommendation performance. Moreover, $LSTM(S2)$ is better than $PLSTM(S2)$. One possible reason is that $PLSTM(S2)$ merges the job seeker metadata into each job in the session, which may weaken the information carried by the job itself, and add some noise (i.e., the location information of job seeker).

(v) $PANAP\ (S1)$ with sampling strategy S1 performs better than $PANAP(S2)$ with sampling strategy S2. The reason will be discussed in Section 6.5.4.

Since similar results can be seen from the *CB12_s* and *CB12_l* datasets, all experiments below are performed on the *CB12_s* dataset.

TABLE 6.4: *Next-application prediction* results, where metrics are HR@5/MRR@5/NDCG@5.

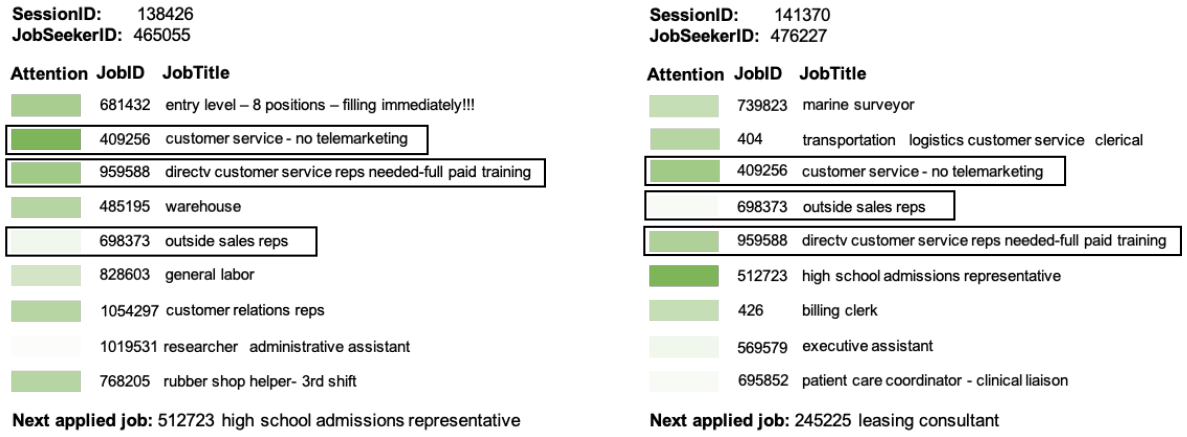
Method	CB12_s	CB12_l
POP	0.089/0.042/0.054	0.093/0.046/0.058
AR	0.273/0.200/0.218	0.202/0.146/0.160
CS	0.226/0.128/0.152	0.214/0.118/0.142
IkNN	0.274/0.202/0.220	0.202/0.147/0.161
SkNN	0.349/0.247/0.272	0.276/0.196/0.216
V-SkNN	0.349/0.248/0.273	0.276/0.196/0.217
VAE_Comb	0.345/0.239/0.256	0.282/0.203/0.224
GRU4Rec	0.367/0.208/0.230	0.403/0.309/0.352
BERT4Rec	0.373/0.232/0.245	0.438/0.311/0.369
PANAP(S1)	**0.756/0.620/0.680**	**0.807/0.671/0.733**
PANAP(S2)	0.691/0.492/0.541	0.742/0.543/0.593
LSTM(S2)	0.540/0.339/0.389	0.569/0.368/0.418
PLSTM(S2)	0.480/0.292/0.338	0.516/0.323/0.371

TABLE 6.5: Performance comparison of different attention mechanisms in CB12_s dataset. All models are trained with S2.

Attention_mechanism	HR@5/MRR@5/NDCG@5
Personalized-attention	**0.691/0.492/0.541**
Vanilla attention	0.685/0.480/0.531
No attention (avg)	0.565/0.367/0.416
LSTM(S2)	0.540/0.339/0.389

6.5.2 Effectiveness of Personalized Attention

In this part, we use *CB12_s* set to analyze the effectiveness of the personalized-attention mechanism in $PANAP(S2)$. As shown in Table 6.5, the models with attention mechanism consistently outperform the model without attention, and our model with the personalized-attention outperforms its variant with vanilla attention (similar to the global attention used in [Meng et al., 2019]). Such a result is probably since the vanilla attention uses a fixed query vector and cannot adjust to different personal preferences. In order to validate that the personalized-attention mechanism is able to select informative jobs in the applied job sequence to characterize the career preference of job seekers, we visualize the corresponding attention scores of the applied jobs obtained by the personalized-attention mechanism in two example sessions in Figure 6.6.



(a) Sampled session 1

(b) Sampled session 2

FIGURE 6.6: Two sampled application sequences with attention weights captured from the personalized-attention mechanism. Darker colors indicate higher attention weights.

Each row is a job application record, including the attention score (i.e., darker colors indicate higher attention weights), the corresponding job identifier $\texttt{ID}_j$ and job title. Note that, for each job j, we treat its title, description, and requirements as the textual content, as explained in Section 6.4.1, which contains hundreds of words. Therefore, we only provide job identifiers and job titles, and darker colors indicate higher attention weights. From Figure 6.6, we find that our model can model the different informativeness of jobs to learn the preferences of different job seekers. For example, compared to the job seeker in Figure 6.6b, the job seeker in Figure 6.6a pays more attention to the same job 409256 and job 959588, and both pay less attention to job 698373. So the predicted next-applications are different.

6.5.3 Effectiveness of Different Features

In particular, we are interested in the influence on recommendation performance of using additional available data, including job textual content and metadata. For this, we use the *CB12_s* dataset to study their effect on our two best models: $PANAP(S2)$ and $LSTM(S2)$. As shown in Table 6.6, (i) it is obvious that methods with the content representation or metadata generally give better results than what is achievable from the job identifier $\texttt{ID}_j$ only (i.e., *Only_JobID*). (ii) The metadata has a fewer influence on $PANAP(S2)$ than $LSTM(S2)$ (29.1% average reduction between *Meta+Content+JobID* and *No_Meta* on three metrics compared to 41.0%). One possible reason is that $PANAP(S2)$ utilizes a personalized-attention mechanism to model career preferences, which already contain personal information. (iii) Since the "location" factor affects the choice of job seekers, our sampling strategy is location-based. The job metadata, e.g., *City*, *State*, and *Country* could give more relevant information about "location", so the scores of *No_JobMeta* are lower than that of *No_SeekerMeta*. (iv) $PANAP(S2)$ consistently outperforms $LSTM(S2)$, even only the job identifier $\texttt{ID}_j$ is used, which indicates that $PANAP(S2)$ does capture personal preference. This observation also proves the advantage of the personalized-attention mechanism in cases where no additional information is available.

TABLE 6.6: Performance comparison of different feature combinations, where metrics are HR@5/MRR@5/NDCG@5. *No_Meta* means that neither the job seeker metadata nor the job metadata is considered.

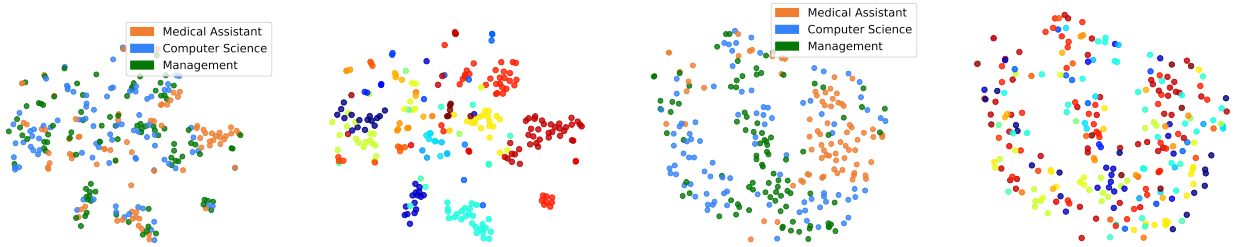
Feature	PANAP(S2)	LSTM(S2)
Meta+Content+JobID	**0.691/0.492/0.541**	0.540/0.339/0.389
No_Meta	0.516/0.334/0.379	0.341/0.189/0.226
No_Content	0.530/0.341/0.386	0.449/0.286/0.326
Only_JobID	0.501/0.312/0.355	0.331/0.182/0.218
No_JobMeta	0.511/0.338/0.381	0.377/0.212/0.253
No_SeekerMeta	0.601/0.399/0.449	0.505/0.297/0.349

TABLE 6.7: Performance comparison of different negative sampling strategies in CB12_s.

Strategy	HR@5/MRR@5/NDCG@5
PANAP(S1)	**0.756/0.620/0.680**
PANAP(S2)	0.691/0.492/0.541

6.5.4 Negative Sampling Analysis

This section examines the performance of the two sampling strategies described in Section 6.4.2 and identifies the importance of the "location" factor when generating negative samples on *CB12_s* dataset. According to the results, as shown in Table 6.7, $PANAP(S1)$ has a better performance than $PANAP(S2)$. To explain the result, we visualize job seeker representations (session representations) generated from $PANAP(S1)$ and $PANAP(S2)$ in Figure 6.7. We use t-SNE [Van der Maaten and Hinton, 2008] to reduce the representation dimension. For illustration purposes, we categorize each job seeker according to his/her *Major*. Here, *Major* denotes a specific subject that a student studies at a college or university, such as "Journalism", "Marketing" and "Nursing". Note that categorizing job seekers through *Major* is not the most reasonable way because some job seekers are currently engaged in occupations that do not match their majors. Thus, we select three non-similar majors. People with these professional backgrounds are more likely to engage in related jobs, including *Management*, *Computer Science* and *Medical Assistant*. We also plot job seeker representations labeled with *State* in Figure 6.7b and Figure 6.7d to show the influences of different sampling strategies. Each color corresponds to one *State*.



(a) S1: *Major* (b) S1: *State* (c) S2: *Major* (d) S2: *State*

FIGURE 6.7: Visualization of career preference representations of job seeker generated from *PANAP(S1)* and *PANAP(S2)*.

We observe from Figure 6.7b that representations learned by our model $PANAP(S1)$ with S1 are well-clustered into groups, and each corresponds to a *State*. This observation can be used to explain why the accuracy scores of $PANAP(S1)$ are better than that of $PANAP(S2)$. A reasonable explanation is that S1 does not consider the "location" factor when generating negative samples. More specifically, as mentioned in Section 6.3.2, job seekers are more likely to apply for jobs located in their cities or other cities in the same state. The two main reasons why a job seeker does not apply for a job are as follows: (i) the job content is inappropriate, or (ii) the work location is not suitable. If most of the negative samples are job postings in other states, regardless of the job content, these negative samples tend to force the model to capture subtle information between different states rather than different job contents. For instance, as illustrated in Figure 6.8, a job seeker in *Seattle-Washington* has a background of *computer science* (i.e., education background, skills and work experience). There are two negative samples: *sales representative* and *java developer* from *Miami-Florida*. They are negative because Seattle is not in Florida state.

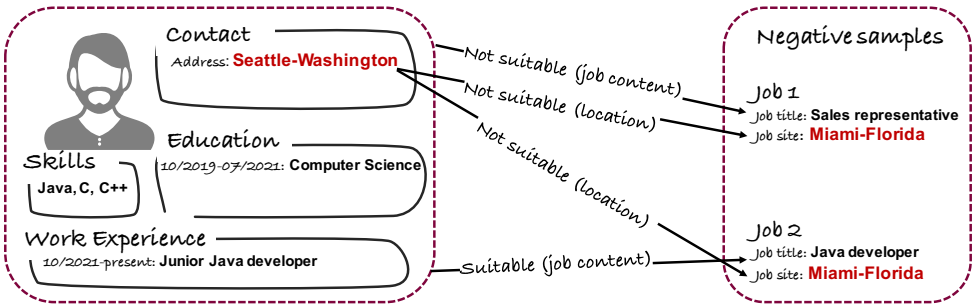


FIGURE 6.8: Example of negative sample sampling without regard to "location".

Due to the "location" factor, this job seeker may not apply for *java developer*, while *sales representative* is not suitable from both "location" and job content perspectives. With these negative samples, the model learns to distinguish between locations (i.e., *Seattle-Washington* and *Miami-Florida*) rather than the contents of jobs (i.e., *computer science* and *sales representative*). As a result, the learned job seeker representations do not capture job content information, so they are not categorized according to their *Major* categories, as shown in Figure 6.7a. Instead, they are grouped together by the "location" as in Figure 6.7b. However, the job content is one of the most important pieces of information for recruitment, and the goal of many works centers on learning how to model it properly, such as the Person-Job Fit task [Zhu et al., 2018; Qin et al., 2018] and job classification task [Jingya Wang et al., 2019]. To handle this problem, we first include location-related metadata, such as *City*, *State* and *Country* into the representation learning scheme. We then propose a location-based sampling strategy, S2, as described in Section 6.4.2, which prioritizes jobs in the same state as the job seeker as negative samples. We visualize the representations learned through S2 in Figures 6.7c and 6.7d to show the promising results of our proposed strategy. We observe that the job seekers are grouped by *Major* in Figure 6.7c, which demonstrates the effectiveness of S2.

In order to emphasize our findings, we further use a classification task to compare the representations obtained from S1 and S2. We use *Major* as the label of job seeker representations, although in practice, this is not always the case because some job seekers are currently engaged in occupations that do not match their majors. Specifically, we first select five non-similar major categories, including *Psychology*, *Criminal Justice*, *Management*, *Computer Science*, and *Medical Assistant*. Then, we randomly sampled 500 job seeker representations for each major from the training set to train a logistic regression classifier and use the representations of the test set for evaluation. Macro-F1 is used to compare performance.

From the classification results on the *CB12_s* dataset, as shown in Table 6.8, we can observe that the representation learned by $PANAP(S2)$ achieves relatively higher *Major* classification accuracy than $PANAP(S1)$. This proves that the negative samples obtained through S2 provide more relevant information about the job content. Both $PANAP(S1)$ and $PANAP(S2)$ have remarkable classification accuracy advantages over *Avg_Vec* method, which uses the average vector of applied jobs to represent job seekers. This result also demonstrates the effectiveness of our proposed method in the characterization of job seekers.

TABLE 6.8: Performance comparison of representations learned by different negative sampling strategies through the classification task.

Sampling_strategy	Macro-F1
PANAP(S1)	0.822
PANAP(S2)	**0.896**
Avg_Vec	0.331

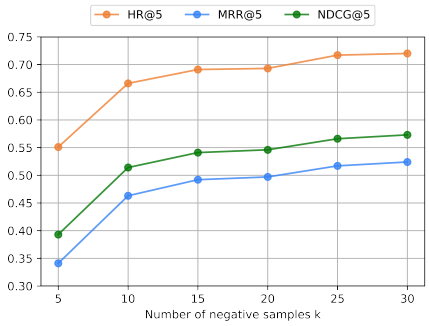


FIGURE 6.9: Performance comparison of different numbers of training negative samples k in CB12_s.

