Zhiyuan Liu · Yankai Lin
Maosong Sun *Editors*

Representation Learning for Natural Language Processing

Second Edition

OPEN ACCESS

Springer

Representation Learning for Natural Language Processing

Zhiyuan Liu • Yankai Lin • Maosong Sun
Editors

Representation Learning for Natural Language Processing

Second Edition

Springer

Editors
Zhiyuan Liu
Department of Computer Science
and Technology
Tsinghua University
Beijing, China

Yankai Lin
Gaoling School of Artificial Intelligence
Renmin University of China
Beijing, China

Maosong Sun
Department of Computer Science
and Technology
Tsinghua University
Beijing, China

This work was supported by Tsinghua University

ISBN 978-981-99-1599-6 ISBN 978-981-99-1600-9 (eBook)
https://doi.org/10.1007/978-981-99-1600-9

© The Editor(s) (if applicable) and The Author(s) 2020, 2023. This book is an open access publication.
Open Access This book is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.
The images or other third party material in this book are included in the book's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the book's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Paper in this product is recyclable.

Preface

In conventional natural language processing (NLP) systems, language items such as words and phrases are handled as distinct symbols. Many classical methods, such as n-gram and bag-of-words models, were proposed and have been widely used until now. All these methods take words as the minimum units for semantic representation, either used to estimate the conditional probabilities of the next word given previous words (e.g., n-gram) or used to represent semantic meanings of text (e.g., bag-of-words models). Even when people find it necessary to model word meanings, they either manually build linguistic knowledge bases such as WordNet or use context words to represent word meaning (i.e., distributional representation). All these semantic representation methods are still based on symbols!

With the development of NLP techniques for years, it has been realized that symbolic representation has caused many issues in NLP. First, symbolic representation always suffers from the data sparsity problem. Take statistical NLP methods, such as n-gram with large-scale corpora, for example. Due to the intrinsic power-law distribution of words, the performance will decay dramatically for those few-shot words, even after many smoothing methods have been developed to calibrate the estimated probabilities.

There are also multiple-grained items in natural languages ranging from words, phrases, and sentences to documents. It is impossible to find a unified symbol set to represent the semantic meanings for all of them. Moreover, many NLP tasks require semantic relatedness between language items at different levels. For example, we have to measure semantic relatedness between words/phrases and documents in Information Retrieval. Due to the absence of a unified scheme for semantic representation, distinct approaches were proposed for different tasks in NLP, which sometimes makes NLP seem not a compatible community. More than that, a deep understanding of natural language requires rich human knowledge, and it is non-trivial for symbolic representation in NLP to identify and incorporate sophisticated external knowledge.

As an alternative approach to symbolic representation, distributed representation was initially proposed by Geoffrey E. Hinton in a technique report in 1984. The report was then included in the well-known two-volume book *Parallel Distributed*

Processing (PDP) that introduced neural networks to model human cognition and intelligence. According to this report, distributed representation is inspired by the neural computation scheme of humans and other animals, and the essential idea is as follows:

> Each entity is represented by a pattern of activity distributed over many computing elements, and each computing element is involved in representing many different entities.

It means that a language item will be represented by multiple neurons, and each neuron is involved in representing many items. It also indicates the meaning of *distributed* in distributed representation. In contrast to distributed representation, another view assumes one neuron corresponds to a specific object. That is, there may be an individual neuron only activated by one specific object, such as someone's grandmother, known as the grandmother-cell hypothesis or local representation. We can see the direct connection between the grandmother-cell hypothesis and symbolic representation.

About 20 years after distributed representation was initiated, Yoshua Bengio presented the neural probabilistic language model for natural languages in 2003, where words are represented as low-dimensional and real-valued vectors based on the idea of distributed representation. However, it was until 2013 that a simpler and more efficient framework *word2vec* was proposed to learn word distributed representations from large-scale corpora. We come to the popularity of distributed representation and neural network techniques in NLP. The recent 10 years witnessed significant improvement in almost all NLP tasks with the support of distributed representation, especially with the development of the neural architecture Transformer in 2017 and pre-trained language models after 2018.

This book aims to overview recent advances in distributed representation learning for NLP. It covers how representation learning takes part in various topics related to NLP, paying particular attention to why representation learning can improve NLP. As an exciting and active research area, we also focus on what remaining challenges are still not well addressed by distributed representation.

Book Organization

This book is organized into 14 chapters with 4 parts. The first part depicts key components in NLP and how representation learning works for them. This part includes (1) the basics of representation learning and why it is vital for NLP (Chap. 1); (2) a comprehensive review of representation learning techniques on multiple-grained entries in NLP, including word representation (Chap. 2), phrase representation as known as compositional semantics (Chap. 3), and sentence and document representation (Chap. 4); (3) the most advanced techniques in recent years, pre-trained models (Chap. 5).

The second part presents representation learning for those topics closely related to NLP. This part includes (1) graph representation handling structured information

around natural languages such as relations of either sentences or documents (Chap. 6); (2) cross-modal representation connecting natural languages to other modalities such as visual data (Chap. 7); (3) robust representation talking about the vulnerability of distributed representation to backdoor attack, adversarial attack, and out-of-distribution data (Chap. 8).

A deep understanding of natural languages requires the support of rich human languages. The third part is about knowledge representation and the framework of knowledge-guided NLP. This part includes (1) a general introduction to knowledge representation with entity-based world knowledge as the example (Chap. 9); (2) linguistic and commonsense knowledge representation with the form of sememes, which are defined as the minimum semantic units in human languages (Chap. 10); (3) domain knowledge representation, taking legal (Chap. 11) and biomedical (Chap. 12) domains as examples.

In the fourth part, we take an open-resource platform as an example to introduce big model systems for representation learning, including pre-training, fine-tuning, model compression, and inference (Chap. 13). Finally, we outlook the future research directions of representation learning for NLP by summarizing ten key problems of pre-trained models (Chap. 14).

Although the book is about representation learning for NLP, those theories and algorithms can also be applied in other related domains, such as machine learning, social network analysis, semantic Web, information retrieval, data mining, and computational biology.

Book Cover

We designed the book cover to correspond to three revolutionized stages of cognition and representation in human history. It is an oracle bone divided into three parts.

The left part shows oracle scripts, the earliest known form of Chinese writing characters used on oracle bones in the late 1200 BC. It represents the emergence of human languages, especially writing systems. We regard this as the first representation revolution of human beings about the world, i.e., **representation symbolization**. It makes information and knowledge transmitted from person to person and from generation to generation.

The upper right part shows the digitalized representation of information and signals. Since the invention of electronic computers in the 1940s, big data and knowledge can be efficiently represented and processed in computer programs. We regard this as the second representation revolution of human beings about the world, i.e., **representation digitalization**. It enables large-scale information and knowledge to be stored, accumulated, searched, and utilized with computer systems.

The bottom right part shows the distributed representation in artificial neural networks initiated in the 1980s. As the representation basis of deep learning, it has extensively revolutionized many fields in artificial intelligence, including NLP, CV,

and speech recognition, since the 2010s. We regard this as the third representation revolution of human beings about the world, i.e., **representation intellectualization**. It enables sophisticated knowledge to be effectively acquired, represented, and utilized in AI systems. This book focuses on the theory, methods, and applications of distributed representation learning in natural language processing.

Note for the Second Edition

Our team has studied representation learning in NLP since 2014, starting with word representation, graph representation, and knowledge representation. As shown throughout the 2010s, distributed representation and deep learning have brought a significant paradigm shift to NLP. This book aims to summarize and showcase critical advances in representation learning in NLP from our point of view, also consisting of related works from our team in those directions. The book's first edition was published in 2020, and its preparation can be traced back to 2016; it is a team work with many efforts of the professors, graduate students, and research assistants, as listed in the Acknowledgments of the 2020 edition.

We have witnessed the great success of Transformer and pre-trained models in recent years, which further enhances our knowledge and understanding of representation learning. We regard pre-training-fine-tuning as another paradigm shift to NLP after deep learning. Hence, we prepare the second edition to reflect these critical changes and advances. There are the following critical modifications as compared to the first edition:

1. **Book Organization.** From our experience organizing the 2020 edition, we realize limited persons cannot master all detailed advances of each direction of representation learning in NLP. In this edition, we invite young researchers and senior graduate students to work with us together on writing or updating specific chapters. All of them have rich research experiences on the topics of their participating chapters. We work together and frequently discuss to ensure all chapters are consistent with each other, following the same goal as elaborated in Chap. 1. We also work together to summarize the key problems of pre-trained models as an outlook to representation learning, as shown in Chap. 14.

   We also optimize writing guidelines and improve writing styles in many aspects. For example, in the 2020 edition, we sometimes tended to emphasize those related works from our team, which make the structure of some parts not so fluent and coherent. In this edition, we comprehensively revise these aspects with one goal, i.e., to thoroughly introduce the key advances of representation learning and help readers grasp the development trends from the past and the present to the future.
2. **Supplements and Updates.** We supplement five new chapters for pre-trained models and other emerging topics. For the recent advances in pre-trained models, we introduce the general knowledge about pre-trained models in Chap. 5 and take

an open-source platform OpenBMB as an example to introduce the programming details of training, tuning, compressing, and inference of big models in Chap. 13. We also add three new chapters on new topics of representation learning, including robustness (Chap. 8) and new knowledge types of the legal domain (Chap. 11) and the biomedical domain (Chap. 12).

Pre-training techniques have a revolutionary impact on almost all areas. With the new perspective, we also have a deeper understanding of representation learning. Hence we update all remaining chapters by either re-organizing the chapter structure (Chaps. 2, 3, 4 and 9) or providing recent advances (Chaps. 6, 7 and 10). Moreover, we merge document representation into sentence representation to better demonstrate the advances in neural language models. Based on the updates of all chapters, we improve the general introduction to representation learning and NLP in Chap. 1, such as providing clues about the intellectual origins of distributed representation.

3. **Corrections.** The first edition of the book was prepared for more than four years from 2016 to 2020, which could have not been published without the contributions of so many students and contributors in our team as indicated in the Acknowledgements. Meanwhile, even after several rounds of revision and proofreading before publication, we still find some inconsistent expressions and equations when integrating chapters, unintentional overlaps with others' articles introduced from the initial draft prepared by a research assistant, and other mistakes in the published version. We sincerely apologize for these mistakes. In this new edition, we have updated all contents and corrected all issues we found. In the future, we will also try our best to make these cases never happen again by writing, proofreading, and applying duplicate detection more carefully to each material released by our team.

Prerequisites

This book is designed for advanced undergraduate and graduate students, post-doctoral fellows, researchers, lecturers, industrial engineers, and anyone interested in representation learning, NLP, knowledge engineering, and AI. This is not a textbook, and we don't introduce basic knowledge. We expect the readers to have prior knowledge of Probability, Linear Algebra, and Machine Learning.