6.5.5 Number of Negative Samples

This section discusses the influence of the number of negative samples used for training on prediction accuracy. The experimental results on *CB12_s* dataset are shown in Figure 6.9, where k negative samples are used for training and 50 samples for evaluation. As explained in Section 6.4.2, the number of negative samples is chosen according to the best performance of the model. In this part, we will discuss this more. When k increases (e.g., from 5 to 10), the performance of our model first has a noticeable improvement. This may be because when k is too small, the information provided by negative samples is relatively limited, and the model cannot learn useful information. Then, when k continues to increase (e.g., from 10 to 20), the performance becomes stable. However, if k is too large, there may not be enough jobs in the same state. Negative samples in other states become dominant, making it difficult for the model to identify the valuable information correctly. As explained in Section 6.5.4, the model will learn to distinguish the difference of locations, just like $PANAP(S1)$. As a result, the performance consistently improves. Therefore, we choose 15 as the number of negative training samples.

6.5.6 Effectiveness of Different Text Encoders

This section analyzes the impact of different text encoders on the final performance of the model. The different encoders that were tested are listed as follows:

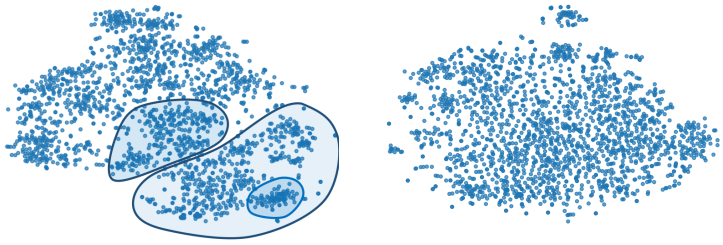
- *Word2Vec*$(W2V)$ [Mikolov, Sutskever, et al., 2013]: It is a distributed representation of words based on training a model to predict the current word from surrounding context words (CBOW). In this method, the job text content is represented as the average of its word embeddings. For this experiment, we tried a pre-trained language model, pre-trained on Google News[9] (i.e., $W2V\_P$) and also trained our word embedding model using all job postings (i.e., $W2V\_O$). The dimension of the job representation is set to 300, and if the number of words exceeds 100,000, we only keep words that occur frequently (i.e., at least ten times) to build the vocabulary.

[9] https://code.google.com/archive/p/word2vec/

- *TF-IDF*W2V*: It is a weighted version of *W2V*, where each word vector is weighted by its corresponding TF-IDF score. We also consider the same two word embedding models as *W2V* (i.e., *TF-IDF*W2V_P* and *TF-IDF*W2V_O*).
- *FastText*: It is another word embedding method that extends Word2Vec by representing each word as an n-gram of characters. For this experiment, we used a 300-dimensional pre-trained model [Grave et al., 2018] (i.e., *FastText_P*), and other experimental settings were the same as *W2V*.
- *TF-IDF*FastText*: It is a weighted version of *FastText* (i.e., *TF-IDF*FastText_P*).
- *Doc2Vec* (*D2V*) [Q. Le and Mikolov, 2014]: It is an extension of Word2Vec that can learn fixed-length feature representations from variable-length pieces of text. In this experiment, the job representation was obtained through a distributed memory model with a dimension equal to 300.
- *BERT* [Devlin et al., 2018]: It is a large-scale Transformer-based language representation model. In this experiment, the job representation is obtained using the average of the second last layer of the pre-trained model BERT-Base-Uncased, which is pre-trained with the entire English Wikipedia and the Brown Corpus.

TABLE 6.9: Performance comparison of different text encoders in *CB12_s* dataset, where metrics are HR@5/MRR@5/NDCG@5.

Encoder	PANAP(S2)
W2V_P	0.680/0.477/0.528
W2V_O	0.760/0.572/0.619
TF-IDF*W2V_P	0.708/0.501/0.552
TF-IDF*W2V_O	**0.765/0.577/0.624**
D2V	0.691/0.492/0.541
FastText_P	0.681/0.487/0.535
TF-IDF*FastText_P	0.679/0.466/0.519
BERT	0.710/0.507/0.559



(a) TF-IDF*W2V_O (b) TF-IDF*FastText_P

FIGURE 6.10: Visualization of job representations obtained by different text encoders.

The results shown in Figure 6.8 confirm that the choice of text encoder can significantly affect the recommendation accuracy. Not surprisingly, word vectors trained on our job postings lead to better results as the out-of-vocabulary issue is handled to some extent. Adding TF-IDF information usually improves prediction accuracy. More advanced models are not necessarily the best (e.g., *TF-IDF*FastText_P* compared to *TF-IDF*W2V_P*). It is worth noting that *BERT* does not greatly or consistently outperform simpler methods such as weighted average word vectors (i.e., *TF-IDF*W2V_P*), and even inferior to *W2V_O* and *TF-IDF*W2V_O*, which use vectors trained on job data.

To get a rough idea of the relationship between the learned job representation and model performance, we randomly select 2,000 jobs and plot their representations (the dimension is reduced by t-SNE) obtained by different encoders in Figure 6.10. Here we use *TF-IDF*W2V_O* and *TF-IDF*FastText_P* two encoders, as they have the best and worst performance, respectively, as shown in Table 6.9. Obviously, the job representations shown in Figure 6.9a are better grouped together than in Figure 6.9b, which can be used to explain why *TF-IDF*W2V_O* outperforms *TF-IDF*FastText_P*. Therefore, a suitable text encoder can capture job content well (i.e., occupation type and field), leading to better prediction results.

6.5.7 Effectiveness of Different Sections of Job Content

For the experiments above, the job representation was obtained on all textual contents, including *Job Title*, *Job Description*, and *Job Requirements*, as described in Section 6.4.1. In fact, different job sections carry different amounts of information, so here we discuss the effectiveness of different sections on the recommendation results. Since job postings may have the same job title/description/requirements, we only keep jobs where *Job Title*, *Job Description* and *Job Requirements* all differ. Sessions are then created in the same way as *CB12_s* described in Section 6.4.1, whose statistics are given in Table 6.10.

TABLE 6.10: Statistics of *CB12_d*, $|\mathcal{S}|$ is the total number of sessions, and $|\mathcal{A}|$ is the total number of applications. $|\mathcal{H}_u|_{\text{avg}}$ is the average application number in sessions. $|Metadata|$ contains the cardinality of each metadata attribute.

Dataset	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{J}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{H}_u\rvert_{\text{avg}}$	$\lvert Metadata\rvert$ City/State/Country/Degree/Major
CB12_d	40,456	52,803	51,419	164,553	3.20	5,276/97/22/7/9,513

Overall, we can observe from Table 6.11 that different job sections lead to different recommendation results. (i) Comparing to *Description* and *Requirements*, modeling the job with just *Job Title* yields the best results. One possible reason is that, although the job title is a short string, it often carries rich information about the job and can be used to characterize job postings. (ii) Modeling with *Job Requirements*, which describe the qualifications required for the job, such as skills, education level, work experience, and language, has poor performance compared to *Job Description*. One possible reason is that there are complex relationships between jobs and required qualifications, so we need a more appropriate method that better captures these relationships, e.g., one that better captures dependencies between skills within the same job and model the similarities between skills at different granularities. (iii) *All* generally has the best performance, as it combines all the information. Moreover, in practice, it is impractical to use only one section, i.e., *Job Title*, *Job Description* or *Job Requirements* to represent job postings, as they are often the same across different job postings. For example, for the same job position in different companies, they may use the same job title, but the job descriptions are different.

TABLE 6.11: Performance comparison of different job content sections in *CB12_d* for *PANAP(S2)*, where metrics are HR@5/MRR@5/NDCG@5.

Section	TF-IDF*W2V_P	TF-IDF*FastText_P	D2V
Title	0.577/0.362/0.415	0.505/0.298/0.349	**0.515/0.309/0.360**
Description	0.552/0.337/0.390	0.490/0.283/0.334	0.479/0.269/0.320
Requirements	0.473/0.270/0.320	0.433/0.252/0.297	0.453/0.253/0.302
All	**0.594/0.369/0.425**	**0.520/0.305/0.358**	0.477/0.276/0.326

6.6 Conclusion and Perspectives

In this task, we mainly address the job recommendation problem under the *Next-Application Prediction* task, which is one of the essential applications of job recommender systems. In response to the issues in existing works, we proposed a personalized-attention model named Personalized-Attention Next-Application Prediction model (PANAP) to answer these issues partially. More specifically, we independently learn job posting representations in an unsupervised manner to address the lack of labeled data in the recruitment domain. Then we use the personalized-attention mechanism to capture specific job information for different job seekers when modeling their career preferences since the same job has different informativeness for different job seekers. The experiments confirm that, by incorporating the personalized-attention, our method can better capture the personal career preference than baseline methods. Finally, our next-application predictor is trained using a similarity-based loss function in response to job postings that increase or decrease over time. In addition, the personal context of the job seeker is a vital factor to consider in the recruitment field, especially the location information, which will influence the job selection of job seekers. Therefore, we incorporate location information when modeling job and job seeker representations, and we propose a location-based sampling strategy that takes the "location" factor into account when generating negative samples.

Extensive experiments were conducted to discuss the advantage of personalized-attention mechanism, the effect of different negative sampling strategies, the importance of incorporating job content information and metadata in the recruitment domain, the effectiveness of different document representation methods, and the amount of information carried by different job posting sections. Based on the experimental results, we can now answer the Research Questions (RQs) proposed in Section 6.2.

- **RQ1**: In the recruitment field, do we really need the sequential recommendation method?
  Answer: *Yes, methods that model application sequences are significantly better than methods that do not rely on application sequences for the recommendation, such as POP, ARL, and CS.*

- **RQ2**: Can the personalized-attention mechanism better capture the personal career preference?
  Answer: *Yes, our model with the personalized-attention outperforms its variant without attention or only with vanilla attention. Furthermore, attention scores can serve as an explanation that different scores represent the different importance of each job for modeling the job seeker profile.*

- **RQ3**: Are job textual information and context information (e.g., geographical location and educational background) important in job recommendation?
  Answer: *Yes, although they have different effects on different models, in general, combining job content and context information can effectively improve the recommendation quality.*

- **RQ4**: How to take into account context information in the recommendation process?
  Answer: *In our work, we take it into account when modeling jobs and job seekers, as well as when generating negative samples.*

In addition to the above observations, we can also draw the following conclusions from the experimental results: (i) The job text encoder has an impact on recommendation results, i.e., a suitable encoder can make recommendations better. Therefore, it is an interesting task to design suitable encoders, such as how to encode job requirements, as described in Section 6.5.7. (ii) Using different parts of a job posting (e.g., job title, job description, and job requirements) as the job representation yields different model performances. In addition to considering all information, the method of only considering job titles has also achieved good results. Such a result supports the point we made in Chapter 4 that job titles are informative.

Perspectives In this chapter, all experiments are performed on the *CareerBuilder12* dataset containing job application sequences. However, the format of the dataset provided by Randstad is not exactly the same. This is why we do not apply our proposed method to the Randstad data. Therefore, one of the future research directions is to adopt the proposed method on the Randstad data (B.1.2) to recommend the next job based on resumes. Another research direction is to design a new job requirement encoder that will take into account the complex relationship between required qualifications and jobs, as described in section 6.5.7. In addition, exploring more advanced job content encoding methods for better job content representation is also a research direction, for example, learning representations from well-constructed graphs via graph embedding methods, as done in Chapter 4 (i.e., learning job title representations from graphs).

Chapter 7

Conclusion and Perspectives

Conclusion

In this thesis, we addressed the recruitment-related representation learning problem, more precisely, exploiting graph-structured data through AI methods.

With the development of technology, the field of recruitment has entered a new era, the era of Artificial Intelligence (AI) helping and guiding. Like other fields, the recruitment field is also looking to use AI to improve efficiency, reduce costs, and improve employer and employee satisfaction. As a pioneer in the recruitment industry, Randstad actively develops AI-assisted recruitment and strongly supports our research, hoping to improve recruitment efficiency by developing new and advanced AI tools. As a consequence of the vigorous development of E-recruitment and the increasingly fierce talent competition, the amount of recruitment data is increasing, and the data forms are becoming more and more complex and diverse, as we summarized in Section 1.2. Therefore, the current industry urgently needs suitable methods to represent, manage and analyze these data.

Massive data provides a guarantee for the research of AI-based methods, allowing researchers to learn more quantitatively, but at the same time, due to the particularity of recruitment, these data also bring challenges to AI methods, as we described in Section 1.3. Beyond these challenges, by summarizing and comparing existing works related to recruitment in Chapter 2, we arrive at a research gap that few studies have investigated: graph-structured data are rarely used explicitly (i.e., tasks are directed towards it) or implicitly (i.e., as auxiliary information for other tasks) in the field of recruitment,especially for representation learning. **In such context, this thesis aims to study AI techniques, especially Deep Learning (DL)-based methods to assist recruitment-related representation learning through a graph perspective.**.

Revisiting Our Contributions After summarizing and analyzing existing works related to recruitment in Chapter 2 and introducing the necessary background knowledge on graph embedding and recommendation methods in Chapter 3, our main contributions are:

- In Chapter 4, we proposed a strategy to enrich graphs to improve job title representation learning. More specifically, we constructed *Job-Transition-Tag Graph* from the career trajectory of talents, a heterogeneous graph containing two types of nodes, i.e., job titles and tags (i.e.,

words related to job responsibilities or functions), and two types of edges (i.e., job transition and "has/in" relationships). Then, we reformulated job title representation learning as a task of learning node representation from the Job-Transition-Tag Graph. The learned representation contains both graph topological and job title semantic information, and its effectiveness has been demonstrated through job classification and next job prediction tasks.

- In Chapter 5, we proposed to learn skill representations by simultaneously considering node pairwise proximity and hierarchical community structure, resulting in better skill representations that facilitate occupation/job/talent classification. More specifically, we first constructed a skill co-occurrence graph from occupations equipped with the required skills. Then we learned skill representations by combining the node proximity information in it with the hierarchical community structure information in a predefined *skill taxonomy*. The effectiveness of learned skill representations has been demonstrated using an occupation classification task.

- In Chapter 6, we studied the next-application prediction problem, an important task of the job recommender system, which aims to recommend the next job for a job seeker to apply for based on his/her historical application record. More specifically, we modeled the task as a sequential recommendation problem. We first learned job representations in an unsupervised way. Then, to adapt the importance of each job to a job seeker, we adopted a personalized-attention mechanism. Moreover, since job seekers need to consider location factors when making a choice, we took into account the geographic location to make recommendations. Extensive experiments have shown the effectiveness of our proposed method.

Listing Our Publications Based on the above three contributions, we have the following publications, including conference papers and book chapters:

- Improving Next-Application Prediction with Deep Personalized-Attention Neural Network, published in *IEEE International Conference on Machine Learning and Applications* (*ICMLA*), 2021.

- Next Job Application Prediction by Leveraging Textual Information, Metadata, and Personalized-Attention Mechanism, published in *Deep Learning Applications, Volume 4, Springer*, 2022.

- Towards Job-Transition-Tag Graph for a Better Job Title Representation Learning, published in the *Findings of Annual Conference of the North American Chapter of the Association for Computational Linguistics* (*NAACL*), 2022.

Facing Our Limitations At the end of each previous chapter, we summarized the corresponding contributions, answered the related research questions, and suggested specific improvements to the proposed method. In this ending chapter, to help refine our proposed methods and ideas, as well as to lay the groundwork for our possible future work, we point out the overall limitations of contributions as follows:

- **Data processing lacks validity analysis**: One of the challenges posed by recruitment data (Section 1.3) is that there is little access to annotated datasets. A detailed summary is given in Table 2.7. Although in some tasks, we utilized unsupervised learning methods to address this

problem, we manually generated some datasets, which introduced some noise. For example, in Chapter 4, in order to evaluate the learned job title representations, we used an online third-party API *O*Net-SOC AutoCoder*[1] (B.3.1) to assign a SOC code to each job title and directly used the French translation version of the IPOD dataset on Randstad dataset. These operations are not rigorous enough, which will bring some data noise, i.e., the classification results cannot be guaranteed to be correct, and the direct translation from English to French ignores the characteristics of different languages.

- **Learning schemes lack versatility and uniformity**: Different contribution targets an independent task, i.e., learning job title representation, learning skill representation, and predicting the next-application. Seemingly independent, these tasks actually "overlap", i.e., the learning scheme can be leveraged from each other. For example, when modeling job representations for next-application prediction task (Chapter 6), job title, even job representations can be learned via the scheme used in Chapter 4.

- **Evaluation datasets lack diversity**: The evaluation of our contribution is not comprehensive and thorough, one of the reasons is that the number of datasets used is not enough, the coverage area is relatively small, or the data accuracy is not high.

- **Lack of theoretical new models**: The general idea in Chapter 4 and Chapter 5 is to "add" new additional information when learning from graphs via graph embedding methods, i.e., tag-nodes in job title representation learning, and hierarchical information in skill representation learning. However, the graph embedding methods used are some existing models which can not fit our recruitment data well, such as ignoring node semantic information crucial to the recruitment domain.

- **Lack of fairness and explainability**: As we noted in Challenge 3, the recruitment process is a series of decision-making tasks, and an explanation for each decision-making action is helpful and extremely meaningful. Therefore, explainable recommendations are also one of the scientific objectives of Randstad. Although the attention mechanism can be viewed as a type of explanation, i.e., using attention scores to reveal the importance of elements, our proposed methods lack explainability overall. Furthermore, no works considered fairness and analyzed fairness in decision-making actions, although it is an important practical problem in the real world.

- **Lack of online testing**: All the methods and ideas proposed are still in the offline testing stage, i.e., using some previous data to learn and evaluate. This also points out another limitation: our methods were designed without considering practical issues such as computational efficiency and parallelism.

Picturing Our Perspectives Based on the overall limitations of this thesis and the specific issues of each model, we propose the following possible future research directions:

[1] http://www.onetsocautocoder.com/plus/onetmatch

- **A novel hierarchical community structure preserving graph embedding method for learning recruitment-related representations**: As stated in Chapter 5, existing graph embedding methods that preserve hierarchical community structures suffer from two major limitations: (i) ignore node features and (ii) only aggregate hierarchical information from top to bottom. These limitations inspire us to propose a new model in the future that will consider both top-bottom and bottom-top information transfer, and this model incorporates node features into the learned representation.
- **A flexible representation learning framework for various recruitment tasks**: Instead of dealing with different tasks separately and using an independent representation learning scheme for each task, it may be more helpful and efficient to introduce a novel learning framework that can freely combine different learning methods to achieve the best results for the target task.
- **A fair recommender system for explainable recruitment**: In Chapter 4 and Chapter 5, the learning of job title and skill representations is mainly through existing graph embedding methods. A line of future work is to propose a novel graph embedding model that can better adapt to the recruitment domain,í
- **A detailed and in-depth survey of recruitment-related works**: The field of recruitment is favored by academia and industry, and various related works are emerging in endlessly, as described in Chapter 2. However, to the best of our knowledge, there are very few studies for sorting out, summarizing, and analyzing existing related works. In the future, we plan to more systematically analyze and sort out the recent works related to the recruitment field, especially those based on DL.

To conclude, although the proposed methods and ideas still have many limitations, and although the proposed contributions are small, we hope to propose more meaningful and constructive works for the recruitment industry in the future.

Appendix A

Supplementary Results

A.1 Job Title Representation Learning from Graphs

A.2 Skill Representation Learning by Leveraging Hierarchical Graph

Method	Param	ESCO_K	ESCO	ROME
DeepWalk	L_walk	100	100	50
	N_walk	100	100	100
Node2Vec	p	1	0.25	1
	q	0.25	0.25	0.25
GCN	N_layer	1	1	1
	dropout	0.2	0.2	0.2
	weight_decay	5e-5	5e-5	5e-5
	lr	0.001	0.001	0.001
GAT	N_layer	1	2	2
	dropout	0.2	0.2	0.2
	dropout(att)	0.2	0.2	0.2
	N_head	8	8	8
	weight_decay	5e-5	5e-5	5e-5
	lr	0.0001	0.001	0.001
DGI	N_layer	1	1	1
	dropout	0.2	0.2	0.2
	weight_decay	5e-4	5e-4	5e-4
	lr	0.0001	0.001	0.0001
M-NMF	N_cluster	25	99	357
CommDGI	N_cluster	75	359	357
GNE	batch_size	64	64	64
	lr(emb)	0.003	0.003	0.01
SpaceNE	max_epoch	200	100	100
	lr	0.001	0.1	0.1

TABLE A.1: Best-performing hyperparameter settings for each method and dataset based on occupation classification.

A.3 Next-Application Prediction from Job Application Sequences

Appendix B

Dataset and Tool Description

B.1 Dataset

B.1.1 CareerBuilder12

Job Title Label Assignment

Job titles are not pre-labeled in the original working experience dataset provided by CareerBuilder12. Therefore, for the job title classification task, we use an online third-party API *O*Net-SOC AutoCoder*[1] (B.3.1) to assign a Standard Occupation Classification (SOC) 2018 code (B.2.2) to each job title, as well as a match score (i.e., the scores above 70 means that the correct code is accurately predicted at least 70% of the time). SOC 2018 is a four-level taxonomy structure consisting of *MajorGroup* (23), *MinorGroup* (98), *BroadGroup* (459) and *DetailedOccupation* (867). For example, *O*Net-SOC AutoCoder* assigns the code 11-2022 (Sales Managers) for the title "sales director", which belongs to the level of *DetailedOccupation*. The *BroadGroup* level is 11-2020 (Marketing and Sales Managers), the *MinorGroup* level is 11-2000 (Advertising, Marketing, Promotions, Public Relations, and Sales Managers), and the *MajorGroup* level is 11-0000 (Management Occupations). In Chapter 4, we categorize job titles into *MajorGroup*. We have annotated a total of 30,000 job titles. The developer guarantees that the code assigned to the title plus description has an accuracy rate of 85%. However, only the job title is provided in our experiments, so the SOC 2018 code may be incorrectly assigned. For this reason, we filtered out job titles with scores below 70. Therefore, 22,590 job titles remain.

B.1.2 Randstad

Parsed Resume

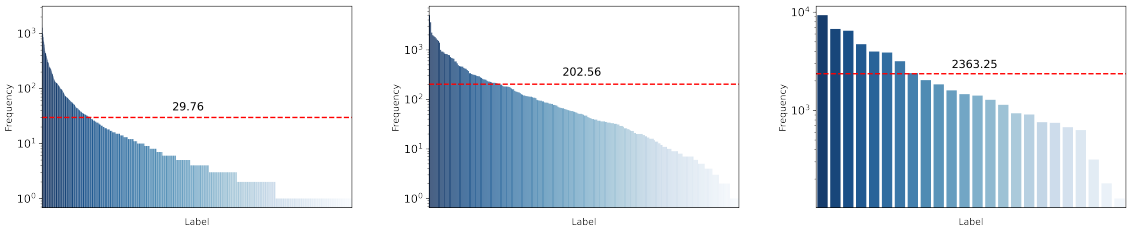
The parsed resume data is provided by the talent pool of Randstad company,[2] where each resume is in French and is parsed into sections, e.g., *Personal EducationHistory* and *EmploymentHistory*, etc. In totally, we have 15,964 raw parsed resumes, 14,698 resumes have *EmploymentHistory*, and 13,161 resumes have valid *EmploymentItem*s.[3] The distribution of job title with respect to different categories is illustrated in Figure B.1 and Table B.1. It can be observed that the number of different

[1] http://www.onetsocautocoder.com/plus/onetmatch

[2] https://www.randstad.com/

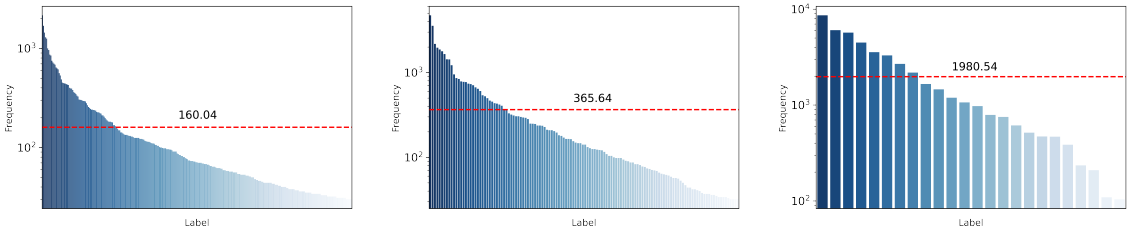
[3] Having all information about *StartDate, JobTitle, JobCode, JobGroup, JobClass*

classes is very unbalanced. Therefore, we removed 'rare" labels. In these experiments, we removed the *EmploymentItem*s labeled 'rare" *JobCode* (i.e., the frequency is lower than the average value 29.76). Furthermore, we discarded resumes with less than two valid *EmploymentItem* for link prediction purpose, and sorted *EmploymentItem*s by *StartDate* in ascending order. Finally, we get a filtered dataset containing 10,183 resumes. Similarly, the distribution of categories and data statistics are shown in Figure B.2 and Table B.1, respectively.



(a) *JobCode* (b) *JobGroup*. (c) *JobClass*

FIGURE B.1: Label distributions (raw), where the red line represents the average value.



(a) *JobCode* (b) *JobGroup*. (c) *JobClass*

FIGURE B.2: Label distributions (filtered), where the red line represents the average value.

TABLE B.1: Dataset statistics of raw/filtered parsed resumes, where *occ* means occurrence and *item* means *EmploymentItem*.s

	Raw			Filtered		
	JobCode	***JobGroup***	***JobClass***	***JobCode***	***JobGroup***	***JobClass***
# Unique	1,906	280	24	297	130	24
# Max occ	2,156	4,997	9,315	2,156	4,719	8,614
# Min occ	1	1	127	30	32	104
# Mean occ	29.76	202.56	2363.25	160.04	365.64	1980.54
# Max item		30			30	
# Mean item		14.71			13.13	

B.1.3 IPOD

This dataset consists of 192k job titles belonging to 56k LinkedIn users, and each of these job titles is manually associated with its associated level of seniority, the domain of work and location.

The French version will be added.

B.2 Terminology resource

TABLE B.2: Summary of all terminology resources.

Name	Organization	Hierarchy	Language
ISCO 2008	International Labour Organization	4 (436/130/43/10)	EN
SOC 2018	U.S. Bureau of Labor Statistics	4 (867/459/98/23)	EN
O*NET 2019	U.S. Dept. of Labor/Employment and Training Administration	5(149/867/459/98/23)	EN
ESCO 2017	European Commission	5(2,942/436/130/43/10)	27 languages
ROME	France Pôle emploi	3(532/110/14)	FR

B.2.1 ISCO 2008

International Standard Classification of Occupations (ISCO) 2008[4] is the current version of International Standard Classification of Occupations, released in 2008 by International Labour Organization. It has a four-level hierarchical occupation taxonomy that covers all jobs globally. It classifies jobs into 436 Unit Groups, then aggregates these Unit Groups into 130 Minor Groups, 43 Sub-Major Groups and 10 Major Groups, based on their similarity in skill level and skill specialization required for the job.

B.2.2 SOC 2018

Standard Occupation Classification (SOC) 2018[5] is developed by the U.S. Bureau of Labor Statistics to meet the growing need for a universal occupation classification system. Such a classification system allows government agencies and private companies to generate comparable data. SOC 2018 divides workers into 867 detailed occupations (i.e., Detailed SOC Occupations) according to their occupational definitions, and these detailed occupations are combined to form 459 Broad Occupations, 98 Minor Groups, and 23 Major Groups. An example is illustrated in Figure B.3.

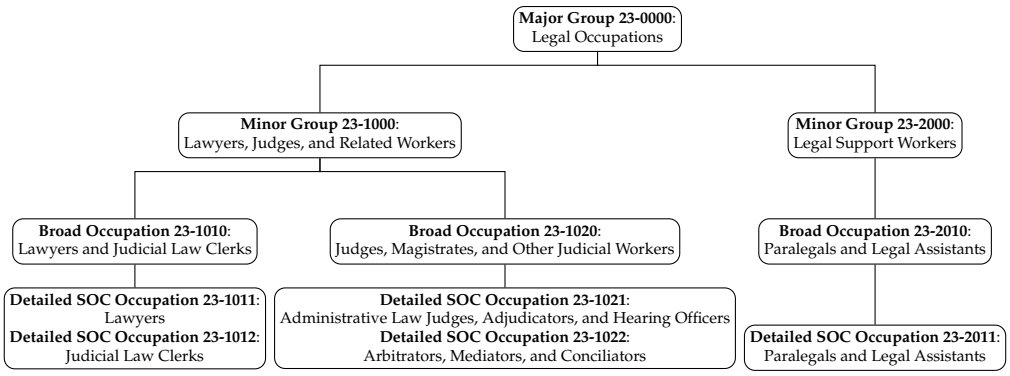


FIGURE B.3: An example of SOC structure.

[4] https://www.ilo.org/public/english/bureau/stat/isco/isco08/

[5] https://www.bls.gov/soc/

B.2.3 O*NET 2019

Occupational Information Network (O*NET) 2019[6], a free online database containing definitions of almost 1,000 occupations covering the entire U.S. economy, was developed under the sponsorship of the U.S. Department of Labor/Employment and Training Administration through a grant to the North Carolina Department of Commerce. The taxonomy structure of O*NET-SOC 2019 has been revised based on the transition to SOC 2018. The latest version includes 1,016 occupational names (867 of which are SOC and 149 are Detailed O*NET-SOCs), of which 923 represent O*NET data-level occupations and 93 represent non data-level.

B.2.4 ESCO 2017

European Skills, Competences, Qualifications and Occupations (ESCO) 2017[7] is the European multilingual classification of Skills, Competences, and Occupations. ESCO 2017 is like a dictionary that describes, identifies, and classifies professional occupations and skills relevant to the EU labor market, education and training. ESCO 2017 provides descriptions of 2,942 occupations and 13,485 skills associated with these occupations, translated into 27 languages.