We recommend that readers interested in NLP read the first part (Chaps. 1–5) in sequence. The remaining parts can be read according to readers' interests. Many areas of representation learning are still evolving quickly. Hence, we list some books, reviews, and resources at the end of each chapter for readers to learn more about the topics.

Contact Information

In this book, we try our best to summarize the advances of representation learning in NLP. When preparing the 2020 edition and this edition, we grow to acknowledge that we cannot be experts in each aspect of this area. We may unintentionally conduct wrong summarizations, omit critical works, make incorrect predictions, or introduce other flaws. We are always expecting feedback, corrections, and suggestions on the book from readers, and improving the book and our research accordingly. The messages may be sent to `liuzy@tsinghua.edu.cn` or raised as issues on the following GitHub repository,

https://github.com/thunlp/Book_RL4NLP

We will keep updating and supplementing the book on the GitHub repository, where readers can find up-to-date news, errata, and updates about the book.

Beijing, China
January, 2023

Zhiyuan Liu
Yankai Lin
Maosong Sun

Acknowledgments

Acknowledgments for the Second Edition

In the second edition, we list the contributors at the end of each chapter respectively. Here we express our thanks to those who have offered help and support to the whole book. We thank Chaojun Xiao for unifying the styles of figures, thank Shengding Hu for making the notation table, and unifying the notations and organizing references across chapters, and thank Zhenning Dai for making the acronym table. We thank Ms. Ruiqi Shao for designing the new book cover.

We also thank the Springer senior editor Dr. Celine Lanlan Chang, for making the second edition of the book happen and for enlightening suggestions on improvements to the first edition.

Acknowledgments for the First Edition

The authors are very grateful to the contributions of our students and research collaborators who have prepared initial drafts of some chapters or have given us comments, suggestions, and corrections. We list main contributors for preparing initial drafts of each chapter as follows:

- Chapter 1: Tianyu Gao, Zhiyuan Liu.
- Chapter 2: Lei Xu, Yankai Lin.
- Chapter 3: Yankai Lin, Yang Liu.
- Chapter 4: Yankai Lin, Zhengyan Zhang, Cunchao Tu, Hongyin Luo.
- Chapter 5: Yankai Lin, Zhenghao Liu, Haozhe Ji.
- Chapter 6: Fanchao Qi, Chenghao Yang.
- Chapter 7: Ruobing Xie, Xu Han.
- Chapter 8: Cheng Yang, Jie Zhou, Zhengyan Zhang.
- Chapter 9: Ji Xin, Yuan Yao, Deming Ye, Hao Zhu.

- Chapter 10: Xu Han, Zhengyan Zhang, Cheng Yang.
- Chapter 11: Cheng Yang, Zhiyuan Liu.

For the whole book, we thank Chaojun Xiao and Zhengyan Zhang for drawing model figures, thank Chaojun Xiao for unifying the styles of figures and tables in the book, thank Shengding Hu for making the notation table and unifying the notations across chapters, thank Jingcheng Yuzhi and Chaojun Xiao for organizing the format of reference, thank Jingcheng Yuzhi, Jiaju Du, Haozhe Ji, Sicong Ouyang, and Ayana for the first-round proofreading, and thank Weize Chen, Ganqu Cui, Bowen Dong, Tianyu Gao, Xu Han, Zhenghao Liu, Fanchao Qi, Guangxuan Xiao, Cheng Yang, Yuan Yao, Shi Yu, Yuan Zang, Zhengyan Zhang, Haoxi Zhong, and Jie Zhou for the second-round proofreading. We also thank Cuncun Zhao for designing the book cover.

In this book, there is a specific chapter talking about sememe knowledge representation. Many works in this chapter are carried out by our research group. These works have received great encouragement from the inventor of HowNet, Mr. Zhendong Dong, who died at 82 on February 28, 2019. HowNet is the great linguistic and commonsense knowledge base composed by Mr. Dong for about 30 years. At the end of his life, he and his son Mr. Qiang Dong decided to collaborate with us and released the open-source version of HowNet, OpenHowNet. As a pioneer of machine translation in China, Mr. Zhendong Dong devoted his whole life to natural language processing. He will be missed by all of us forever.

We thank our colleagues and friends, Yang Liu and Juanzi Li at Tsinghua University, and Peng Li at Tencent Wechat, who offered close and frequent discussions which substantially improve this book. We also want to express our special thanks to Prof. Bo Zhang. His insights to deep learning and representation learning, and sincere encouragements to our research of representation learning on NLP, have greatly stimulated us to move forward with more confidence and passion.

We proposed the plan of this book in 2015 after discussing it with the Springer senior editor Dr. Celine Lanlan Chang. As the first of the time of preparing a technique book, we were not expecting it took so long to finish this book. We thank Celine for providing insightful comments and incredible patience to the preparation of this book. We are also grateful to Springer's Assistant Editor Jane Li, for offering invaluable help during manuscript preparation.

Finally, we give our appreciations to our organizations, Department of Computer Science and Technology at Tsinghua University, Institute for Artificial Intelligence at Tsinghua University, Beijing Academy of Artificial Intelligence (BAAI), Chinese Information Processing Society of China, and Tencent Wechat, who have provided outstanding environment, supports, and facilities for preparing this book.

Contents

Contributors

Ganqu Cui Department of Computer Science and Technology, Tsinghua University, Beijing, China

Ning Ding Department of Computer Science and Technology, Tsinghua University, Beijing, China

Xu Han Department of Computer Science and Technology, Tsinghua University, Beijing, China

Shengding Hu Department of Computer Science and Technology, Tsinghua University, Beijing, China

Yankai Lin Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China

Zhiyuan Liu Department of Computer Science and Technology, Tsinghua University, Beijing, China

Yujia Qin Department of Computer Science and Technology, Tsinghua University, Beijing, China

Maosong Sun Department of Computer Science and Technology, Tsinghua University, Beijing, China

Chaojun Xiao Department of Computer Science and Technology, Tsinghua University, Beijing, China

Cheng Yang School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China

Yuan Yao Department of Computer Science and Technology, Tsinghua University, Beijing, China

Zheni Zeng Department of Computer Science and Technology, Tsinghua University, Beijing, China

Guoyang Zeng ModelBest Inc. and OpenBMB, Beijing, China

Zhengyan Zhang Department of Computer Science and Technology, Tsinghua University, Beijing, China

Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long Short-Term Memory
BIO	The Begin-Inside-Outside tagging format for sequence tagging
BOW	Bag-of-Words
CBOW	Continuous Bag-of-Words
CNN	Convolutional Neural Network
COT	Chain-of-Thought
CPM	Chinese Pre-trained Model introduced by BAAI and Tsinghua
CPU	Central Processing Unit
CRF	Conditional Random Field
CV	Computer Vision
DNA	Deoxyribonucleic Acid
EM	Expectation-Maximization Algorithm
FNN	Feed-Forward Neural Network
GCN	Graph Convolutional Network
GPT	Generative Pre-trained Model
GPU	Graph Processing Unit
GRU	Gated Recurrent Unit
HCI	Human-Computer Interaction
HMM	Hidden Markov Model
IDF	Inverse Document Frequency
IR	Information Extraction
KB	Knowledge Base
KG	Knowledge Graph
KRL	Knowledge Representation Learning
LDA	Latent Dirichlet Allocation
LM	Language Model
LSTM	Long Short-Term Memory
MLM	Mask Language Modeling

MLP	Multi-layer Perceptron
MoE	Mixture-of-Experts
MT	Machine Translation
NER	Named Entity Recognition
NLI	Natural Language Inference
NLP	Natural Language Processing
NPLM	Neural Probabilistic Language Model
ODE	Ordinary Differential Equations
OFAI	Old-Fashioned AI
PDP	Parallel Distributed Processing
PLM	Pre-trained Language Model
PMI	Pointwise Mutual Information
POS	Part-of-Speech
PTM	Pre-trained Model
QA	Question Answering
RE	Relation Extraction
ReLU	Rectified Linear Unit activation function
RNA	Ribonucleic Acid
RNN	Recurrent Neural Network
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TF-IDF	Term Frequency-Inverse Document Frequency
VQA	Visual Question Answering

Symbols and Notations

$\odot$	element-wise multiplication
$\otimes$	Kronecker product of two matrices
$*$	convolution operator
$\wedge$	logical and
$\leftarrow$	update the value of left-hand side value with right-hand side value
$\propto$	proportional to
∇	gradient of a function
$\frac{\partial f}{\partial x}$	partial derivative of a function
$^\top$	transpose
$\|\cdot\|_p$	p norm of a vector
$\|\cdot\|$	size of a set
argmax	find the arguments that give the maximum value from a target function
argmin	find the arguments that give the minimum value from a target function
lim	limitation
$\mathcal{L}(\cdot)$	loss function
$O(\cdot)$	complexity of an algogrithm
$s(\cdot)$	similarity, especially cosine similarity
$\exp(\cdot)$	exponential function
$\text{ATT}(\cdot)$	self-attention layer
$\text{concat}(\cdot; \ldots; \cdot)$	concatenate a list of vectors/tensors
$\text{LN}(\cdot)$	layer norm layer in transformers
$\text{FFN}(\cdot)$	feed-forward network in transformers
$\text{MLP}(\cdot)$	multilayer perceptron
$\text{Softmax}(\cdot)$	softmax function
$\text{Sigmoid}(\cdot)$	sigmoid function
$\tanh(\cdot)$	hyperbolic tangent function
$\mathcal{N}(x; \mu, \Sigma)$	normal distribution
$\mathbf{h}$	hidden representation

$\mathbf{W}$;$\mathbf{M}$	weight matrix
Θ	model parameter
P	probability
$\mathcal{K}$	external knowledge
$\mathcal{R}$	relationship rule
$\mathbb{R}$	set of real numbers
$\mathbb{Z}_+$	set of positive integers
`[CLS]`	a special token to aggregate semantics in PTMs
`[MASK]`	a special token for the masked tokens in PTMs

Chapter 1
Representation Learning and NLP

Zhiyuan Liu and Maosong Sun

Abstract Natural language processing (NLP) aims to build linguistic-specific programs for machines to understand and use human languages. Conventional NLP methods heavily rely on feature engineering to constitute semantic representations of text, requiring careful design and considerable expertise. Meanwhile, representation learning aims to automatically build informative representations of raw data for further application and achieves significant success in recent years. This chapter presents a brief introduction to representation learning, including its motivation, history, intellectual origins, and recent advances in both machine learning and NLP.