- **Occupations**: in ESCO 2017, the occupation concepts use their hierarchical relationships, metadata, and mapping to the ISCO 2008 to build the occupation structure. In ESCO 2017, each occupation maps exactly to a ISCO 2008 code (i.e., the current version, released in 2008) B.2.1. Therefore, ISCO 2008 can be used as a hierarchical structure for the occupation taxonomy. ISCO 2008 provides the top four levels for the occupation taxonomy. ESCO 2017 occupations are at level five and lower. An example of ocupation taxonomy is shown in Figure B.4.

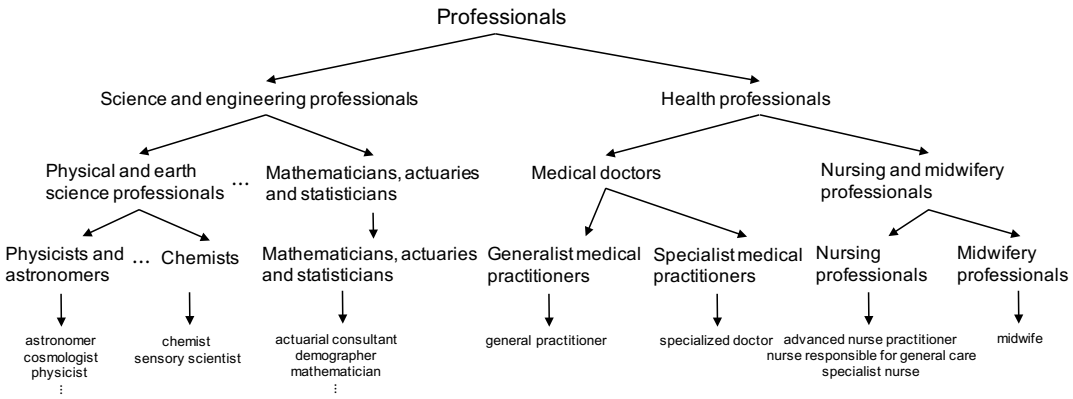


FIGURE B.4: A sample of *occupation taxonomy* in ESCO 2017.

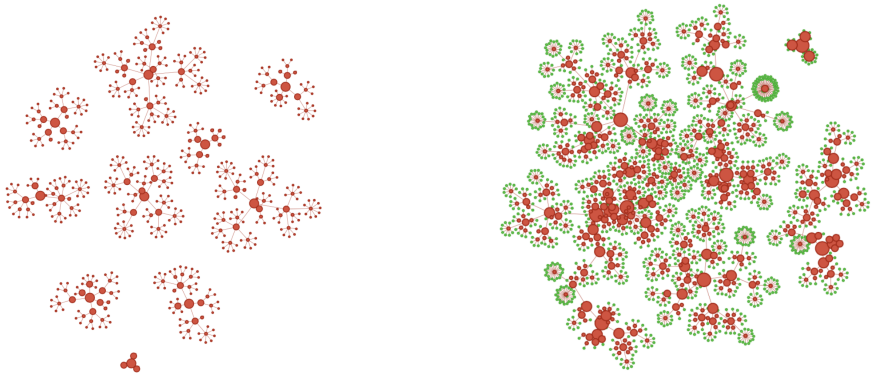
  - **occupation**: 2,942
  - **occupation group**: 619

[6] https://www.onetonline.org/

[7] https://ec.europa.eu/esco/portal

  - 436 Unit Groups
  - 130 Minor Groups
  - 43 Sub-Major Groups
  - 10 Major Groups

We show the occupation group relationship in Figure B.5a, where the larger the circle is, the higher its level. We can see that occupation groups are grouped into 10 clusters, which is 10 Major Groups.



(a) Occupation_group. (b) Occupation_group and occupation

FIGURE B.5: The illustration of occupation groups and occupations in ESCO 2017, where the red point is the occupation group node, and green point is occupation.

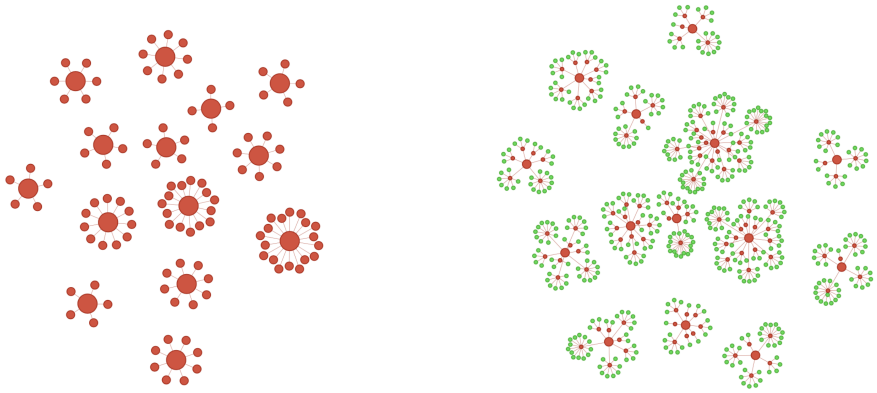
- **Skills/Competences**: the ESCO 2017 skill taxonomy distinguishes between (i) the concept of skill/competence (i.e., there is no distinction between skills and competencies) and (ii) the concept of knowledge by indicating the type of skill. Each of these concepts comes with a preferred term and some non-preferred terms. Each concept also includes an explanation in descriptive form. The skills taxonomy of ESCO 2017 contains 13,485 concepts, which are structured in a hierarchy with four sub-categories: (i) Attitudes and values, (ii) Knowledge, (iii) Language skills and knowledge, and (iv) Skills. Each sub-category targets different types of knowledge and skill/competency concepts.
  - **skill**: 13,485
    - 6,412 sector-specific
    - 3,643 cross-sector
    - 2,977 occupation-specific
    - 453 transversal
  - **skill group**: 656

ESCO_K

B.2.5 ROME

Répertoire Opérationnel des Métiers et des Emplois (ROME)[8] is build by the Pôle emploi[9], which is a tool at the service of professional mobility and bringing together offers and candidates. It has an inventory of the names of trades, jobs, and knowledge, as well as groups them according to the principle of equivalence or proximity.

- **Occupations**
  - **occupation**: 532
  - **occupation group**: 114
    - 110 sub-major occupation groups
    - 14 major occupation groups



(a) Occupation_group. (b) Occupation group and occupation

FIGURE B.6: The illustration of occupation group and occupation in ROME, where the red point is the occupation group node, and green point is occupation.

- **skill**: 13.913
  - 8,983 SavoirFaire (know-how)
  - 4,930 Savoir (knowledge)

[8] https://data.europa.eu/data/datasets/58da857388ee384902e505f5?locale=en
[9] https://www.pole-emploi.fr/accueil/

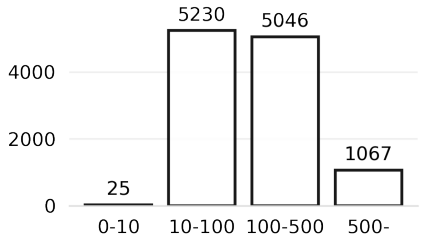
B.3 Tool

B.3.1 O*NET-SOC AutoCoder

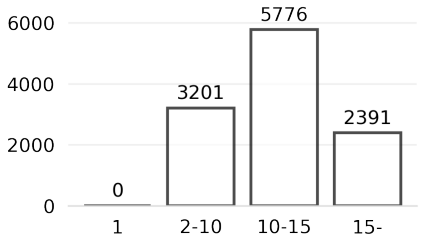
O*NET-SOC AutoCoder[10] is a tool that can instantly provide a high-quality occupational classification code (i.e., SOC 2018, O*NET 2019, and OES 2020) for a job, resume and, UI claim. The developers guarantee that the code assigned to job postings (title plus description) is 85% accurate. The primary mechanism of O*NET-SOC AutoCoder is simple: it allows two main inputs: job title and job description. It splits the text of a job posting, resume, or UI statement into individual words and phrases, which are then matched against a database of words and phrases associated with O*Net codes. The words in the database have been reviewed and weighted by analysts, so the most important words to a given occupation will have more weight in the match calculation. O*NET-SOC AutoCoder has several different methods to match occupation codes to inputs. The final result is a weighted average of these methods.

B.4 Experimental Datasets

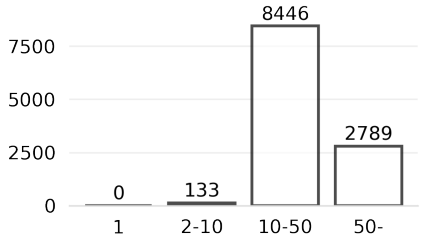
B.4.1 Chapter 4: Skill Representation Learning



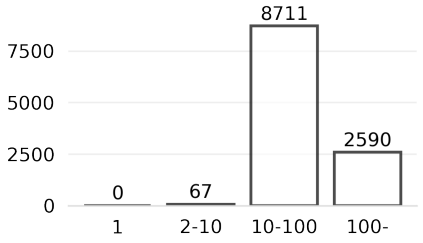
(A) Neighbor number.



(B) 4-level skill categories (17).



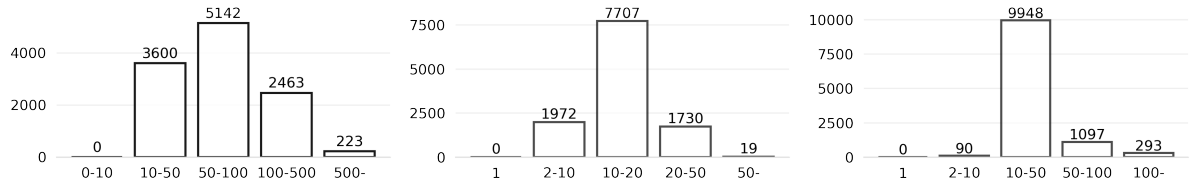
(C) 3-level skill categories (99).



(D) 2-level skill categories (359).

FIGURE B.7: Neighborhood statistics for *ESCO*. In (a), the x-axis represents the size of the neighborhood $|\mathcal{N}(s)|$, and in (b), (c) and (d), the x-axis represents the number of neighbor types (i.e., "skill categories"). The y-axis of all subfigures represents the corresponding number of nodes.

[10] https://www.onetsocautocoder.com/plus/onetmatch?action=guide



(A) Number of neighbors. (B) 3-level skill categories (53). (C) 2-level skill categories (357).

FIGURE B.8: Neighborhood statistics for *ROME*. In (a), the x-axis represents the size of the neighborhood $|\mathcal{N}(s)|$, and in (b), (c) and (d), the x-axis represents the number of neighbor types (i.e., "skill categories"). The y-axis of all subfigures represents the corresponding number of nodes.

Appendix C

Feature Extraction Methods

In this appendix, we briefly introduce various feature extraction models [Naseem et al., 2021] commonly used in different Natural Language Processing (NLP)-related tasks, including some classical models and some representation learning models.

C.1 Classical Methods

We first introduce some early classic methods commonly used to represent texts, including (i) categorical word representation and (ii) weighted word representation [Naseem et al., 2021]:

C.1.1 Categorical Word Representation

Categorical data is data that takes only a limited number of values. For example, a sweater may have several "color values" such as red, blue, and green. Most Machine Learning (ML) algorithms cannot work with categorical data and need to convert it to numerical data. Therefore, categorical word representation thus allows turning categorical data into features with numerical values, which is the easiest way to represent texts.

- **One-Hot Encoding**: One-Hot Encoding is the most straightforward method to transfer categorical data into numerical values. In One-Hot encoding, the vector dimension is the number of terms present in the corpus (vocabulary), i.e., N. Each unique term has a special index, represented by a 1 at that index and 0s everywhere else. An example of the One-Hot encoding of the location of the job seeker used in Section 6.4.2 is given in Figure C.1.

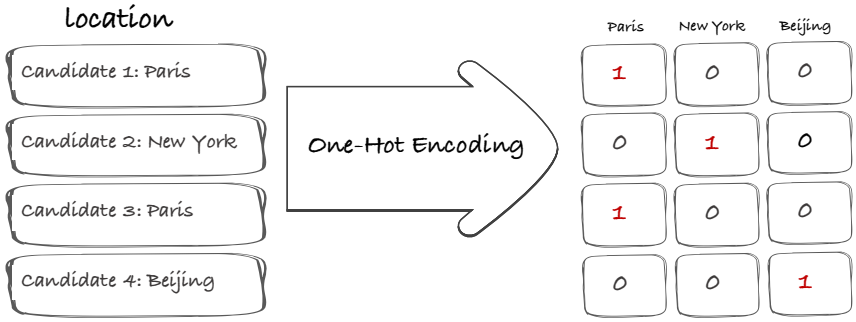


FIGURE C.1: Example of One-Hot encoding of the location of the job seeker.

- **Bag-of-Words (BoW) Encoding**: BoW encoding is a simple extension of One-Hot encoding that adds up the One-Hot representations of words in the text. As shown in Figure C.2, there are two sentences, and the corresponding vocabulary set is $\{a, day, is, it, meet, nice, to, you\}$.

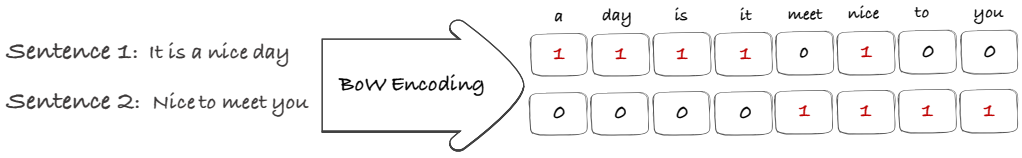


FIGURE C.2: Example of BoW encoding of two sentences.

The most notable advantage of BoW is its simplicity and ease of use. However, as the vocabulary grows, it suffers from sparse representation and dimensionality curve issues. Besides, BoW ignores the order of words and the semantic relationship between words.

C.1.2 Weighted Word Representation

Unlike count-only categorical word representation methods, weighted word representation methods further take word frequency into account.

- **Term Frequency (TF)**: TF calculates how often a word occurs in a document. It is usually formulated as follows:

$$\texttt{TF}(t,d) = \frac{\texttt{occur}(t,d)}{n^d},$$

where $\texttt{occur}(t,d)$ represents the number of times the term t appears in the document d, and n^d is the number of terms in the document d.

- **Term Frequency Inverse Document Frequency (TF-IDF)**: To reduce the impact of common words in the corpus, such as "a", "and" and "the", TF-IDF [Salton and Buckley, 1988] combines Inverse Document Frequency (IDF) with TF, which is mathematically represented as follows:

$$\texttt{TF-IDF}(t,d,\mathcal{D}) = \texttt{TF}(t,d) \times \log\left(\frac{|\mathcal{D}|}{\texttt{df}_t}\right),$$

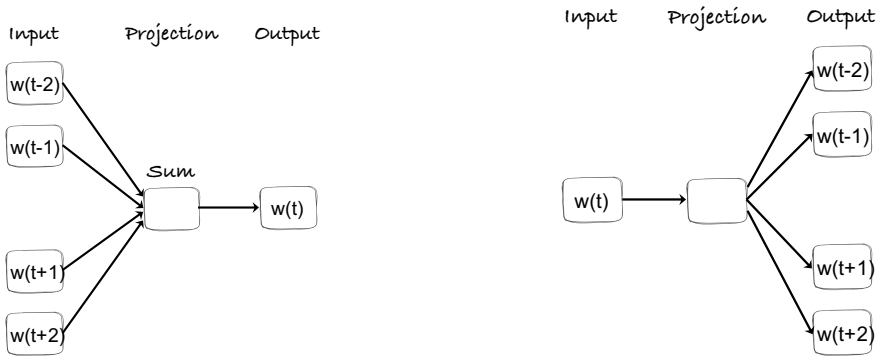
where $\mathcal{D}$ represents the collection of documents, and $\mathtt{df}_t$ represents the number of documents that contain the term t.