1.1 Motivation

Machine learning addresses the problem of automatically learning computer programs from data. A typical machine learning system consists of three components [13]:

$$\text{Machine Learning} = \text{Representation} + \text{Objective} + \text{Optimization}.$$

We first transform helpful information from raw data into internal representations such as feature vectors to build an effective machine learning system. Afterward, by designing appropriate objective functions, we can employ optimization algorithms to find the optimal parameter settings for the system.

Data representation methods determine what and how valuable information can be extracted from raw data for further classification or prediction. If more information is transformed from raw data to feature representations, the performance of classification or prediction will be better. Hence, data representation is a crucial component of supporting effective machine learning.

Z. Liu (✉) · M. Sun
Department of Computer Science and Technology, Tsinghua University, Beijing, China
e-mail: liuzy@tsinghua.edu.cn; sms@tsinghua.edu.cn

© The Author(s) 2023

Z. Liu et al. (eds.), *Representation Learning for Natural Language Processing*,
https://doi.org/10.1007/978-981-99-1600-9_1

Conventional machine learning systems adopt careful feature engineering as preprocessing to build feature representations from raw data. Feature engineering needs careful design and considerable expertise. A specific task usually requires customized algorithms for feature engineering, which makes the process labor-intensive, time-consuming, and inflexible.

Representation learning aims to learn informative representations of objects from raw data automatically. The learned representations can be further fed as input to machine learning systems for prediction or classification. This way, machine learning algorithms will be more flexible and desirable while handling large-scale and noisy unstructured data, such as speech, images, videos, time series, and texts.

Deep learning [22] is a typical approach for representation learning, which has recently achieved great success in speech recognition, computer vision (CV), and natural language processing (NLP). Deep learning has two distinguishing features:

Distributed Representation Deep learning algorithms represent each object with a low-dimensional and real-valued dense vector. The representation form is usually named as *distributed representation* or *embedding*. Compared to conventional symbolic representation, distributed representation is more compact and smooth by mapping data in the low-dimensional and continuous space, as shown in Fig. 1.1. Hence, it is more robust to address the sparsity issue that is ubiquitous and inevitable due to the power-law distribution in large-scale data.

Deep Architecture Deep learning algorithms usually learn a *deep hierarchical architecture* to represent objects, known as multilayer neural networks. Deep architecture may capture informative features and complicated patterns of objects from raw data. Take the sentence "you are a night owl." For example, as illustrated

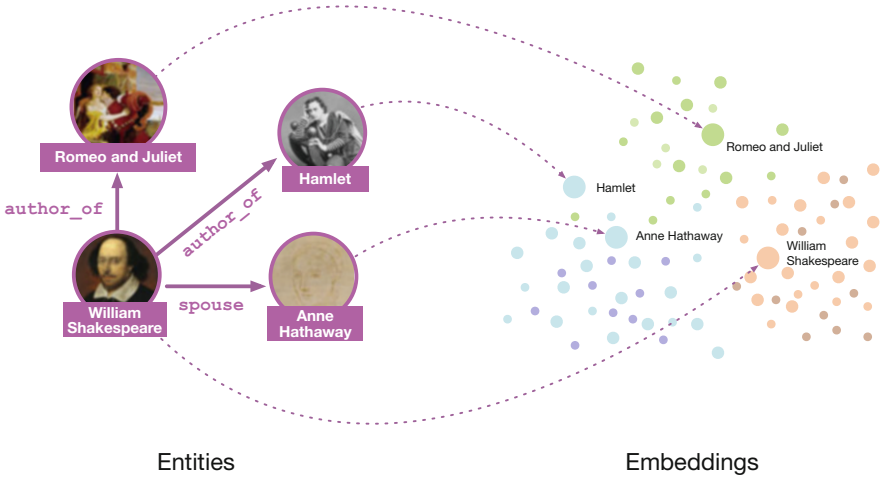


Fig. 1.1 Distributed representation of words and entities in human languages. (The images are obtained from wikimedia.org.)

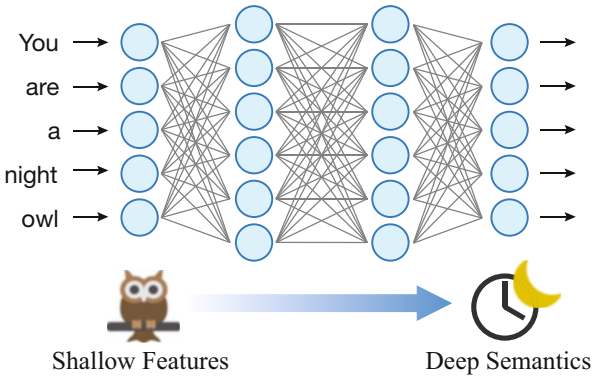


Fig. 1.2 Deep architecture enables representation learning to capture informative features and complicated patterns of human languages. Icons in this figure are bought or freely downloaded from IconFinder (https://www.iconfinder.com/)

in Fig. 1.2, the deep architecture of neural networks will be able to understand the deep semantics of the sentence, indicating a person stays up late using a metaphor, beyond the surface and shallow meanings. Hence, it is regarded as an important reason for the great success of deep learning for speech recognition, CV, and NLP.

The success of deep learning happens in speech recognition and CV first in around the 2010s, and in the following years, NLP also achieves significant improvements by following the deep learning approach. At the beginning of the revolution, deep learning for NLP significantly reduced feature engineering in NLP. In recent years, with the development of pre-trained language model techniques [17] in deep learning, the performance of almost all NLP tasks has achieved consistent and groundbreaking improvements. Hence, a growing number of researchers have devoted to developing effective deep learning methods for NLP.As per standard style, a footnote is not in the figure caption. So Footnote 1 has been moved to the corresponding citation, and the remaining footnotes are renumbered accordingly. Please check if okay.

In this chapter, we will first discuss why representation learning is essential for NLP and briefly review the development history and intellectual origins of representation learning for NLP. After that, we will introduce typical approaches of contemporary representation learning and summarize existing and potential applications of representation learning. Finally, we will introduce the general organization of this book.

1.2 Why Representation Learning Is Important for NLP

NLP aims to build linguistic-specific programs for machines to understand and use languages. Natural language texts are typically unstructured data with multiple granularities used in multiple domains. A deep understanding of natural languages also requires considerable human knowledge. There are multiple NLP tasks addressing different goals of NLP. These characteristics make NLP challenging to achieve satisfactory performance.

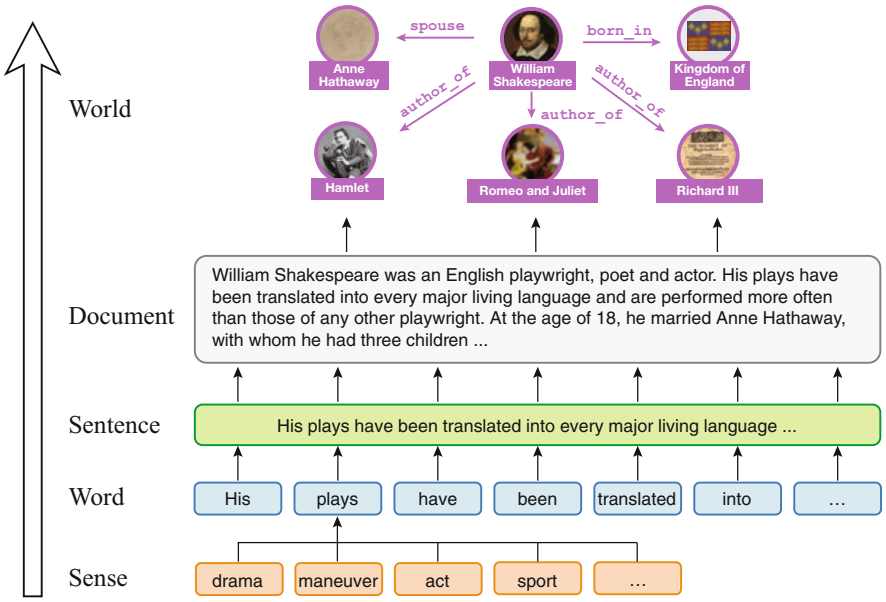


Fig. 1.3 There are multiple-grained language items, including characters, senses, words, phrases, sentences, paragraphs, documents, World Wide Web, as well as external world knowledge, with complicated compositional semantics. This figure shows some of them composed with each other. (The images are obtained from wikimedia.org.)

1.2.1 Multiple Granularities

NLP is concerned about multiple levels of language items, including but not limited to characters, senses, words, phrases, sentences, paragraphs, and documents. As shown in Fig. 1.3, each word may be composed of multiple senses, a sentence is composed of multiple words, and a document is composed of multiple sentences. The World Wide Web is even composed of billions of documents linked to each other, which is not shown in the figure. Moreover, human languages connect with the physical world, which may be described as world knowledge.

These language items are composed of each other following complicated patterns of compositional semantics. NLP is about understanding and processing these language items, and the key challenge is to model the complicated composition patterns. Representation learning is able to represent the semantic meanings of all language items in a unified semantic space. This significantly contributes to model complex semantic relations among these language items.

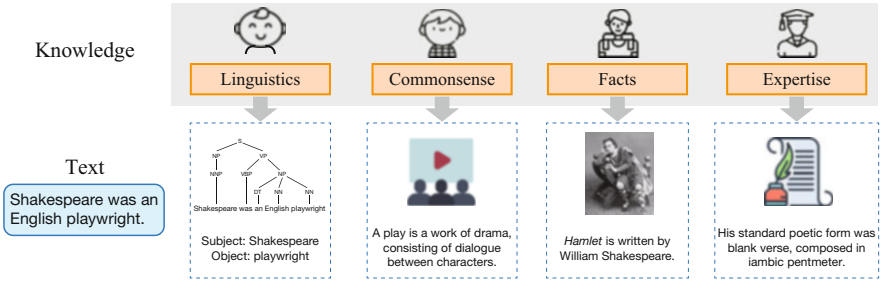


Fig. 1.4 Deep understanding of natural languages requires the support of multiple external knowledge such as linguistic knowledge, commonsense knowledge, world knowledge, and domain knowledge. Icons in the figure are bought or freely downloaded from IconFinder

1.2.2 Multiple Knowledge

A deep understanding of natural languages requires external human knowledge such as linguistic, commonsense, world, cognitive, and domain knowledge. The types of knowledge will be introduced in detail in Chap. 9. People and machines with different knowledge will have different-level understandings of the text.

As shown in Fig. 1.4, let's take the sentence "Shakespeare was an English playwright," for example. With the support of linguistic knowledge, we can capture the subject, and the object from the sentence by parsing the syntactic structure. With the commonsense knowledge of *A play is a work of drama, consisting of dialogue between characters*, we know most of Shakespeare's plays consist of character dialogues. If some persons also have some factual knowledge about Shakespeare, such as *Hamlet is written by William Shakespeare*, we can infer that *Hamlet* is an English play. A person with expert knowledge of literature may further think about the poetic form of Shakespeare.

Knowledge should be provided as much as possible to make machines more intelligent. For this goal, people have built many knowledge bases of multiple types and organized them in different structured forms. However, it is difficult for symbolic text and knowledge to work together due to their diverse representation forms, which are usually remedied by additional engineering efforts such as entity linking and suffered from error propagation. Representation learning, in contrast, can easily incorporate multiple types of structured knowledge into NLP systems by encoding both sequential text and structured knowledge into unified embedding forms.

1.2.3 Multiple Tasks

Many NLP tasks have been proposed and studied based on the same input to meet the needs of different scenarios, aspects, and levels. Take the sentence in Fig. 1.5 for example. We can perform multiple tasks on the same sentence as follows:

- **Part-of-speech (POS) tagging** aims to classify each word in a text into corresponding part-of-speech types, such as nouns, verbs, adjectives, adverbs, and prepositions, based on its context. In this figure, we show the annotated part-of-speech tags (e.g., `NNP`, `VBD`) following *Penn Part of Speech Tags* [34].
- **Dependency parsing** is a language grammar to build syntactic relations between language items in a sentence. Here we show the binary dependencies of language items and the dependency types. It can identify complicated syntactic relations inside a sentence, which is important in statistical NLP.
- **Named entity recognition** aims to find named entities mentioned in the text with pre-defined classes such as person names, organizations, locations, and time expressions.
- **Entity linking** further links named entity mentions to corresponding entities in external knowledge graphs by resolving those entities with the same names. The task is important for grounding human language understanding with the real world.
- **Relation extraction** aims to find relations between two entities expressed by the sentence. Relation extraction is a core task of information extraction to acquire structured knowledge from unstructured text and complete large-scale knowledge graphs.
- **Question answering** is to read a text and find answers for a given question. The task is important for the service of user information acquisition beyond search engines.
- **Machine translation** automatically translates the sentence from one language to another language. Machine translation is a long-standing NLP task to break the language barrier among people all over the world.

Here we only show several NLP tasks, and there are many more tasks concerning different goals and specific languages. For example, since there are no natural space marks between words in the text of Chinese and Japanese, automatic word segmentation has been proposed for these languages.

It is evident that all NLP tasks rely on accurate understanding and representation of given text input. In this case, building a unified and learnable representation of an input for multiple tasks will be more efficient and robust: on the one hand, a better and unified text representation will help to promote all NLP tasks, and on the other hand, taking advantage of more learning signals from multitask learning may contribute to building better semantic representations of natural languages. Hence, representation learning can benefit from multitask learning and further promote the performance of multiple tasks.

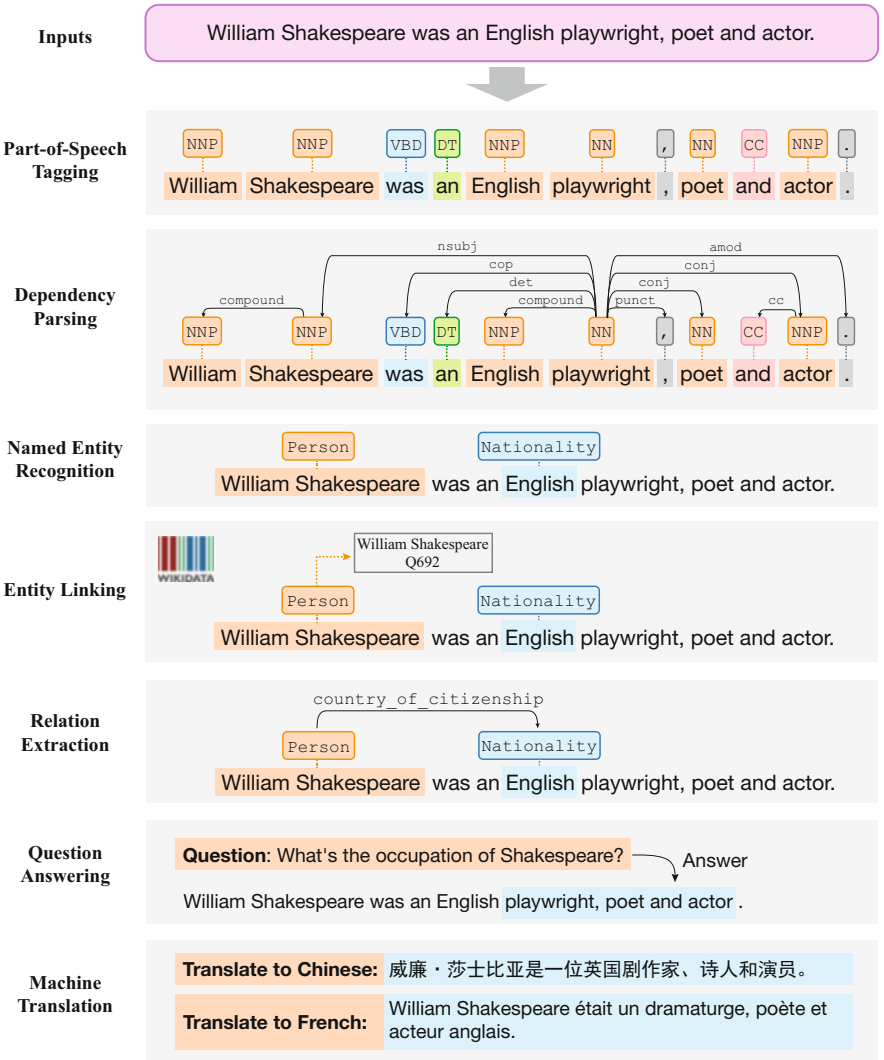


Fig. 1.5 There are various NLP tasks given the same sentence input, such as part-of-speech tagging, dependency parsing, named entity recognition, entity linking, relation extraction, question answering, and machine translation

1.2.4 Multiple Domains

Natural language texts may be generated from multiple domains, including but not limited to news articles, scientific articles, literary works, and online user-generated content such as product reviews. Moreover, we can also regard texts in different languages as multiple domains. Conventional NLP systems must design specific

feature extraction algorithms for each domain according to its characteristics. In contrast, representation learning can take advantages of large-scale domain-specific data and can also transfer representation knowledge across multiple domains, especially from a much larger general domain to those specific domains.

1.3 Development of Representation Learning for NLP

We give a brief introduction to the development history of representation learning for NLP, from which we can see the paradigm shift of representation from symbolic representation to distributed representation, accompanied by the paradigm shift of machine learning from statistical learning to deep learning and further to pre-trained models. The development timeline is also shown in Fig. 1.6.

1.3.1 Symbolic Representation and Statistical Learning

Words would be a good start for studying representation schemes in NLP, because words are the minimum units in natural languages. The easiest way to represent a word in a computer-readable way (e.g., using a vector) is **one-hot vector**, which has the dimension of the vocabulary size and assigns 1 to the corresponding index of the word to be represented and 0 to others. It is apparent that one-hot vectors hardly contain semantic information about words other than distinguishing them from each other.

The idea of one-hot word representation can be further used for document representation, i.e., **bag-of-words (BOW) models** [18]. BOW models regard a document as a bag of its words, neglecting the orders of these words in this document. BOW represents a document as a vocabulary-size vector, with each word in the document corresponding to a nonzero dimension and other words to a zero dimension. The entry value of a word can be used to indicate the importance score of this word in the document, e.g., the number of its occurrences. BOW can be regarded as a combination of one-hot representations of all words in the document. BOW models are straightforward and work great in applications like spam filtering, text classification and clustering, and information retrieval. For example, in information retrieval, we build BOW vectors of a query and a document and compute the cosine distances as the semantic similarity for document ranking. Those documents that also attach importance to the important words in the query will be ranked higher. It proves that the distributions of words can serve as a good representation of documents.

One of the earliest ideas of word representation learning can date back to **n-gram models** [35]. It is easy to understand: when we want to predict the next word in a sentence, we usually look back at some previous words (and in the case of n-gram, they are the previous $n - 1$ words). And if going through a large-scale corpus, we can count and estimate a reasonable probability of a word under the

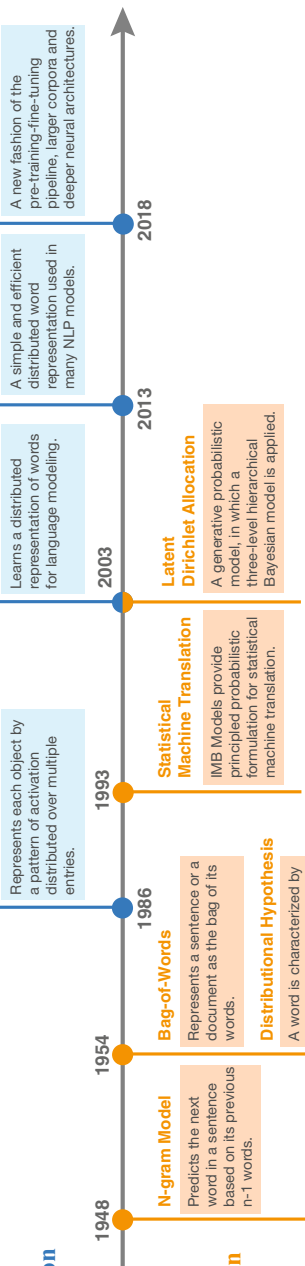


Fig. 1.6 The timeline for the development of representation learning in NLP. With the growing computing power and large-scale text data, distributed representation trained with neural networks from large corpora has become the mainstream

condition of all combinations of $n-1$ previous words. These probabilities can predict word sequences and form vector representations for word meanings because similar words usually share similar probabilistic distributions of previous words.

The idea of n-gram models is coherent with the idea of **distributional hypothesis**: language items sharing similar distributions of context have similar meanings [18]. In another phrase, "a word is characterized by the company it keeps" [14]. The distributional hypothesis has been a fundamental idea of many NLP models, from n-gram in statistical NLP to word2vec and BERT in neural NLP.

In the above cases of one-hot word representation, BOW document representation, and n-gram models, each entry in the representation explicitly matches one language item (e.g., word scores in BOW models). This one-to-one correspondence between representation entries and language items is called **local representation** or **symbolic representation**.