C.2 Representation Learning Methods

Classical methods often fail to capture the syntactic and semantic meaning of words, and these models suffer from the curse of high dimensionality. The shortcomings of these methods motivate research on distributed word representation in low-dimensional spaces. Next, we briefly introduce several commonly used distributed representations, which are categorized into (i) non-contextual word representation (continuous word representation) methods and (ii) contextual word representation methods.

C.2.1 Non-Contextual Word Representation

- **Word2Vec**: Word2Vec [Mikolov, K. Chen, et al., 2013] is the most representative work on continuous word representation. It comes with two architectures for learning high-quality word vectors: (1) Continuous Bag Of Words (CBOW) and (2) Skip-Gram (SG). Both architectures are feed-forward Neural Networks (NNs), consisting of three layers: an input layer, a hidden layer, and an output layer, but the objective functions they try to approximate are different. CBOW takes the context of a given word as input and tries to predict the word. SG is similar to CBOW, but instead of predicting the current word based on the context, it tries to predict the context words associated with the current word. Figure C.3 shows the principle of the two structures.



(a) Continuous Bag Of Words. (b) Skip-Gram.

FIGURE C.3: The CBOW and SG two structure of Word2Vec, adapted from [Mikolov, K. Chen, et al., 2013].

- **Global Vectors (GloVe)**: Word2Vec has difficulty leveraging statistics across all corpus because it is trained on local context windows and does not exploit the statistics contained in the global co-occurrence matrix. In contrast, GloVe [Pennington, Socher, and Manning, 2014]

is trained on global word-word co-occurrence counts, making efficient use of global statistics. For example, given three sentences:

- `I like Machine Learning`
- `I like summer`
- `Machine Learning is powerful`

If the window size is equal to 2, the corresponding co-occurrence matrix is given as follows:

	I	like	Machine	Learning	summer	is	powerful
like	2	0	1	0	1	0	0
Machine	0	1	0	2	0	0	0
Learning	0	0	2	0	0	1	0
summer	0	1	0	0	0	0	0
is	0	0	0	0	0	0	1
powerful	0	0	0	0	0	1	0

- **FastText**: FastText [Bojanowski et al., 2017] uses a similar CBOW structure with hierarchical softmax as Word2Vec, but FastText uses a bag of character-level n-gram.

  The character-level n-gram is a set of co-occurring characters within a given window, similar to the word-level n-grams, except that the window size is at the character level. For example, for the word "hello", when $n = 2$, the 2-grams are: {"<h", "he", "el", "ll", "lo", "o>"}, so "hello" is represented by the sum of its character-level n-grams. The character-level n-gram has two advantages: (i) It is better for low-frequency words (ii) For out-of-vocabulary words, their word vectors can still be constructed, i.e., stack their character-level n-gram vectors.

The models described above have demonstrated their effectiveness in various downstream tasks. However, one of the most severe weaknesses is that they can not handle words with multiple senses. For example, the word "bank" has two commonly used linguistic meanings, i.e., the meaning of financial institution and the meaning of riverside. Different word senses should have different context words, but Word2Vec, GloVe, and FastText are trained to predict the same word regardless of different contexts. Therefore, the word "bank" has a static embedding. To address this problem, some contextual models are proposed, which will be explained in the next section.

C.2.2 Contextual Word Representation

- **Embeddings from Language Models (ELMo)**: Different from traditional word embeddings, which assign a vector to each token based on the entire input sentence. ELMo derives word embeddings from the top of two-layer bidirectional Language Models, where each layer is a

bidirectional LSTM. The bidirectional Language Model combines both a forward and backward Language Model, then jointly maximizes the log-likelihood of the forward and backward directions, as shown in Figure C.4.

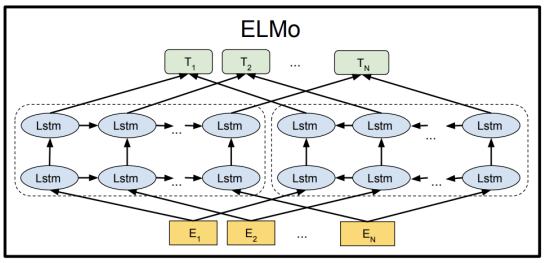


FIGURE C.4: ELMo architecture, from [Devlin et al., 2018].

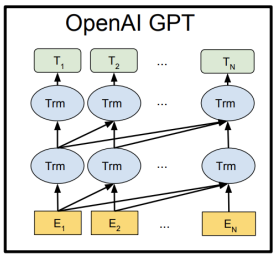


FIGURE C.5: OpenAI GPT architecture, from [Devlin et al., 2018].

- **OpenAI Generative Pre-Training (GPT)**: Similar to ELMo and BERT, GPT also uses a two-step framework: (i) pre-training and (ii) fine-tuning. The pre-training of GPT as shown in Figure C.5 is similar to ELMo, except that GPT uses Transformer [Vaswani et al., 2017] as the feature extractor instead of LSTM. Furthermore, GPT uses a left-to-right architecture, where each token can only focus on previous tokens in the self-attention layers of the Transformer.

- **Bidirectional Encoder Representation from Transformers (BERT)** [Devlin et al., 2018]: It is a pre-trained model proposed by Google in 2018, where the encoder is bidirectional transformers. The framework of BERT has two steps: (i) pre-training and (ii) fine-tuning. During pre-training, the model is pre-trained using two unsupervised objectives to capture word- and sentence-level representations, respectively.

  – **Masked Language Model**: simply mask 15% of the input tokens at random, and then predict those masked tokens. For example:

  $$\texttt{I like Machine Learning} \rightarrow \texttt{I like Machine [MASK]}$$

  However, since the [MASK] token does not appear during fine-tuning, this leads to a mismatch between pre-training and fine-tuning. To address this problem, the authors do not always replace "masked" words with the actual `[MASK]` token. Instead, for these 15% token positions, only 80% tokens are actually replaced with `[MASK]`, 10% tokens are replaced by random tokens, and the 10% remains unchanged.

  – **Next Sentence Prediction**: each pre-training example consists of two sentences: the sentence `A` and `B`, 50% of the time `B` is the actual next sentence after `A`, and 50% of the time it is a random sentence from the corpus.

  For fine-tuning, BERT is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks.

Appendix D

Preliminary of Graph Embedding

D.1 Skip-Gram

In 2013, [Mikolov, K. Chen, et al., 2013] presented two architectures for learning high-quality word vectors from huge data sets, under the common name of Word2Vec: Continuous Bag Of Words (CBOW) and Skip-Gram (SG). Both architectures are feed-forward Neural Networks (NNs) [Bengio, Ducharme, et al., 2003], consisting of three layers: an input layer, a hidden layer, and an output layer, but the objective functions they try to approximate are different. CBOW takes the context of a given word as input and tries to predict the word. SG is similar to CBOW, but instead of predicting the current word based on the context, it tries to predict the context words associated with the current word. In the case of SG architecture, two main optimizations have been presented: *negative sampling*, which modifies only some weights related to negative examples (i.e., words not in the context), and *hierarchical softmax*, where the output vector is determined by a tree-like traversal of the network layers.

D.1.1 Language Modeling with Skip-Gram

Generally speaking, language modeling aims to estimate the likelihood of a specific word sequence appearing in a corpus with the vocabulary set $\mathcal{V}$. More formally, given a sequence of words $\{w_1, \ldots, w_n\}$, where $w_i \in \mathcal{V}$, we want to maximize the co-occurrence probability among the words that appear within a context of the pre-fixed window size c in a sentence of the corpus:

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{-c\leq j\leq c, j\neq 0}\log Pr\left(w_{i+j}|w_i\right) \tag{D.1}$$

The basic SG uses the softmax function to define $Pr(w_{i+j}|w_i)$ as:

$$Pr(w_{i+j}|w_i) = \frac{\exp\left(\Phi'(w_{i+j})^T\Phi(w_i)\right)}{\sum_{w'\in\mathcal{V}}\exp\left(\Phi'(w')^T\Phi(w_i)\right)},$$

where $\Phi := \mathcal{V} \to \mathbb{R}^{|\mathcal{V}|\times d}$ and $\Phi' := \mathcal{V} \to \mathbb{R}^{|\mathcal{V}|\times d}$ are the "source" (i.e., it is treated as the source word) and "target" (i.e., it is treated as the "context" of other words) d-dimensional embedding mappings, respectively.

The model is optimized by Gradient Descent. To speed up the training time, [Mikolov, Sutskever, et al., 2013; Mikolov, K. Chen, et al., 2013] propose two approximations: *negative sampling* and *hierarchical softmax*.

D.1.2 Negative Sampling

Negative Sampling allows each training example to modify only a small part of parameters, rather than all of them. For each observed pair (w_{i+j}, w_i), [Q. Le and Mikolov, 2014] samples n^- negative context words $w' \in \mathcal{V}$ from a noise distribution $P_n(w')$. The loss function is defined as follows:

$$\log \sigma\left(\Phi'(w_{i+j})^T \Phi(w_i)\right) + \sum_{p=1}^{n^-} \mathbb{E}_{w_p \sim P_n(w')}\left[\log \sigma\left(-\Phi'(w_p)^T \Phi(w_i)\right)\right],$$

where σ denotes the softmax function. Thus the task is to use logistic regression to distinguish the target word w_{i+j} from n^- negative exmples sampled from the noise distribution $P_n(w')$.

D.1.3 Hierarchical Softmax

Another way to deal with the high cost of computing $Pr(w_{i+j}|w_i)$ is hierarchical softmax. This model factorizes the conditional probability of assigning words to $|\mathcal{V}|$ leaves of a binary tree, turning the prediction problem into maximizing the probability of a particular path in the hierarchy. More precisely, each word w can be reached by an appropriate path from the root of the tree. Let $L(w_{i+j}, k)$ be the k-th node on the path from the root to w_{i+j}, and let $|L(w_{i+j})|$ be the length of this path, so $L(w_{i+j}, 1) = \texttt{root}$ and $L(w_{i+j}, |L(w_{i+j})|) = w_{i+j}$. In addition, for any inner node $w_{i+k}, 1 < k < |L(w_{i+j})|$, let $\texttt{Ch}(w_{i+k})$ be an arbitrary fixed child of w_{i+k} and $\mathbb{I}(x)$ be the indicator, equals to 1 if x is true, then -1 otherwise. Then,

$$\begin{aligned} &Pr(w_{i+j}|w_i) = \\ &\prod_{k=1}^{|L(w_{i+j})-1|} \sigma\left(\mathbb{I}\left(L(w_{i+j}, k+1) = \texttt{Ch}(L(w_{i+j}, k))\right) \cdot \Phi'(L(w_{i+j}, k))^T \Phi(w_i)\right), \end{aligned} \tag{D.2}$$

where $\sigma(x) = 1/(1+\exp(-x))$. This implies that the cost of computing $\log Pr(w_{i+j}|w_i)$ and $\Delta Pr(w_{i+j}|w_i)$ is proportional to $|L(w_{i+j})|$, the average is not greater than $\log |\mathcal{V}|$. Furthermore, unlike the standard softmax formulation of SG which assigns two representations $\Phi(w)$ and $\Phi'(w)$ to each word w, hierarchical softmax uses a representation $\Phi(w)$ for each word and a representation $\Phi'(w)$ for each inner node of the binary tree.

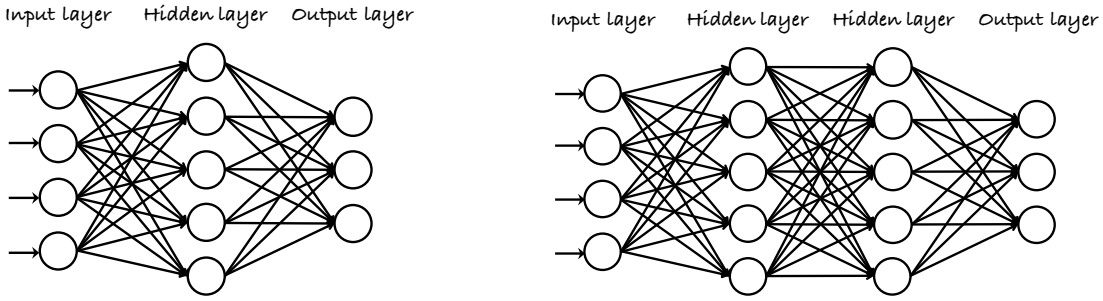
Appendix E

Neural Network Architecture

This appendix briefly introduces some major architectures of Neural Network (NN)s, more appropriately called Artificial Neural Networks (ANNs). For more details on NN and Deep Learning (DL), we refer readers to [Goodfellow, Bengio, and Courville, 2016].

E.1 Multilayer Perceptron

Multilayer Perceptron (MLP), also known as Feedforward Neural Network (FNN), is a typical NN that maps a set of input vectors to a set of output vectors. Its general architecture consists of at least three layers of nodes: an *input layer*, a *hidden layer*, and an *output layer*, as shown in Figure E.1. Except for the input nodes, each node is a neuron (or called unit) using a nonlinear activation function. Neurons in each layer are connected to neurons in the next layer, and there are no connections between neurons in the same layer. The neurons in the input layer receive input information, the hidden layer and output layer neurons process the information, and the final result is output by the output layer neurons. This type of architecture is called *feedforward* because there is no feedback connection feeding the output of the model back to itself.



(a) One hidden layer MLP.

(b) Two hidden layer MLP.

FIGURE E.1: The architecture of MLP.

For better comparison with other architectures described below, we formally describe a MLP with one hidden layer. Taking classification as an example, we have a set of input vectors $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, where each $\boldsymbol{x}_i \in \mathbb{R}^m$ has m features, and their predefined labels $\boldsymbol{y} = \{y_1, \ldots, y_n\}$,

$y_i \in \mathbb{R}$. The feedforward is formulated as:

$$\boldsymbol{H} = \sigma_h \left(\boldsymbol{X}\boldsymbol{W}_h^T + \boldsymbol{b}_h \right),$$

$$\boldsymbol{O} = \sigma_o \left(\boldsymbol{H}\boldsymbol{W}_o^T + \boldsymbol{b}_o \right).$$

$\boldsymbol{H} = \{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n\} \in \mathbb{R}^{n \times h}$ are h-dimensional hidden states, and $\boldsymbol{O} = \{\boldsymbol{o}_1, \ldots, \boldsymbol{o}_n\} \in \mathbb{R}^{n \times o}$ are o-dimensional output vectors. $\boldsymbol{W}_h \in \mathcal{R}^{h \times m}$ and $\boldsymbol{b}_h \in \mathcal{R}^h$ are weights and bias of the hidden layer. $\boldsymbol{W}_o \in \mathcal{R}^{o \times h}$ and $\boldsymbol{b}_o \in \mathcal{R}^o$ are weights and bias of the output layer. σ_h and σ_o are two activation functions used in the hidden layer and output layer, respectively. Common hidden layer activation functions are ReLU, sigmoid, and tanh [Goodfellow, Bengio, and Courville, 2016]. The output layer uses the sigmoid function for binary classification or the softmax function for multi-label classification.