The idea of symbolic representation is natural and straightforward, and most NLP algorithms in the stage of statistical learning in the 1980s–2000s are based on symbolic representation. Here we give another two iconic examples, IBM Model and latent Dirichlet allocation. IBM model [7] is a classical word alignment algorithm in statistical machine translation. It automatically builds lexical translation probabilities between the words of two languages from their parallel sentences, where words are regarded as symbolic items without considering their internal semantic representation. Latent Dirichlet allocation (LDA) [4] is a classical word and document representation algorithm. LDA builds latent topics to represent words and documents. These learned topics are typically interpretable, even capable of being labeled with symbolic names [29]. By regarding each latent topic as a meaningful symbol, we can also regard LDA as an example of symbolic representation, especially when learning with Gibbs Sampling [16] by iteratively assigning latent topics for each word in documents.

1.3.2 Distributed Representation and Deep Learning

Distributed representation, on the other hand, represents an object by a pattern of activation distributed over multiple entries, i.e., a low-dimensional and real-valued dense vector, and each computing entry can be involved in representing multiple objects [27]. Distributed representation has been proved to be more efficient because it usually has low dimensions. It also prevents from the sparsity issue that is inevitable for the symbolic representation due to the power-law distribution in large-scale data. Beneficial hidden properties can be learned from large-scale data and emerge in distributed representation.

Word embeddings can also learn complicated word relations automatically from large-scale corpora. As revealed by word2vec, we can identify the analogical properties of words such as

$$\mathbf{w}(\texttt{king}) - \mathbf{w}(\texttt{man}) \approx \mathbf{w}(\texttt{queen}) - \mathbf{w}(\texttt{woman}), \tag{1.1}$$

or

$$\mathbf{w}(\texttt{king}) - \mathbf{w}(\texttt{man}) + \mathbf{w}(\texttt{woman}) \approx \mathbf{w}(\texttt{queen}). \tag{1.2}$$

It indicates the embeddings of both `king` and `queen` accurately encode similar semantic meanings with each other except for gender. The example shows the powerful capabilities of word embeddings for semantic representations.

The idea of distributed representation was initially inspired by the neural computing scheme of humans and other animals [20]. Brains can use various activation patterns of neurons to represent different objects. In distributed representation, the values of an entry in the low-dimensional vector can be regarded as the activation state of the specific neuron. It is named *distributed* because an object is represented as the activation pattern distributed over multiple neurons, and the activation state of one specific neuron does not mean anything.

With the great success of deep learning, distributed representation has become the most commonly used approach for representation learning. One of the pioneering practices of distributed representation in NLP is **neural probabilistic language model (NPLM)** [3]. A language model predicts the conditional probability of the next word given those previous words in a sentence. n-gram models can be regarded as simple language models based on symbolic representation. NPLM assigns a low-dimensional vector for each word (i.e., **word embedding**) and then uses a neural network to predict the next word based on distributed representations of previous words (i.e., context embedding, a combination of embeddings of previous words). By going through the training corpora, NPLM successfully learns word embeddings as model parameters to optimize the conditional probability of the next word or the joint probability of a sentence. Although it is hard to tell what each entry of a word embedding means, the vectors indeed encode semantic meanings about the words, verified by the performance of NPLM.

Inspired by NPLM, many methods have been proposed to learn word embeddings as model parameters optimized with language modeling objective, such as **word2vec** [30], **GloVe** [31], and **fastText** [6]. Although different in model and algorithm details, these methods are all very efficient for learning from large-scale corpora and have been widely used as word embeddings in many NLP models. Word embeddings can map discrete words into low-dimensional vectors as informative features in the NLP pipeline and help to shine a light on neural networks in computing languages. It makes representation learning a critical part of NLP.

1.3.3 Going Deeper and Larger with Pre-training on Big Data

The research on representation learning in NLP further takes a great leap by **ELMo** [32] and **BERT** [11]. These models apply larger corpora, more parameters, and more computing resources to build deeper and larger models. Moreover, they consider the complicated context of the text to learn richer knowledge of human

languages. Instead of mapping a word to a fixed vector, ELMo and BERT use multilayer neural networks to build dynamic contextualized representations of each word based on its specific context in text, which is especially useful for those words with multiple meanings. Moreover, BERT starts a new fashion (although not originated from it) of the **pre-training-fine-tuning** pipeline. As shown in Fig. 1.7, previous word embeddings learned from large corpora were adopted as initialization of input representations of neural networks for downstream tasks; starting from BERT, it becomes a common practice to take the whole neural network structure such as BERT and all parameters pre-trained on large text corpora to downstream tasks, with those parameters further fine-tuned on supervised data of downstream tasks.

The models like BERT are pre-trained through language modeling objectives on large corpora, thus named as **pre-trained language models (PLM)**. PLMs take advantage of large-scale text corpora and have achieved state-of-the-art on almost all NLP benchmarks. Hence, although not a big theoretical breakthrough, PLMs have attracted wide attention in the NLP and machine learning community. Of course, PLMs reveal many distinct characteristics compared to conventional deep learning methods, such as parameter-efficient tuning capabilities [12] and in-context few-shot learning capabilities [8]. Some experiments of knowledge probing demonstrate that PLMs implicitly encode a variety of linguistic and world knowledge and patterns inside the multilayer neural network parameters [19, 24]. All these notable performances and interesting analyses suggest that there are a lot of open problems to explore in PLMs, as the future of representation learning for NLP.

In summary, representation learning for NLP has evolved from symbolic to distributed representation following the distributional hypothesis. Starting from word2vec, word embeddings learned from large corpora have shown outstanding performance in many NLP tasks. Recent PLMs learn complicated contextualized representations from large-scale text corpora and start the new paradigm of the pre-training-fine-tuning pipeline. Representation learning has revolutionized NLP in the past decades. What will be the next big breakthrough in representation learning for NLP? We hope this book can give some inspiration by introducing the evolutionary paths and most recent advances of representation learning for NLP.

1.4 Intellectual Origins of Distributed Representation

For this vital revolution in artificial intelligence (AI), one may be interested in the intellectual origins of the essential idea of distributed representation. To our knowledge, the exact term "distributed representation" was first proposed in parallel distributed processing (PDP) [27]. Still, the idea of distributed representation may have its prototypes in different areas. Here we try to find some clues between distributed representation and related areas, including neuroscience, AI, machine learning, and linguistics.

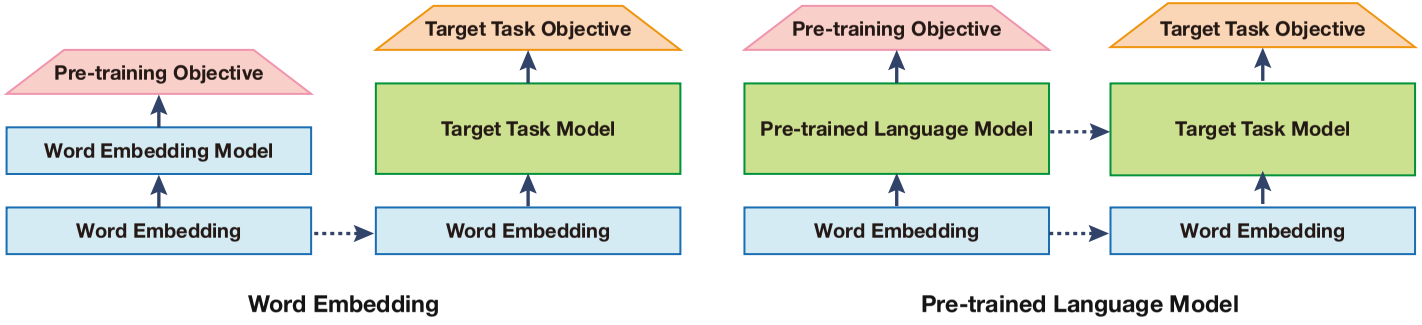


Fig. 1.7 This figure shows how word embeddings and pre-trained language models work in NLP pipelines. They both learn distributed representations for language items (e.g., words) through pre-training objectives and transfer them to target tasks. Furthermore, pre-trained language models can also transfer model parameters

1.4.1 Representation Debates in Cognitive Neuroscience

The most direct intellectual origin of distributed representation is **cognitive neuroscience**. A central topic in cognitive neuroscience is how information and knowledge are represented in human brains, and a long-standing debate is whether representation is *localized* or *distributed*. In the history of cognitive neuroscience, researchers used the terms *local* and *distributed* in different ways. For example, they may be used to describe the views that particular knowledge is either stored in specific brain regions or spread across the entire cortex [15].

Here we focus on the most well-known version from the classical book of parallel distributed processing (PDP) [27]. In this book, they raise two opposite representation schemes where the definition of local representation is

> Given a network of simple computing elements and some entities to be represented, the most straightforward scheme is to use one computing element for each Entity. This is called a *local* representation. (from [27])

and the definition of distributed representation is

> Each Entity is represented by a pattern of activity distributed over many computing elements, and each computing element is involved in representing many different entities. (from [27])

We can build a metaphor between one-hot representation in NLP and local representation in neuroscience, by regarding each entry in the vocabulary-size vector as a neuron in human brains, with the value 1 indicating active and the value 0 inactive. The view of local representation was also referred to as the famous *grandmother cell hypothesis*, which assumes a hypothetical neuron can encode and respond to a specific and complicated entity such as someone's grandmother [2]. From the view of one-hot representation, there will be an entry corresponding to someone's grandmother.

Local representation is straightforward because it is a simple mirror of the knowledge structure, with each object and concept corresponding to distinct neurons. Based on local representation, high-level knowledge can be organized into symbolic systems, such as concept hierarchy, propositional networks, semantic networks, schemas, frames, and scripts [1].

The above metaphor also works on distributed representation between NLP and neuroscience by regarding each vector entry as a neuron. The entry values indicate the status of neurons, active or inactive. The distributed representation scheme is not so straightforward, but it is already widely accepted that visual signals may activate millions of neurons throughout many regions of the visual cortex.

Although it is debatable whether individual neurons encode high-level concepts or objects, distributed representation seems to be a general solution for information processing at different levels, ranging from visual stimulus to high-level concepts. As discussed in PDP [27], distributed representation has better representation capacity, automatic generalization to novel entities, and learnability to changing environments. All these characteristics are valuable to our modern world with rich data for machine learning and have been verified in the recent revolution of deep learning.

1.4.2 Knowledge Representation in AI

An essential branch of philosophy is the theory of knowledge, also known as epistemology. Epistemology studies the nature, origin, and organization of human knowledge and concerns the problems like where knowledge is from and how knowledge is organized.

In philosophy, for the problem of **how knowledge is organized**, philosophers have developed many tools and methods, typically in the *symbolic* form, to describe human knowledge, and some of them, such as formal logic, have played an essential role in computer science. For the problem of **where knowledge is from**, there are two basic views: *rationalism* regards reason as the chief source of human knowledge and regards intellectual and deductive as truth criteria; *empiricism* generally appreciates the role of sensory experience in the generation of knowledge.

By building a metaphor between AI and humans, knowledge representation can be regarded as the epistemology for machines in AI. AI is also concerned about the above two problems, and many works have been done following the two lines.

For the problem of **how knowledge is organized** in AI, we can conclude two main approaches, *symbolism* and *connectionism*.

Symbolism aims to develop formal symbolic systems to organize knowledge of machine intelligence. Most pioneering works in AI follow the approach of symbolism, ranging from general problem solvers by Newell and Simon in 1959 to expert systems and knowledge bases by Ed Feigenbaum in the 1980s. Hence, some news articles on AI may imply symbolism as an obsolete approach, named old-fashioned AI (OFAI). With the rise of the Internet, WWW, and big data, remarkable works such as semantic Web by Tim Berners-Lee in the 2000s and recent large-scale Knowledge Graphs by Google in the 2010s can also be regarded as following the symbolism approach for knowledge representation.

Connectionism is the approach inspired by cognitive neuroscience, rooted in the works such as the perceptron by Frank Rosenblatt in the 1950s and parallel distributed processing (PDP) [27] in the 1980s. We usually regard deep learning in the 2010s as the great success of this approach, which has been almost 40 years since distributed representation was first proposed in the 1980s.

For the problem of **where knowledge is from** in AI, the approaches can be divided into *rationalism* and *empiricism*. The rationalism approach indicates that knowledge, including facts and rules, is directly designed or collected by human experts, i.e., the creators of AI agents; AI agents can complete tasks based on the given knowledge. The expert systems in the 1980s typically follow this approach. It is obvious that the manually organized knowledge is not flexible and dynamic, cannot generalize well to novel cases, and cannot evolve as the environment changes. With the development of the Internet and big data, the empiricism approach becomes feasible to learn from large-scale data, including unlabeled general data and labeled task-specific data. Statistical learning starting from the end of the 1980s and deep learning starting from the 2010s both follow the empiricism approach.

Epistemology of philosophy only developed formal and symbolic tools extensively. But for computational epistemology in AI, the approach of knowledge source and the approach of knowledge form have been studied in mixed ways in different periods of AI history: in the preliminary stage of the 1950s–1980s, most works followed a mix of symbolism and rationalism, also named as old-fashioned AI (OFAI); in statistical learning 1980s–2000s, the mainstream is a mix of symbolism and empiricism; and in deep learning from 2010s, it becomes the mix of connectionism and empiricism.

Note that, although *rationalism* and *empiricism* represent two distinct approaches for where knowledge is from, it does not mean they don't or cannot work together. On the one hand, machine learning models typically learn from data following the empiricism approach. On the other hand, the design of model architectures and algorithms involves the wisdom of human experts following the rationalism approach. It is the same to *symbolism* and *connectionism*. McCulloch and Pitts designed a computational scheme of symbolic logics using elementary units of neural systems in 1943 [28]; in the era of deep learning, neural-symbolic networks are also an active research area for reasoning and planning with neural networks [38].

It seems not feasible to explicitly mix rationalism and connectionism. However, as Noam Chomsky and many researchers indicated, human brains are not blank slates [9]. We can also regard the architectural design of neural networks as a kind of prior knowledge following the theory of rationalism. For humans and AI, we should not set up barriers between symbolism and connectionism or between rationalism and empiricism. All of them may play some roles in human and machine intelligence. *It doesn't matter whether a cat is black or white, as long as it catches mice.*

As we will show in this book, deep learning with distributed representation can manipulate symbols as well as other discrete information, such as instructions, operations, and codes (Chap. 5). We can also modify the architecture of neural networks given prior knowledge to fit downstream tasks better (Chap. 9). Hence, we believe distributed representation is a good foundation to take advantage of all promising approaches of knowledge sources and forms, with many open and interesting problems deserving further exploration.

1.4.3 Feature Engineering in Machine Learning

Feature engineering is a critical step in the pipeline of statistical learning, aiming to build feature vectors of instances for machine learning. It can be regarded as the representation learning of instances in the era of statistical learning. Feature engineering provides another intellectual origin of distributed representation, i.e., dimensionality reduction of raw data by mapping from a high-dimensional space into a low-dimensional space, usually with the term "embedding."

Feature engineering can be divided into feature selection and feature extraction. Feature selection techniques select the most informative features and remove redundant and irrelevant ones from large amounts of candidates to represent instances such as words and documents. This approach is expected to improve representation efficiency and remedy the curse of dimensionality. Many methods of feature selection and term weighting have been explored on specific NLP tasks such as text classification [36]. Since candidates for feature selection are usually symbols such as words, phrases, and n-grams, the selected feature vocabulary is also a symbolic representation.

Feature extraction aims to build a novel feature space from raw data, with each dimension of the feature space either interpretable or not. Latent topic models and dimensionality reduction can be regarded as representative approaches for feature extraction. Latent topic models represent each document and word as a distribution over latent topics, which can be regarded as interpretable space. Examples are probabilistic latent semantic analysis (pLSA) [21] and latent Dirichlet allocation (LDA) [4]. Dimensionality reduction methods learn to map objects into a low-dimensional and uninterpretable space. Examples are principal component analysis (PCA) and matrix factorization methods like singular value decomposition (SVD).

Note that the term **embedding** in machine learning refers to either the projection process (such as the algorithm locally linear embedding [33]) or the corresponding low-dimensional representation of objects. We can see that, without the metaphor between human brains and AI in deep learning and distributed representation, the idea of representing objects in a low-dimensional space has already been widely used in statistical learning. The representation scheme is the same between low-dimensional embedding and distributed representation in deep learning. Some neural networks, such as Autoencoder, used to be regarded as a dimensionality reduction method of data.

Hence, in recent years of deep learning and AI, the term *distributed representation* and *embedding* are used mutually to refer to each other. The difference is that the model architecture of most dimensionality reduction methods in statistical learning is usually shallow and straightforward and the algorithm is also restricted to specific data forms such as matrix decomposition. Those tasks that can easily organize data as matrix benefit much from these methods, such as recommender systems focusing on user-item interactions [23]. In contrast, the model architecture of deep learning is typically deep with multiple neural layers, capable of modeling complicated interactions and capturing sophisticated semantic compositions ubiquitous in human languages.

1.4.4 Linguistics

Human languages are regarded as the epitome of human intelligence, and linguistics aims to study the nature of human languages. Since human languages are regarded as one of the most complicated symbolic systems, linguistics typically follows the symbolism approach.

An influential theory of linguistics is *structuralism* derived from the founder of modern linguistics, Ferdinand de Saussure. Saussure proposed the following perspectives [10]: (1) a *symbol* (or sign) in human languages is composed of the *signified* (i.e., a concept in mind) and the *signifier* (i.e., a word token, or its sound or image). (2) For a symbol, "the bond between the signified and the signifier is arbitrary" [10]. For example, there is no *intrinsic* relationship between the concept of "sister" and the sound of the word "sister"; for another example, the words in different languages may refer to the same concept. (3) Hence, a symbol can only get its meaning from its relationship with other symbols. For example, the meaning of the word "parent" is related to the meaning of the corresponding word "child." In summary, the structuralism theory regards human languages as a symbolic system where each item is defined by its relationship to other items in the system [26].

The idea of distributed representation coincides in spirit with structuralism. By distributed representation learning, we can see that all language items we are interested in are projected into a unified low-dimensional semantic space. As demonstrated in Fig. 1.1, the geometric distance between two language items in the semantic space indicates their semantic relatedness; the semantic meaning of an item corresponds to its geometric relationships with other items, such as above-mentioned $\mathbf{w}(\texttt{queen}) \approx \mathbf{w}(\texttt{king}) - \mathbf{w}(\texttt{man}) + \mathbf{w}(\texttt{woman})$. In other words, the relative closeness with other items rather than its absolute position in the semantic space reveals an item's meaning.

Later, the structuralism theory evolved into a more computational version, *distributionalism*, arguing that the meanings of linguistic items are defined by their distribution in text corpora. The distributionalism is further developed into the *distributional hypothesis* formalized by American linguist Zellig Harris, arguing that language items sharing similar distributions of context have similar meanings [18]. The distributional hypothesis provides a computational way of following the empiricism approach to learning semantic representations of text from large-scale corpora, which is essential to distributed representation learning.

1.5 Representation Learning Approaches in NLP

In the history of AI, researchers have developed various effective and efficient approaches to learning semantic representations for NLP. Here we list some typical approaches.

1.5.1 Feature Engineering

As introduced above, semantic representations for NLP in the early stage often come from statistics instead of learning with optimization. Feature engineering is a typical

approach to representation learning in statistical learning and can be divided into feature selection and feature extraction.

During the era of statistical learning, feature selection techniques have been extensively explored in NLP, focusing on selecting the most informative symbolic features because of the symbolic nature of human languages. For feature engineering of NLP, researchers should take care of issues such as feature set construction, feature weighting, and smoothing.

For statistical learning of various NLP tasks, we should determine what features should be considered. All syntactic and semantic features of language items, such as words and their part-of-speech (POS) tags, n-grams, word and entity types, semantic roles, and parse trees, may be helpful in specific NLP tasks. These linguistic features may be extracted by specific NLP systems or provided by given tasks. Even for the features of language items, how to select those most informative ones to form the feature set is also an important issue [36].

After the feature set is determined, measuring the feature weight for a specific instance is also essential. For example, in n-gram or bag-of-words models, entries in the representation are usually frequencies, occurrence numbers, or other weight scores of the corresponding language items counted in a given text or large-scale corpora. These feature scores indicate essential semantic characteristics of the given instance.

Moreover, the dimension of the feature space in statistical NLP is usually substantial, and the feature vector of a word or document correspondingly exhibits the sparsity issue, i.e., the curse of dimensionality in the context of NLP. To address the sparsity issue, besides dimensionality reduction techniques, researchers also developed smoothing techniques of semantic representation [37] by taking advantage of more context of the given document, such as its related documents.

In summary, in a long period before the era of distributed representation, researchers devoted lots of effort to manually designing, selecting, and weighing useful linguistic features and incorporating them as inputs of NLP models. The feature engineering pipeline heavily relies on human experts of specific tasks and domains, is thus time-consuming and labor-intensive, and cannot generalize well across objects, tasks, and domains.

1.5.2 Supervised Representation Learning

Distributed representations can emerge from the optimization of neural networks under supervised learning. In hidden layers of neural networks, the different activation patterns of neurons represent different objects. With a training objective (usually a loss function for the target task) and supervised signals (usually the gold-standard labels for training instances of the target task), the networks can learn to find better parameters for representing language items via optimization such as gradient descent. With proper training, the hidden states will become informative and generalized as good semantic representations of natural languages.