E.2 Convolutiona Neural Network

Convolutional Neural Network (CNN)s are a family of NNs designed to process data with a known grid-like topology, such as time series and images.

E.3 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a family of NNs designed to process sequential data, such as time series. Different from MLP, RNN contains feedback connections, i.e., the output of the RNN at a particular step is based not only on current input but also on previous inputs. Formally, for sequential input $\{\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(T)}\}$, $\boldsymbol{x}^{(t)} \in \mathbb{R}^m$, the general RNN architecture is formulated as:

$$\boldsymbol{h}^{(t)} = \texttt{RNN}\left(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta} \right),$$

where $\boldsymbol{h}^{(t)} \in \mathbb{R}^h$ is the hidden state at step t, and $\boldsymbol{x}^{(t)}$ is the input value for that step. $\boldsymbol{h}^{(t)}$ is considered to be a summary of the sequence of past inputs up to t. This formula can be drawn in two different ways, as shown in Figure E.2.

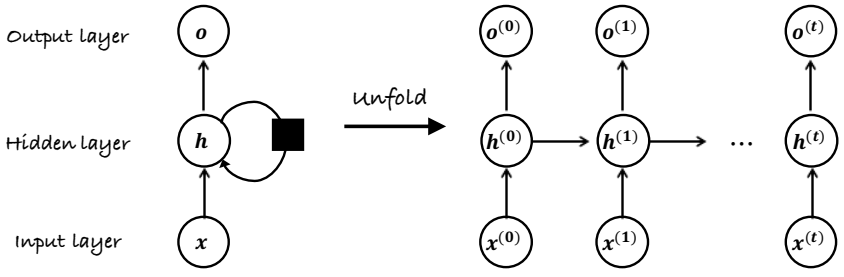


FIGURE E.2: The architecture of RNN can be expressed in two ways. On the left is the circuit diagram, where the black square represents a delay for a single time step, adapted from [Goodfellow, Bengio, and Courville, 2016], and on the right is the same network, but in an unfolded way.

The feedforward of the RNN is formulated as:

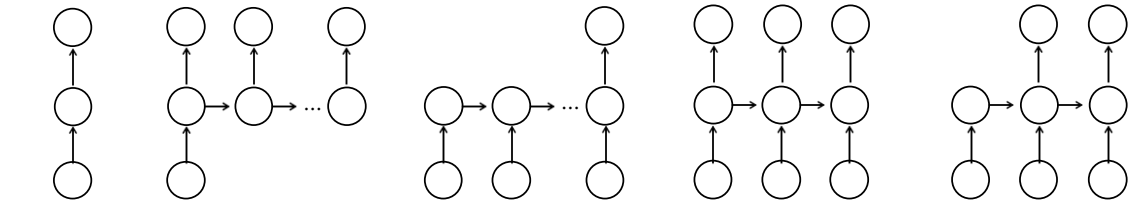
$$\boldsymbol{h}^{(t)} = \sigma_h \left(\boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{V}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h \right),$$

$$\boldsymbol{o}^{(t)} = \sigma_o \left(\boldsymbol{W}_o \boldsymbol{h}^{(t)} + \boldsymbol{b}_o \right),$$

where $\boldsymbol{h}^{(t)} \in \mathbb{R}^h$ is the hidden state and $\boldsymbol{o}^{(t)} \in \mathbb{R}^o$ is the output. $\boldsymbol{U}_h \in \mathcal{R}^{h \times m}$, $\boldsymbol{V}_h \in \mathcal{R}^{h \times h}$ and $\boldsymbol{b}_h \in \mathcal{R}^h$ are parameters of the hidden layer. $\boldsymbol{W}_o \in \mathcal{R}^{o \times h}$ and $\boldsymbol{b}_o \in \mathcal{R}^o$ are parameters of the output layer.

Depending on the number of inputs and outputs, there are different types of RNN, as shown in Figure E.3, for different applications.

- **one-to-one**: The simplest structure of RNN, which allows single input and single output. It can be used for image classification.
- **one-to-many**: This type of RNN takes a single input and returns a sequence of outputs. It can be used for music generation and image captioning.
- **many-to-one**: This type of RNN takes a sequence of input, and returns a single output. It can be used for sentiment analysis.
- **many-to-many**: This type of RNN takes a sequence of input, and returns a sequence of outputs. This type can be further divided into two subclasses,
  - **equal unit size**: the number of input and output units is the same. It can be used for Name-Entity recognition.
  - **unequal unit size**: the number of input and output units is different. It can be used for machine translation.



(a) one-to-one. (b) one-to-many. (c) many-to-one. (d) many-to-many (equal unit size). (e) many-to-many (unequal unit size).

FIGURE E.3: The different RNN types.

E.3.1 Long Short Term Memory

Long Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] is a more sophisticated variant of the RNN architecture that learns long-term dependencies better than vanilla RNNs. Just like the name of LSTM, it has both "long-term memory" and "short-term memory". The main difference between the vanilla RNN and LSTM is that LSTM has gated cells. A standard LSTM unit consists

of a cell, an *input gate*, a *forget gate*, and an *output gate*. These three gates regulate the flow of information into and out of the cell. More specifically, the *forget gate* first decides which information to discard from the previous state $\boldsymbol{c}^{(t-1)} \in \mathbb{R}^h$ by comparing it with the current input, the formula is:

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{V}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_f\right),$$

where the subscript f stands for the *forget gate*, similarly, the subscripts i, o stand for the *input gate*, *output gate* respectively. The output of the *forget gate* is a vector where each element has a value between 0 and 1, i.e., $\boldsymbol{f}^{(t)} \in (0,1)^h$. A value of 1 means to keep all information, while a value of 0 means to discard all information.

Next, the *input gate* decides what new information to store in the current state, using the same system as the *forget gate*.

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{V}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i\right),$$

where $\boldsymbol{i}^{(t)} \in (0,1)^h$.

Then, the current state $\boldsymbol{C}^{(t)}$ is updated according to the previous state $\boldsymbol{c}^{(t-1)}$,

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \odot \tilde{\boldsymbol{c}}^{(t)},$$

where the operator $\odot$ denotes the element-wise product, $\tilde{\boldsymbol{c}}^{(t)}$ is a vector of new candidate value obtained through a tanh layer, i.e., $\tilde{\boldsymbol{c}}^{(t)} = \texttt{tanh}\left(\boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{V}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_c\right)$.

Finally, the *output gate* decides what information is output in the current state by taking into account the previous and current states. The formula is:

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{V}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_o\right),$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot \texttt{tanh}\left(\boldsymbol{c}^{(t)}\right),$$

where $\boldsymbol{o}^{(t)} \in (0,1)^h$ and $\boldsymbol{h}^{(t)} \in (-1,1)^h$. The above $\boldsymbol{U}_* \in \mathbb{R}^{h \times m} / \boldsymbol{V}_* \in \mathbb{R}^{h \times h}$ and $\boldsymbol{b}_* \in \mathbb{R}^h$ are both weight matrices and biases in the corresponding gate, and $\sigma(\dot{})$ is the sigmoid activation function.

With these gates, LSTM can selectively output relevant information about the current state, allowing LSTM to maintain useful, long-term dependencies for prediction.

E.3.2 Gated Recurrent Unit

Gated Recurrent Unit (GRU) is another gated RNN. Like LSTM, it is also proposed to solve the ladder problem in long-term memory and backpropagation. The key difference between LSTM and GRU is that GRU has two gates, *reset gate* and *update gate*, while LSTM has three gates. Therefore, GRU is easier to train than LSTM.

The first step of GRU is to get the state of *update gate* $\boldsymbol{u}^{(t)}$ and the state of *reset gate* $\boldsymbol{r}^{(t)}$ according to the previous hidden state $\boldsymbol{h}^{(t-1)}$ and the current input $\boldsymbol{x}^{(t)}$:

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{V}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_u\right),$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{V}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_r\right),$$

where $\boldsymbol{U}_* \in \mathbb{R}^{h \times m} / \boldsymbol{V}_* \in \mathbb{R}^{h \times h}$ and $\boldsymbol{b}_* \in \mathbb{R}^h$ are both weight matrices and biases in the corresponding gate.

The *reset gate* is used to determine how much past information to discard, which is roughly similar to the *forget gate* of LSTM. The reset operation is formulated as:

$$\tilde{\boldsymbol{h}}^{(t-1)} = \boldsymbol{h}^{(t-1)} \odot \boldsymbol{r}^{(t)}.$$

This reset information is then fed into a tanh layer along with the input $\boldsymbol{x}^{(t)}$:

$$\tilde{\boldsymbol{h}}^{(t)} = \texttt{tanh}\left(\boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{V}_c \tilde{\boldsymbol{h}}^{(t-1)} + \boldsymbol{b}_c\right).$$

Finally, the *update gate* updates the output:

$$\boldsymbol{h}^{(t)} = (\boldsymbol{1} - \boldsymbol{u}^{(t)}) \odot \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \odot \tilde{\boldsymbol{h}}^{(t)},$$

where the first term $(\boldsymbol{1} - \boldsymbol{u}^{(t)}) \odot \boldsymbol{h}^{(t-1)}$ indicates the amount of forgotten information in $\boldsymbol{h}^{(t-1)}$, while the second term $\boldsymbol{u}^{(t)} \odot \tilde{\boldsymbol{h}}^{(t)}$ indicates the amount of forgotten information in the current information $\tilde{\boldsymbol{h}}^{(t)}$

Appendix F

Attention Mechanism

In real life, the word "attention" means to focus more on something because it is more "important" to a particular goal. The attention mechanism in Deep Learning (DL) is based on this concept of attention in real life, which aims to pay more attention to certain factors when processing data for a task. Along this line, the attention mechanism is initially proposed in [Bahdanau, Cho, and Bengio, 2014] for the neural machine translation task, the idea of which is to pay different attention to different input words to obtain better translation output. Next, this appendix briefly describes the attention mechanism and different types of attention.

F.1 Attention

The attention mechanism was introduced by [Bahdanau, Cho, and Bengio, 2014] to address the bottleneck of the traditional encoder-decoder structure in the neural machine translation task. More specifically, neural machine translation is a sequence-to-sequence task that aims to transform an input sequence into an output sequence through a series of operations. An example (adapted from[1]) is shown in Figure F.1, where the input is an English sentence, and the expected output is a French translation.

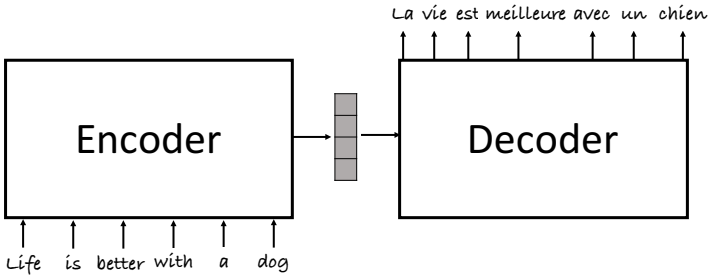


FIGURE F.1: The framework of Encoder-Decoder structure for translating English sentences into French.

The bottleneck of this framework is that the entire input sequence needs to be encoded into a fixed-length vector (i.e., the grey squares), which causes information loss, and it is difficult for the

[1] https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

decoder to obtain input information only through this fixed vector. In order to address this problem, the attention mechanism is proposed in [Bahdanau, Cho, and Bengio, 2014] so that the encoder does not need to compress the entire input sequence into a single vector but instead gives each part of the input a d-dimensional representation $\boldsymbol{s}_i \in \mathbb{R}^d$ (encoder state), and at each decoding step, giving different attention to different parts of the input. Figure F.2 gives an example where the input sentence is first encoded into six vectors, i.e., $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_6$, and then the attention mechanism computes six attention scores for each decoder hidden state $\boldsymbol{h}_j \in \mathbb{R}^d$, i.e., each score corresponds to each encoder vector $\boldsymbol{s}_i$, the darker the color of the square, the larger the attention value. Note that in [Bahdanau, Cho, and Bengio, 2014], the hidden state $\boldsymbol{h}_j$ is obtained via a RNN structure, by using the previous hidden state $\boldsymbol{h}_{j-1}$ and the current context vector $\boldsymbol{c}_j$ as input. This context vector is a weighted sum of encoder states, which is explained below.

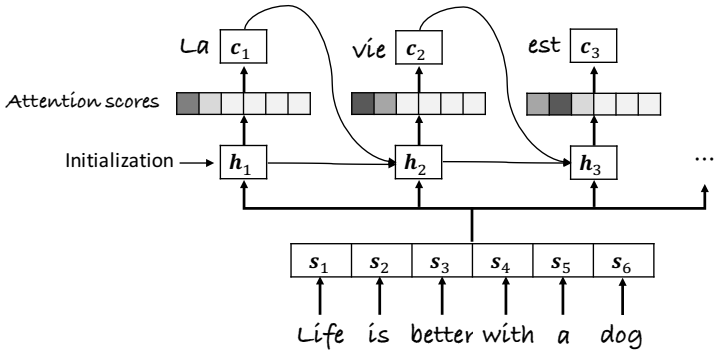


FIGURE F.2: An example of sentence translation, where at each decoding step, different attention scores are assigned to different parts of the input. The darker the color of the square, the larger the attention value.

The attention score given by $\boldsymbol{h}_j$ to $\boldsymbol{s}_i$ can be calculated as $\texttt{Att}\left(\boldsymbol{h}_j, \boldsymbol{s}_i\right), i = 1 \ldots, l_{\texttt{s}}, j = 1 \ldots, l_{\texttt{t}}$, where l_s is the length of the input sequence (i.e., source), and l_t is the length of the output sequence (i.e., target). Typically, attention scores are further normalized by a softmax function:

$$\alpha_{ji} = \frac{\texttt{exp}(\texttt{Att}\left(\boldsymbol{h}_j, \boldsymbol{s}_i\right))}{\sum_{i'=1,\ldots,l_s} \texttt{exp}\left(\texttt{Att}\left(\boldsymbol{h}_j, \boldsymbol{s}_{i'}\right)\right)}.$$

The corresponding output of each decoder step is a weighted sum of encoder vectors: $\boldsymbol{c}_{j+1} = \alpha_{j1}\boldsymbol{s}_1 + \cdots + \alpha_{jl_s}\boldsymbol{s}_{l_s}$.