For example, to train a neural network for sentiment classification, the loss function is usually formalized as the cross entropy of model predictions considering gold-standard sentiment labels as supervision. By optimizing the objective with many supervised training instances, in company with the training loss getting smaller and the classification performance getting better, the model is expected to build better sentence representations as classification features.

1.5.3 Self-supervised Representation Learning

In many cases, we do not have human-labeled data for supervised learning. We need to find "labels" intrinsically from large-scale unlabeled data to acquire the training objective necessary for neural networks. The approach can be regarded as a mixed way between supervised and unsupervised learning, called self-supervised learning.

Language modeling is a typical self-supervised objective because it does not require human annotations. For the learning objective of predicting the next word given previous context words, we can effortlessly obtain the gold standard of the next words from large-scale corpora.

Another example of self-supervised representation learning is Autoencoder. An Autoencoder has a reduction (encoding) phase and a reconstruction (decoding) phase. The model will encode an object into a low-dimensional representation in the reduction phase and reconstruct the object from the intermediate representation in the reconstruction phase. Here the training objective is the reconstruction loss, taking the original data as the gold standard. During the training process, meaningful information will be encoded and kept in latent representations, and noisy or useless signals will be discarded.

The most advanced **pre-trained language models** combines the advantages of both self-supervised learning and supervised learning. In the **pre-training-fine-tuning** pipeline, pre-training can be regarded as self-supervised learning from large-scale unlabeled corpora, and fine-tuning is supervised learning with labeled task-specific data. Self-supervised learning has dramatically succeeded in NLP because the plain text contains abundant knowledge and patterns about languages. Self-supervised learning can effectively learn from almost infinite large-scale text corpora. Nowadays, it is still one of the most exciting research areas of representation learning for NLP, and a growing number of researchers are devoting their efforts to learning better pre-trained language models.

Besides, many other machine learning approaches have also been explored in representation learning for NLP, such as adversarial training, contrastive learning, few-shot learning, meta-learning, continual learning, and reinforcement learning. It is still an active research topic on developing effective and efficient representation learning methods for NLP from large-scale and complicated corpora and computing power.

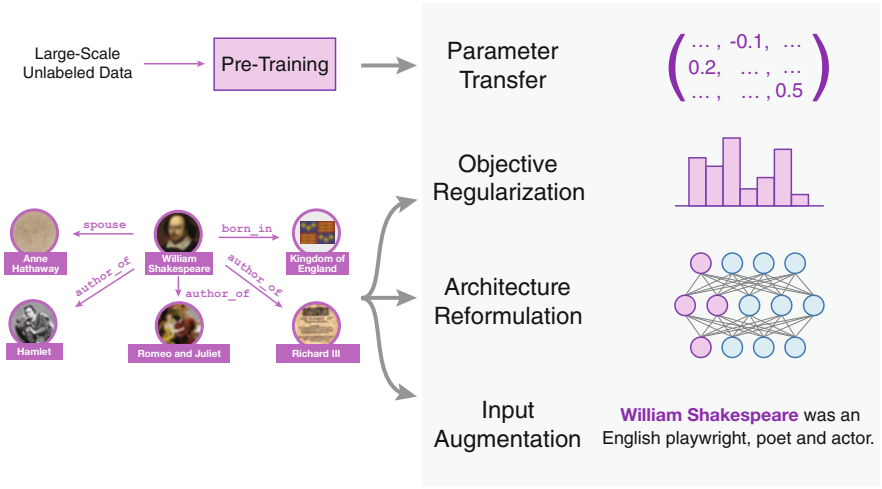


Fig. 1.8 The approaches of applying representation learning in NLP systems, including input augmentation, architecture reformulation, objective regularization, and parameter transfer. (The images are obtained from wikimedia.org.)

1.6 How to Apply Representation Learning to NLP

We summarize four typical approaches to applying representation learning of multiple objects to promote NLP systems, including input augmentation, architecture reformulation, objective regularization, and parameter transfer.

As shown in Fig. 1.8, a typical scenario is, if we have some structured knowledge, representation learning can help to incorporate the knowledge into various components of an NLP system, such as input, architecture, and objective. It also works with unstructured knowledge, such as word representations from large-scale text corpora. Moreover, the pre-training-fine-tuning pipeline in pre-trained language models offers parameter transfer to an NLP system.

1.6.1 Input Augmentation

The basic idea of input augmentation is to learn semantic representations of objects in advance. Then, object representations can be augmented as some parts of the input in downstream models. For example, word embeddings can be learned with language modeling from large-scale corpora and then used as input initialization for downstream NLP models. During the learning process of downstream models, we can either keep these word embeddings fixed and only tune other model parameters or tune all parameters considering word embeddings. There is no answer to which strategy is better. In practice, it should be determined by empirical comparison,

influenced by many factors, such as the amount of supervised downstream data and the complexity of downstream tasks.

We can also introduce external knowledge related to input to augment inputs of downstream models. In this book, we will introduce world knowledge (Chap. 9), linguistic and commonsense knowledge (Chap. 10), and domain knowledge (Chaps. 11 and 12), whose representations can be learned based on either knowledge graphs or symbolic rules and then integrated into specific NLP systems as input augmentation for improving performance.

1.6.2 Architecture Reformulation

We can use objects (such as entity knowledge) and their distributed representations to restructure the architecture of neural networks for downstream tasks. For example, as we introduce in Chap. 10, we take sememe as the minimum indivisible unit of semantic meanings in human languages [5] and build linguistic and commonsense knowledge graphs with sememe-sense-word hierarchy. With the help of sememe knowledge, we can reformulate the next-word prediction task in neural language modeling into a pipeline of first predicting sememes of the next word, then predicting related senses, and finally predicting the next word. In this way, we make neural language models more interpretable and robust.

1.6.3 Objective Regularization

We can also apply object representations to regularize downstream model learning. As mentioned above, there are usually multiple language items in NLP tasks. Since all these items are mapped into a unified semantic space using representation learning, we can formalize various learning objectives to regularize model learning. For example, suppose we train neural language models from large-scale corpora and learn entity representations from a world knowledge graph. We can add a new learning objective of entity linking as regularization by minimizing the loss of predicting a mentioned entity in a sentence to the corresponding entity in the knowledge graph. With the help of more informative signals for learning, NLP models are expected to achieve better performance.

1.6.4 Parameter Transfer

The semantic composition and representation capabilities of language items such as sentences and documents lie in the weights within neural networks. We can directly transfer these pre-trained model parameters to downstream tasks in an

end-to-end fashion. We have mentioned the approach of parameter transfer in the pre-training-fine-tuning pipeline of pre-trained language models. Most NLP tasks are at the levels of sentences and documents: the tasks like sentiment classification, natural language inference, machine translation, and relation extraction require sentence representation; the tasks like question answering and information retrieval require document representation. All these tasks can benefit from the capabilities of sentence or document representations from pre-trained language models on large-scale corpora. Moreover, many representation learning methods have been designed specifically for sentences and documents and benefit these NLP tasks, which will be introduced in the corresponding chapters of this book.

1.7 Advantages of Distributed Representation Learning

From the above brief introduction, we can summarize the following advantages of distributed representation learning for NLP.

Unified Representation Space As shown in Fig. 1.9, distributed representation can provide a unified representation scheme and space for natural languages. The unified scheme and space can facilitate knowledge transfer across multiple language items, multiple human knowledge, multiple NLP tasks, and multiple application domains, as discussed in Sect. 1.2, and significantly improve the effectiveness and robustness of NLP performance.

Learnable Representation The embeddings in distributed representation can be learned as a part of model parameters in supervised or self-supervised ways. It is the reason for the name "representation learning." Unlike previous feature-engineered

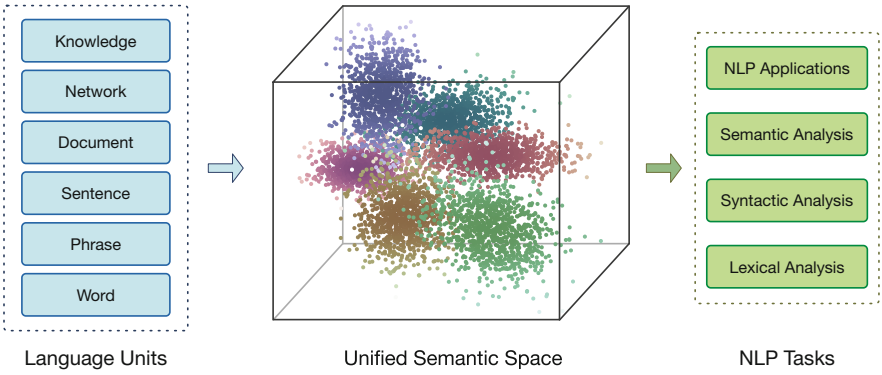


Fig. 1.9 Distributed representation can provide unified semantic space for multiple language items and for multiple NLP tasks

representation methods, this enables distributed representation adaptable to NLP tasks by learning from task-specific data.

End-to-End Learning Feature engineering in the symbolic representation scheme usually consists of multiple learning components, such as feature selection and term weighting. These components are conducted step-by-step in a pipeline, which cannot be well optimized according to the ultimate goal of the given task. In contrast, distributed representation learning supports end-to-end learning via back-propagation across neural network hierarchies.

1.8 The Organization of This Book

The book focuses on the distributed representation scheme (i.e., embedding) in NLP and talks about recent advances in representation learning methods for (1) multiple language items including words, sentences, and documents; (2) closely related topics including graphs, cross-modality, and robustness; and (3) external knowledge including world knowledge, linguistic and commonsense knowledge, and domain knowledge.

We start the book from word representation. By giving a thorough introduction to word representation in Chap. 2, readers are expected to learn basic ideas of representation learning for NLP. After that, we introduce the techniques of sentence and document representation learning in Chap. 4, focusing on compositionally acquiring semantic representations of a higher-level language item from its components. We further introduce the most advanced techniques, pre-trained language models, in Chap. 5. After going through these chapters, readers will establish essential knowledge about deep learning techniques in NLP and realize the key to the deep learning revolution in NLP is distributed representation learning.

There are three essential and closely related topics for representation learning in NLP. First, the graph is also a natural way to represent objects and their relationships. In Chap. 6, we introduce representation learning techniques for modeling nodes, edges, and graphs; how graph representation learning can help NLP. Second, another important topic related to NLP is cross-modal representation learning. It studies how to build unified semantic representations across distinct modalities, such as texts, audios, images, and videos. In Chap. 7, we focus on the interaction between vision and text to introduce techniques and advances in cross-modal representation learning. Third, the robustness of semantic representations is critical for NLP applications, which will be introduced in Chap. 8.