There are many ways to formulate the attention function $\texttt{Att}\left(\boldsymbol{h}_j, \boldsymbol{s}_i\right)$,[2] such as:

- **Dot product** [Luong, Pham, and Manning, 2015]: $\texttt{Att}\left(\boldsymbol{h}_j, \boldsymbol{s}_i\right) = \boldsymbol{h}_j^T \boldsymbol{s}_i$.
- **Bilinear function** [Luong, Pham, and Manning, 2015]: $\texttt{Att}\left(\boldsymbol{h}_j, \boldsymbol{s}_i\right) = \boldsymbol{h}_j^T \boldsymbol{W} \boldsymbol{s}_i$.
- **Multilayer Perceptron** [Bahdanau, Cho, and Bengio, 2014]: $\texttt{Att}\left(\boldsymbol{h}_j, \boldsymbol{s}_i\right) = \boldsymbol{w}_{\texttt{Alignment}}^T \cdot \texttt{tanh}\left(\boldsymbol{U}\boldsymbol{h}_j + \boldsymbol{V}\boldsymbol{s}_i\right)$.

[2] https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

$\mathbf{W} \in \mathbb{R}^{d \times d}$, $\mathbf{U} \in \mathbb{R}^{d \times d}$ and $\mathbf{V} \in \mathbb{R}^{d \times d}$ are all learnable weight matrices of linear layers. $\boldsymbol{w}_{\texttt{Alignment}}^T \in \mathbb{R}^d$ is the alignment vector, which will be trained with the model.

F.2 Self-Attention

The self-attention is another attention mechanism, which has attracted attention with the proposal of Transformer [Vaswani et al., 2017]. The difference between attention and self-attention is that self-attention takes into account all information, i.e., all encoder states. More specifically, in self-attention, each input token has three representations, each for one role, i.e., *query*, *key*, and *value*. When the token acts as a query, use the *query* vector $\boldsymbol{q}$; when the token responds to a certain query, use the *key* vector $\boldsymbol{k}$ to calculate attention scores, the attention score is then multiplied by the *value* vector $\boldsymbol{v}$ to get the final output. The formula of self-attention is as follows:

$$\texttt{Att}(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) = \texttt{softmax}\left(\frac{\boldsymbol{q}\boldsymbol{k}^T}{\sqrt{d}}\right)\boldsymbol{v},$$

where d is the dimension of *key* and *value* vector. The output is computed as a weighted sum of *value* vectors, where the weight assigned to each value is measured by the "relevance" of the *key* to the *query*.

For a better understanding, Figure F.3 gives an example, in which there are three m-dimensional input tokens, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$. Each token is multiplied by $\boldsymbol{W}^Q \in \mathbb{R}^{m \times d}$, $\boldsymbol{W}^K \in \mathbb{R}^{m \times d}$, and $\boldsymbol{W}^V \in \mathbb{R}^{m \times d}$, three weight matrices to produce the corresponding *query*, *key* and *value* vectors. The output is a weighted sum of $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ and $\boldsymbol{v}_3$.

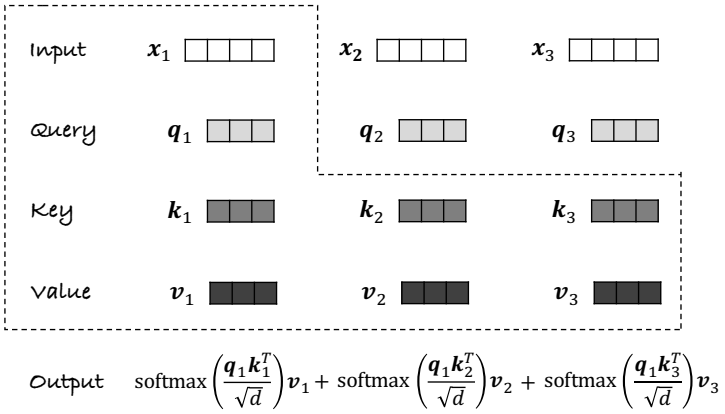


FIGURE F.3: An example of self-attention, adapted from[3].

[3] http://jalammar.github.io/illustrated-transformer/

Bibliography

Liu, Junhua et al. (2019). "Ipod: An industrial and professional occupations dataset and its applications to occupational data mining and analysis". In: *arXiv preprint arXiv:1910.10495*.

Singh, Arjun et al. (2017). "Gradescope: a fast, flexible, and fair system for scalable assessment of handwritten work". In: *Proceedings of the fourth (2017) acm conference on learning@ scale*, pp. 81–88.

Nie, Allen, Emma Brunskill, and Chris Piech (2021). "Play to Grade: Testing Coding Games as Classifying Markov Decision Process". In: *Advances in Neural Information Processing Systems* 34, pp. 1506–1518.

Smith, Brent and Greg Linden (2017). "Two decades of recommender systems at Amazon. com". In: *Ieee internet computing* 21.3, pp. 12–18.

Wang, Jizhe et al. (2018). "Billion-scale commodity embedding for e-commerce recommendation in alibaba". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 839–848.

McKinney, Scott Mayer et al. (2020). "International evaluation of an AI system for breast cancer screening". In: *Nature* 577.7788, pp. 89–94.

The Role of Digital Age in Recruitment (n.d.). https://currandaly.com/the-role-of-digital-age-in-recruitment/.

Okolie, Ugo Chuks and Ikechukwu Emmanuel Irabor (2017). "E-recruitment: practices, opportunities and challenges". In: *European Journal of Business and Management* 9.11, pp. 116–122.

81 LinkedIn Statistics You Need to Know in 2022 (n.d.). https://www.omnicoreagency.com/linkedin-statistics/.

Shaha, T Al-Otaibi and Ykhlef Mourad (2012). "A survey of job recommender systems". In: *International Journal of Physical Sciences* 7.29, pp. 5127–5142.

Malinowski, Jochen et al. (2006). "Matching people and jobs: A bilateral recommendation approach". In: *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*. Vol. 6. IEEE, pp. 137c–137c.

Lee, Danielle H and Peter Brusilovsky (2007). "Fighting information overflow with personalized comprehensive information access: A proactive job recommender". In: *Third International Conference on Autonomic and Autonomous Systems (ICAS'07)*. IEEE, pp. 21–21.

Upadhyay, Ashwani Kumar and Komal Khandelwal (2018). "Applying artificial intelligence: implications for recruitment". In: *Strategic HR Review*.

Van Esch, Patrick, J Stewart Black, and Joseph Ferolie (2019). "Marketing AI recruitment: The next phase in job application and selection". In: *Computers in Human Behavior* 90, pp. 215–222.

Javed, Faizan, Qinlong Luo, et al. (2015). "Carotene: A job title classification system for the online recruitment domain". In: *2015 IEEE First International Conference on Big Data Computing Service and Applications*. IEEE, pp. 286–293.

Javed, Faizan, Matt McNair, et al. (2016). "Towards a job title classification system". In: *arXiv preprint arXiv:1606.00917*.

Zhu, Chen et al. (2018). "Person-job fit: Adapting the right talent for the right job with joint representation learning". In: *ACM Transactions on Management Information Systems (TMIS)* 9.3, pp. 1–17.

Qin, Chuan et al. (2018). "Enhancing person-job fit for talent recruitment: An ability-aware neural network approach". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 25–34.

Al-Otaibi, Shaha T and Mourad Ykhlef (2012). "A survey of job recommender systems". In: *International Journal of Physical Sciences* 7.29, pp. 5127–5142.

Geyik, Sahin Cem, Stuart Ambler, and Krishnaram Kenthapadi (2019). "Fairness-aware ranking in search & recommendation systems with application to linkedin talent search". In: *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*, pp. 2221–2231.

Kenthapadi, Krishnaram, Benjamin Le, and Ganesh Venkataraman (2017). "Personalized job recommendation system at linkedin: Practical challenges and lessons learned". In: *Proceedings of the eleventh ACM conference on recommender systems*, pp. 346–347.

Bizer, Christian et al. (2005). "The impact of semantic web technologies on job recruitment processes". In: *Wirtschaftsinformatik 2005: eEconomy, eGovernment, eSociety*. Springer, pp. 1367–1381.

Malherbe, Emmanuel and Marie-Aude Aufaure (2016). "Bridge the terminology gap between recruiters and candidates: A multilingual skills base built from social media and linked data". In: *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, pp. 583–590.

Lundqvist, Karsten Øster, Keith Baker, and Shirley Williams (2008). "An ontological approach to competency management". In:

Fazel-Zarandi, Maryam and Mark S Fox (2009). "Semantic matchmaking for job recruitment: an ontology-based hybrid approach". In: *Proceedings of the 8th International Semantic Web Conference*. Vol. 525. 01, p. 2009.

Khobreh, Marjan (2017). "Ontology enhanced representing and reasoning of job specific knowledge to identify skill balance". In:

Kessler, Rémy, Guy Lapalme, and Eric Tondo (2016). *Génération d'une ontologie dans le domaine des ressources humaines*.

Bengio, Yoshua, Aaron Courville, and Pascal Vincent (2013). "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8, pp. 1798–1828.

Zhang, Shuai et al. (2019). "Deep learning based recommender system: A survey and new perspectives". In: *ACM Computing Surveys (CSUR)* 52.1, pp. 1–38.

Zhang, Ye and Byron Wallace (2015). "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1510.03820*.

Sundermeyer, Martin, Ralf Schlüter, and Hermann Ney (2012). "LSTM neural networks for language modeling". In: *Thirteenth annual conference of the international speech communication association*.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in neural information processing systems*, pp. 5998–6008.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473*.

Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton (2013). "Speech recognition with deep recurrent neural networks". In: *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee, pp. 6645–6649.

Yu, Feng et al. (2016). "A dynamic recurrent model for next basket recommendation". In: *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 729–732.

Hidasi, Balázs, Alexandros Karatzoglou, et al. (2015). "Session-based recommendations with recurrent neural networks". In: *arXiv preprint arXiv:1511.06939*.

Hidasi, Balázs, Massimo Quadrana, et al. (2016). "Parallel recurrent neural network architectures for feature-rich session-based recommendations". In: *Proceedings of the 10th ACM conference on recommender systems*, pp. 241–248.

Zhou, Jie, Ganqu Cui, Shengding Hu, et al. (2020). "Graph neural networks: A review of methods and applications". In: *AI Open* 1, pp. 57–81.

Wu, Zonghan et al. (2020). "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* 32.1, pp. 4–24.

Bian, Shuqing, Wayne Xin Zhao, et al. (2019). "Domain adaptation for person-job fit with transferable deep global match network". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4812–4822.

Schumann, Candice et al. (2020). "We need fairness and explainability in algorithmic hiring". In: *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Zhang, Yongfeng and Xu Chen (2018). "Explainable recommendation: A survey and new perspectives". In: *arXiv preprint arXiv:1804.11192*.

Arrieta, Alejandro Barredo et al. (2020). "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information fusion* 58, pp. 82–115.

Li, Liangyue et al. (2017). "Nemo: Next career move prediction with contextual embedding". In: *Proceedings of the 26th International Conference on World Wide Web Companion*, pp. 505–513.

Meng, Qingxin et al. (2019). "A hierarchical career-path-aware neural network for job mobility prediction". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 14–24.

Wang, Jingya et al. (2019). "DeepCarotene-Job Title Classification with Multi-stream Convolutional Neural Network". In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 1953–1961.

Zhao, Meng et al. (2015). "SKILL: A system for skill identification and normalization". In: *Twenty-Seventh IAAI Conference*.

Xu, Tong et al. (2018). "Measuring the popularity of job skills in recruitment market: A multi-criteria approach". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1.

Zhang, Denghui et al. (2019). "Job2Vec: Job title benchmarking with collective multi-view representation learning". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 2763–2771.

Dave, Vachik S et al. (2018). "A combined representation learning approach for better job and skill recommendation". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 1997–2005.

Hamilton, William L, Rex Ying, and Jure Leskovec (2017a). "Representation learning on graphs: Methods and applications". In: *arXiv preprint arXiv:1709.05584*.

Ospino, Carlos (2018). "Occupations: Labor Market Classifications, Taxonomies, and Ontologies in the 21st Century". In: *Inter-American Development Bank*.

Souza Pereira Moreira, Gabriel de, Felipe Ferreira, and Adilson Marques da Cunha (2018). "News session-based recommendations using deep neural networks". In: *Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems*, pp. 15–23.

Li, Jing et al. (2017). "Neural attentive session-based recommendation". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 1419–1428.

Fang, Hui, Danning Zhang, et al. (2020). "Deep Learning for Sequential Recommendation: Algorithms, Influential Factors, and Evaluations". In: *ACM Transactions on Information Systems (TOIS)* 39.1, pp. 1–42.

Kivimäki, Ilkka et al. (2013). "A graph-based approach to skill extraction from text". In: *Proceedings of TextGraphs-8 graph-based methods for natural language processing*, pp. 79–87.

Javed, Faizan, Phuong Hoang, et al. (2017). "Large-scale occupational skills normalization for online recruitment". In: *Twenty-ninth IAAI conference*.

Sayfullina, Luiza, Eric Malmi, and Juho Kannala (2018). "Learning representations for soft skill matching". In: *Analysis of Images, Social Networks and Texts: 7th International Conference, AIST 2018, Moscow, Russia, July 5–7, 2018, Revised Selected Papers 7*. Springer, pp. 141–152.

Khaouja, Imane et al. (2019). "Building a soft skill taxonomy from job openings". In: *Social Network Analysis and Mining* 9.1, pp. 1–19.

Gugnani, Akshay and Hemant Misra (2020). "Implicit skills extraction using document embedding and its use in job recommendation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 08, pp. 13286–13293.

Wings, Ivo, Rohan Nanda, and Kolawole John Adebayo (2021). "A context-aware approach for extracting hard and soft skills". In: *Procedia Computer Science* 193, pp. 163–172.

Zhou, Wenjun et al. (2016). "Quantifying skill relevance to job titles". In: *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 1532–1541.

Börner, Katy et al. (2018). "Skill discrepancies between research, education, and jobs reveal the critical need to supply soft skills for the data economy". In: *Proceedings of the National Academy of Sciences* 115.50, pp. 12630–12637.

Nigam, Amber et al. (2020). "SkillBERT:"Skilling" the BERT to classify skills!" In:

Liu, Mengshu et al. (2019). "Tripartite Vector Representations for Better Job Recommendation". In: *arXiv preprint arXiv:1907.12379*.

Decorte, Jens-Joris et al. (2021). "JobBERT: Understanding job titles through skills". In: *arXiv preprint arXiv:2109.09605*.

Zhang, Le et al. (2021). "Attentive heterogeneous graph embedding for job mobility prediction". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2192–2201.

Yamashita, Michiharu et al. (2022). "JAMES: Job Title Mapping with Multi-Aspect Embeddings and Reasoning". In: *arXiv preprint arXiv:2202.10739*.

Xu, Huang, Zhiwen Yu, Bin Guo, et al. (2018). "Extracting Job Title Hierarchy from Career Trajectories: A Bayesian Perspective." In: *IJCAI*, pp. 3599–3605.

Liu, Ye et al. (2016). "Fortune teller: predicting your career path". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1.

James, Charlotte et al. (2018). "Prediction of next career moves from scientific profiles". In: *arXiv preprint arXiv:1802.04830*.

Yang, Yang, De-Chuan Zhan, and Yuan Jiang (2018). "Which one will be next? An analysis of talent demission". In: *The 1st International Workshop on Organizational Behavior and Talent Analytics (Held in conjunction with KDD'18)*.

Xu, Huang, Zhiwen Yu, Jingyuan Yang, et al. (2016). "Talent circle detection in job transition networks". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 655–664.

Shen, Dazhong et al. (2018). "A joint learning approach to intelligent job interview assessment." In: *IJCAI*. Vol. 18, pp. 3542–3548.

Yan, Rui et al. (2019). "Interview choice reveals your preference on the market: to improve job-resume matching through profiling memories". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 914–922.

Luo, Yong et al. (2019). "ResumeGAN: An Optimized Deep Representation Learning Framework for Talent-Job Fit via Adversarial Learning". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1101–1110.

Bian, Shuqing, Xu Chen, et al. (2020). "Learning to Match Jobs with Resumes from Sparse Interaction Data using Multi-View Co-Teaching Network". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 65–74.

Lacic, Emanuel et al. (2020). "Using autoencoders for session-based job recommendations". In: *User Modeling and User-Adapted Interaction* 30.4, pp. 617–658.

Bastian, Mathieu et al. (2014). "Linkedin skills: large-scale topic extraction and inference". In: *Proceedings of the 8th ACM Conference on Recommender systems*, pp. 1–8.

Mikolov, Tomas, Kai Chen, et al. (2013). "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

Bojanowski, Piotr et al. (2017). "Enriching word vectors with subword information". In: *Transactions of the association for computational linguistics* 5, pp. 135–146.

Devlin, Jacob et al. (2018). "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805*.

Lee, Jinhyuk et al. (2020). "BioBERT: a pre-trained biomedical language representation model for biomedical text mining". In: *Bioinformatics* 36.4, pp. 1234–1240.

Beltagy, Iz, Kyle Lo, and Arman Cohan (2019). "SciBERT: A pretrained language model for scientific text". In: *arXiv preprint arXiv:1903.10676*.

Peters, Matthew E et al. (2018). "Deep contextualized word representations". In: *arXiv preprint arXiv:1802.05365*.

Mikolov, Tomas, Ilya Sutskever, et al. (2013). "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*, pp. 3111–3119.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Edwards, Jeffrey R (1991). *Person-job fit: A conceptual integration, literature review, and methodological critique.* John Wiley & Sons.

Sekiguchi, Tomoki (2004). "Person-organization fit and person-job fit in employee selection: A review of the literature". In: *Osaka keidai ronshu* 54.6, pp. 179–196.

Makhzani, Alireza et al. (2015). "Adversarial autoencoders". In: *arXiv preprint arXiv:1511.05644*.

Cai, Hongyun, Vincent W Zheng, and Kevin Chen-Chuan Chang (2018). "A comprehensive survey of graph embedding: Problems, techniques, and applications". In: *IEEE Transactions on Knowledge and Data Engineering* 30.9, pp. 1616–1637.

Zhang, Daokun et al. (2018). "Network representation learning: A survey". In: *IEEE transactions on Big Data* 6.1, pp. 3–28.

Liben-Nowell, David and Jon Kleinberg (2007). "The link-prediction problem for social networks". In: *Journal of the American society for information science and technology* 58.7, pp. 1019–1031.

Lü, Linyuan and Tao Zhou (2011). "Link prediction in complex networks: A survey". In: *Physica A: statistical mechanics and its applications* 390.6, pp. 1150–1170.

Bhagat, Smriti, Graham Cormode, and S Muthukrishnan (2011). "Node classification in social networks". In: *Social network data analytics*. Springer, pp. 115–148.

Cavallari, Sandro et al. (2017). "Learning community embedding with community detection and node embedding on graphs". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 377–386.

Wang, Xiao, Peng Cui, et al. (2017). "Community preserving network embedding". In: *Thirty-first AAAI conference on artificial intelligence*.

Shi, Chuan et al. (2018). "Heterogeneous information network embedding for recommendation". In: *IEEE Transactions on Knowledge and Data Engineering* 31.2, pp. 357–370.

Tang, Jian, Jingzhou Liu, et al. (2016). "Visualizing large-scale and high-dimensional data". In: *Proceedings of the 25th international conference on world wide web*, pp. 287–297.

Le, Quoc and Tomas Mikolov (2014). "Distributed representations of sentences and documents". In: *International conference on machine learning*. PMLR, pp. 1188–1196.

Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena (2014). "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710.

Grover, Aditya and Jure Leskovec (2016). "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864.

Kipf, Thomas N and Max Welling (2016). "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907*.

Hamilton, William L, Rex Ying, and Jure Leskovec (2017b). "Inductive representation learning on large graphs". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035.

Veličković, Petar, Guillem Cucurull, et al. (2017). "Graph attention networks". In: *arXiv preprint arXiv:1710.10903*.

Schlichtkrull, Michael et al. (2018). "Modeling relational data with graph convolutional networks". In: *European semantic web conference*. Springer, pp. 593–607.

Tang, Jian, Meng Qu, et al. (2015). "Line: Large-scale information network embedding". In: *Proceedings of the 24th international conference on world wide web*, pp. 1067–1077.

Veličković, Petar, William Fedus, et al. (2018). "Deep graph infomax". In: *arXiv preprint arXiv:1809.10341*.

Dong, Yuxiao, Nitesh V Chawla, and Ananthram Swami (2017). "metapath2vec: Scalable representation learning for heterogeneous networks". In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 135–144.

Chen, Hongxu et al. (2018). "PME: projected metric embedding on heterogeneous networks for link prediction". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1177–1186.

Wang, Xiao, Houye Ji, et al. (2019). "Heterogeneous graph attention network". In: *The World Wide Web Conference*, pp. 2022–2032.

Gilmer, Justin et al. (2017). "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR, pp. 1263–1272.

Adomavicius, Gediminas and Alexander Tuzhilin (2005). "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions". In: *IEEE transactions on knowledge and data engineering* 17.6, pp. 734–749.

Ricci, Francesco, Lior Rokach, and Bracha Shapira (2010). "Introduction to recommender systems handbook". In: *Recommender systems handbook*. Springer, pp. 1–35.

Breese, John S, David Heckerman, and Carl Kadie (2013). "Empirical analysis of predictive algorithms for collaborative filtering". In: *arXiv preprint arXiv:1301.7363*.

Fkih, Fethi (2022). "Similarity measures for Collaborative Filtering-based Recommender Systems: Review and experimental comparison". In: *Journal of King Saud University-Computer and Information Sciences* 34.9, pp. 7645–7669.

Salton, Gerard and Christopher Buckley (1988). "Term-weighting approaches in automatic text retrieval". In: *Information processing & management* 24.5, pp. 513–523.

Jin, Xin, Yanzan Zhou, and Bamshad Mobasher (2005). "A maximum entropy web recommendation system: combining collaborative and content features". In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 612–617.

Wang, Chong and David M Blei (2011). "Collaborative topic modeling for recommending scientific articles". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 448–456.

Musto, Cataldo et al. (2015). "Word Embedding Techniques for Content-based Recommender Systems: An Empirical Evaluation." In: *Recsys posters*.

— (2016). "Learning word embeddings from wikipedia for content-based recommender systems". In: *Advances in Information Retrieval: 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20–23, 2016. Proceedings 38*. Springer, pp. 729–734.

Burke, Robin (2002). "Hybrid recommender systems: Survey and experiments". In: *User modeling and user-adapted interaction* 12.4, pp. 331–370.

— (2007). "Hybrid web recommender systems". In: *The adaptive web*. Springer, pp. 377–408.

Wang, Shoujin, Liang Hu, et al. (2019). "Sequential recommender systems: challenges, progress and prospects". In: *arXiv preprint arXiv:2001.04830*.

Wang, Shoujin, Longbing Cao, and Yan Wang (2019). "A survey on session-based recommender systems". In: *arXiv preprint arXiv:1902.04864*.

Quadrana, Massimo, Paolo Cremonesi, and Dietmar Jannach (2018). "Sequence-aware recommender systems". In: *ACM Computing Surveys (CSUR)* 51.4, pp. 1–36.

Ludewig, Malte and Dietmar Jannach (2018). "Evaluation of session-based recommendation algorithms". In: *User Modeling and User-Adapted Interaction* 28.4-5, pp. 331–390.

He, Ruining and Julian McAuley (2016). "Fusing similarity models with markov chains for sparse sequential recommendation". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, pp. 191–200.

Rendle, Steffen, Christoph Freudenthaler, and Lars Schmidt-Thieme (2010). "Factorizing personalized markov chains for next-basket recommendation". In: *Proceedings of the 19th international conference on World wide web*, pp. 811–820.

Jannach, Dietmar and Malte Ludewig (2017). "When recurrent neural networks meet the neighborhood for session-based recommendation". In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pp. 306–310.

Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami (1993). "Mining association rules between sets of items in large databases". In: *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pp. 207–216.

Hidasi, Balázs and Alexandros Karatzoglou (2018). "Recurrent neural networks with top-k gains for session-based recommendations". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 843–852.

Lerche, Lukas, Dietmar Jannach, and Malte Ludewig (2016). "On the value of reminders within e-commerce recommendations". In: *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization*, pp. 27–35.

Bonnin, Geoffray and Dietmar Jannach (2014). "Automated generation of music playlists: Survey and experiments". In: *ACM Computing Surveys (CSUR)* 47.2, pp. 1–35.

Kamehkhosh, Iman, Dietmar Jannach, and Malte Ludewig (2017). "A Comparison of Frequent Pattern Techniques and a Deep Learning Method for Session-Based Recommendation." In: *RecTemp@ RecSys*, pp. 50–56.

Linden, Greg, Brent Smith, and Jeremy York (2003). "Amazon. com recommendations: Item-to-item collaborative filtering". In: *IEEE Internet computing* 7.1, pp. 76–80.

Su, Xiaoyuan and Taghi M Khoshgoftaar (2009). "A survey of collaborative filtering techniques". In: *Advances in artificial intelligence* 2009.

Quadrana, Massimo, Alexandros Karatzoglou, et al. (2017). "Personalizing session-based recommendations with hierarchical recurrent neural networks". In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pp. 130–137.

Cho, Kyunghyun et al. (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078*.

Tan, Yong Kiam, Xinxing Xu, and Yong Liu (2016). "Improved recurrent neural networks for session-based recommendations". In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pp. 17–22.

Szegedy, Christian et al. (2015). "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Tang, Jiaxi and Ke Wang (2018). "Personalized top-n sequential recommendation via convolutional sequence embedding". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pp. 565–573.

Tuan, Trinh Xuan and Tu Minh Phuong (2017). "3D convolutional networks for session-based recommendation with content features". In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pp. 138–146.

Kang, Wang-Cheng and Julian McAuley (2018). "Self-attentive sequential recommendation". In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, pp. 197–206.

Liu, Qiao et al. (2018). "STAMP: short-term attention/memory priority model for session-based recommendation". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1831–1839.

Sun, Fei et al. (2019). "BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1441–1450.

Zhou, Jie, Ganqu Cui, Zhengyan Zhang, et al. (2018). "Graph neural networks: A review of methods and applications". In: *arXiv preprint arXiv:1812.08434*.

Wu, Shiwen et al. (2022). "Graph neural networks in recommender systems: a survey". In: *ACM Computing Surveys* 55.5, pp. 1–37.

Wu, Shu et al. (2019). "Session-based recommendation with graph neural networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33, pp. 346–353.

Li, Yujia et al. (2015). "Gated graph sequence neural networks". In: *arXiv preprint arXiv:1511.05493*.

Fang, Hui, Guibing Guo, et al. (2019). "Deep learning-based sequential recommender systems: Concepts, algorithms, and evaluations". In: *International Conference on Web Engineering*. Springer, pp. 574–577.

Fauconnier, Jean-Philippe (2015). *French Word Embeddings*. URL: http://fauconnier.github.io.

Xiao, Han (2018). *bert-as-service*. https://github.com/hanxiao/bert-as-service.

Wang, Minjie et al. (2019). "Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks". In: *arXiv preprint arXiv:1909.01315*.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Valentini, Giorgio et al. (2021). "Het-node2vec: second order random walk sampling for heterogeneous multigraphs embedding". In: *arXiv preprint arXiv:2101.01425*.

Van der Maaten, Laurens and Geoffrey Hinton (2008). "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11.

Zhang, Tianqi et al. (2020). "CommDGI: community detection oriented deep graph infomax". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 1843–1852.

Du, Lun et al. (2018). "Galaxy Network Embedding: A Hierarchical Community Structure Preserving Approach." In: *IJCAI*, pp. 2079–2085.

Long, Qingqing et al. (2019). "Hierarchical community structure preserving network embedding: A subspace approach". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 409–418.

Bhowmick, Ayan Kumar et al. (2020). "Louvainne: Hierarchical louvain method for high quality and scalable network embedding". In: *Proceedings of the 13th International Conference on Web Search and Data Mining*, pp. 43–51.

Wang, Meng et al. (2015). "Community detection in social networks: an in-depth benchmarking study with a procedure-oriented framework". In: *Proceedings of the VLDB Endowment* 8.10, pp. 998–1009.

Blondel, Vincent D et al. (2008). "Fast unfolding of communities in large networks". In: *Journal of statistical mechanics: theory and experiment* 2008.10, P10008.

Zhong, Zhiqiang, Cheng-Te Li, and Jun Pang (2020). "Hierarchical Message-Passing Graph Neural Networks". In: *arXiv preprint arXiv:2009.03717*.

Tripathi, Pooja, Ruchi Agarwal, and Tanushi Vashishtha (2016). "Review of job recommender system using big data analytics". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, pp. 3773–3777.

Huang, Po-Sen et al. (2013). "Learning deep structured semantic models for web search using clickthrough data". In: *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pp. 2333–2338.

Wu, Chuhan et al. (2019). "NPA: neural news recommendation with personalized attention". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2576–2584.

Gabriel De Souza, P Moreira, Dietmar Jannach, and Adilson Marques Da Cunha (2019). "Contextual hybrid session-based news recommendation with recurrent neural networks". In: *IEEE Access* 7, pp. 169185–169203.

Abel, Fabian et al. (2017). "Recsys challenge 2017: Offline and online evaluation". In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pp. 372–373.

Maas, Andrew L, Awni Y Hannun, and Andrew Y Ng (2013). "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. Citeseer, p. 3.

Grave, Edouard et al. (2018). "Learning Word Vectors for 157 Languages". In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Naseem, Usman et al. (2021). "A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models". In: *Transactions on Asian and Low-Resource Language Information Processing* 20.5, pp. 1–35.

Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Bengio, Yoshua, Réjean Ducharme, et al. (2003). "A neural probabilistic language model". In: *The journal of machine learning research* 3, pp. 1137–1155.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

Luong, Minh-Thang, Hieu Pham, and Christopher D Manning (2015). "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025*.