In this book, we also argue that a deep understanding of natural languages requires the support of multiple human knowledge. Representation learning can incorporate external knowledge for NLP, known as knowledge-guided NLP, as shown in Fig. 1.10. Here, we introduce three typical forms of knowledge representation closely related to NLP: entity-based world knowledge, sememe-based linguistic and commonsense knowledge, and legal and biomedical domain knowledge.

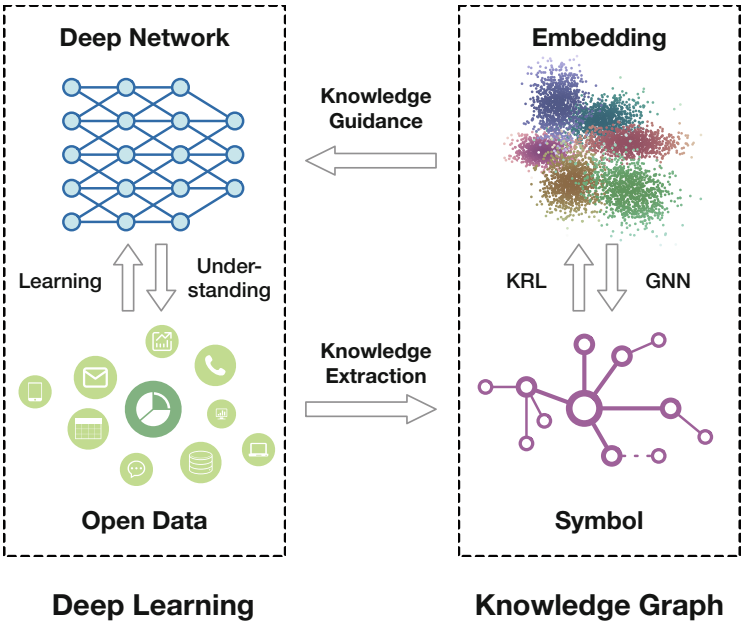


Fig. 1.10 The framework of knowledge representation learning (KRL), knowledge acquisition, and knowledge-guided NLP

In Chap. 9, we give a general introduction to knowledge representation learning (KRL) and take entity-based world knowledge as an example to introduce KRL methods and knowledge-guided NLP. World knowledge representation typically encodes world facts from knowledge graphs with entities and their relations into continuous semantic space. With world KRL, we can make NLP models knowledgeable of more information about those entities in text, such as rich attributes or relations with other entities.

Sememe representation encodes linguistic and commonsense knowledge of natural languages, where sememe is defined as the minimum indivisible unit of semantic meanings in human languages [5]. As shown in Chap. 10, with the help of sememe representation learning, we can get more interpretable and robust NLP models.

There is also rich and complicated domain knowledge along with large amounts of domain-specific texts. Domain knowledge is important for the accurate understanding of domain texts. In Chaps. 11 and 12, we take the legal and biomedical domains as examples to introduce how to represent domain knowledge of distinct forms and facilitate domain-specific NLP systems.

At the end of the book, we share some views about challenging topics in representation learning for NLP. We hope the outlook can inspire more readers to play a part in building more powerful representation learning for NLP and AI.

Acknowledgments We thank Tianyu Gao for preparing some initial draft materials for the first edition and thank Chaojun Xiao for drawing figures.

This is the introductory chapter of the second edition of the book *Representation Learning for Natural Language Processing*, with its first edition published in 2020 [25]. As compared to the first edition of this chapter, the main changes include the following: (1) we added the part Intellectual Origins of Distributed Representation, and (2) we comprehensively supplemented and updated the information, discussions, examples, and figures in other existing sections.

References

1. John Robert Anderson. *Cognitive psychology and its implications, seventh edition*. Worth Publishers, 2010.
2. H Barlow. Grandmother cells, symmetry, and invariance: how the term arose and what the facts suggest. *The cognitive neurosciences*, pages 309–320, 2009.
3. Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3, 2003.
4. David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
5. Leonard Bloomfield. A set of postulates for the science of language. *Language*, 2(3):153–164, 1926.
6. Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 2017.
7. Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.
8. Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Proceedings of NeurIPS*, 2020.
9. Noam Chomsky et al. *Language and mind*. Cambridge University Press, 2006.
10. Ferdinand De Saussure. *Course in general linguistics*. Columbia University Press, 2011.
11. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.
12. Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*, 2022.
13. Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.
14. John R Firth. A synopsis of linguistic theory, 1930-1955. *Studies in Linguistic Analysis*, 1957.
15. Tim van Gelder. *The MIT Encyclopedia of the cognitive sciences (MITECS)*, chapter Distributed vs. local representation, pages 236–238. MIT Press, 2001.
16. Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National academy of Sciences*, 101:5228–5235, 2004.
17. Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2021.
18. Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.
19. John Hewitt and Christopher D. Manning. A structural probe for finding syntax in word representations. In *Proceedings of NAACL-HLT*, 2019.

20. Geoffrey E Hinton, James L McClelland, and David E Rumelhart. *Parallel distributed processing*, chapter Distributed representations, pages 77 – 109. MIT Press, 1986.
21. Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of SIGIR*, 1999.
22. Goodfellow Ian, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.
23. Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.
24. Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. Linguistic knowledge and transferability of contextual representations. In *Proceedings of NAACL-HLT*, 2019.
25. Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation Learning for Natural Language Processing*. Springer, 2020.
26. Peter Hugoe Matthews. *The concise Oxford dictionary of linguistics, third edition*, chapter Structural linguistics. Oxford University Press, 2014.
27. James L McClelland, David E Rumelhart, PDP Research Group, et al. *Parallel distributed processing*. MIT Press, 1986.
28. Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
29. Qiaozhu Mei, Xuehua Shen, and ChengXiang Zhai. Automatic labeling of multinomial topic models. In *Proceedings of KDD*, pages 490–499, 2007.
30. T Mikolov and J Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of NeurIPS*, 2013.
31. Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of EMNLP*, 2014.
32. Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, 2018.
33. Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.
34. Beatrice Santorini. Part-of-speech tagging guidelines for the penn treebank project. 1990.
35. Claude E Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.
36. Yiming Yang and Jan O Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of ICML*, pages 412–420, 1997.
37. Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004.
38. Jing Zhang, Bo Chen, Lingxi Zhang, Xirui Ke, and Haipeng Ding. Neural, symbolic and neural-symbolic reasoning on knowledge graphs. *AI Open*, 2:14–35, 2021.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Chapter 2
Word Representation Learning

Shengding Hu, Zhiyuan Liu, Yankai Lin, and Maosong Sun

Abstract Words are the building blocks of phrases, sentences, and documents. Word representation is thus critical for natural language processing (NLP). In this chapter, we introduce the approaches for word representation learning to show the paradigm shift from symbolic representation to distributed representation. We also describe the valuable efforts in making word representations more informative and interpretable. Finally, we present applications of word representation learning to NLP and interdisciplinary fields, including psychology, social sciences, history, and linguistics.

2.1 Introduction

The nineteenth-century philosopher Wilhelm von Humboldt described language as *the infinite use of finite means*, which is frequently quoted by many linguists such as Noam Chomsky, the father of modern linguistics. Apparently, the vocabulary in human language is a finite set of words that can be regarded as a kind of *finite means*. Words can be *infinitely used* as building blocks of phrases, sentences, and documents. As human beings start learning languages from words, machines need to understand each word first so as to master the sophisticated meanings of human languages. Hence, effective word representations are essential for natural language processing (NLP), and it is also a good start for introducing representation learning in NLP.

We can consider word representations as the knowledge of the semantic meanings of words. As discussed in Chap. 1, we can investigate word representations

S. Hu · Z. Liu (✉) · M. Sun
Department of Computer Science and Technology, Tsinghua University, Beijing, China
e-mail: shengdinghu@gmail.com; liuzy@tsinghua.edu.cn; sms@tsinghua.edu.cn

Y. Lin
Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China
e-mail: yankailin@ruc.edu.cn

© The Author(s) 2023

Z. Liu et al. (eds.), *Representation Learning for Natural Language Processing*,
https://doi.org/10.1007/978-981-99-1600-9_2

from two aspects, how knowledge is organized and where knowledge is from, i.e., the **form** and **source** of word representations.

The form of word representation can be divided into the **symbolic representation** (Sect. 2.2) and the **distributed representation** (Sect. 2.3), which respectively correspond to *symbolism* and *connectionism* mentioned in Chap. 1. Both forms represent words into vectors to facilitate computer processing. The essential difference between these two approaches lies in the meaning of each dimension. In symbolic word representation, each dimension has clear meanings, corresponding to concrete concepts such as words and topics. The symbolic representation form is straightforward to human understanding and has been adopted by linguists and old-fashioned AI (OFAI). However, it's not optimal for computers due to high dimensionality and sparsity issues: computers need large storage for these high-dimensional representations, and computation is less meaningful because most entries of the representations are zeros. Fortunately, the distributed word representation overcomes these problems by representing words as *low-dimensional* and *real-valued* dense vectors. In distributed word representation, each dimension in isolation is meaningless because semantics is distributed over all dimensions of the vector. Distributed representations can be obtained by factorizing the matrices of symbolic representations or learned by gradient descent optimization from data. In addition to overcoming the aforementioned problems of symbolic representation, it handles emerging words easily and accurately.

The effectiveness of word representation is also determined by the source of word semantics. A word in most alphabetic languages, such as English, is usually a sequence of characters. The internal structure usually reflects its speech or sound but helps little in understanding word semantics, except for some informative prefixes and suffixes. By taking human languages as a typical and complicated symbolic system as *structuralism* suggests (Chap. 1), words obtain their semantics from their relationship to *other words*. Given a word, we can find its hypernyms, synonyms, hyponyms, and antonyms from a human-organized **linguistic knowledge base (KB)** like WordNet [52] to represent word semantics. By extending *structuralism* to the *distributional hypothesis*, i.e., *you shall know a word by the company it keeps* [24], we can build word representations from their rich context in **large-scale text corpora**. Since most linguistic knowledge graphs are usually annotated by linguists, they are convenient to be used by humans but difficult to comprehensively and immediately reflect the dynamics of human languages. Meanwhile, word representations obtained from large-scale text corpora can capture up-to-date semantics of words in the real world with few subjective biases.

We can summarize existing methods of word representation as a mix of the above two perspectives. In the era of statistical NLP, word representation follows the symbolic form, obtained either from a linguistic knowledge graph (Fig. 2.1a) or from large-scale text corpora (Fig. 2.1b), which will be introduced in Sect. 2.2.

In the era of deep learning, distributed word representation follows the spirits of *connectionism* and *empiricism*. It learns powerful low-dimensional word vectors from large-scale text corpora and achieves ground-breaking performance on numerous tasks (Fig. 2.1c). In Sect. 2.3, we will present representative works

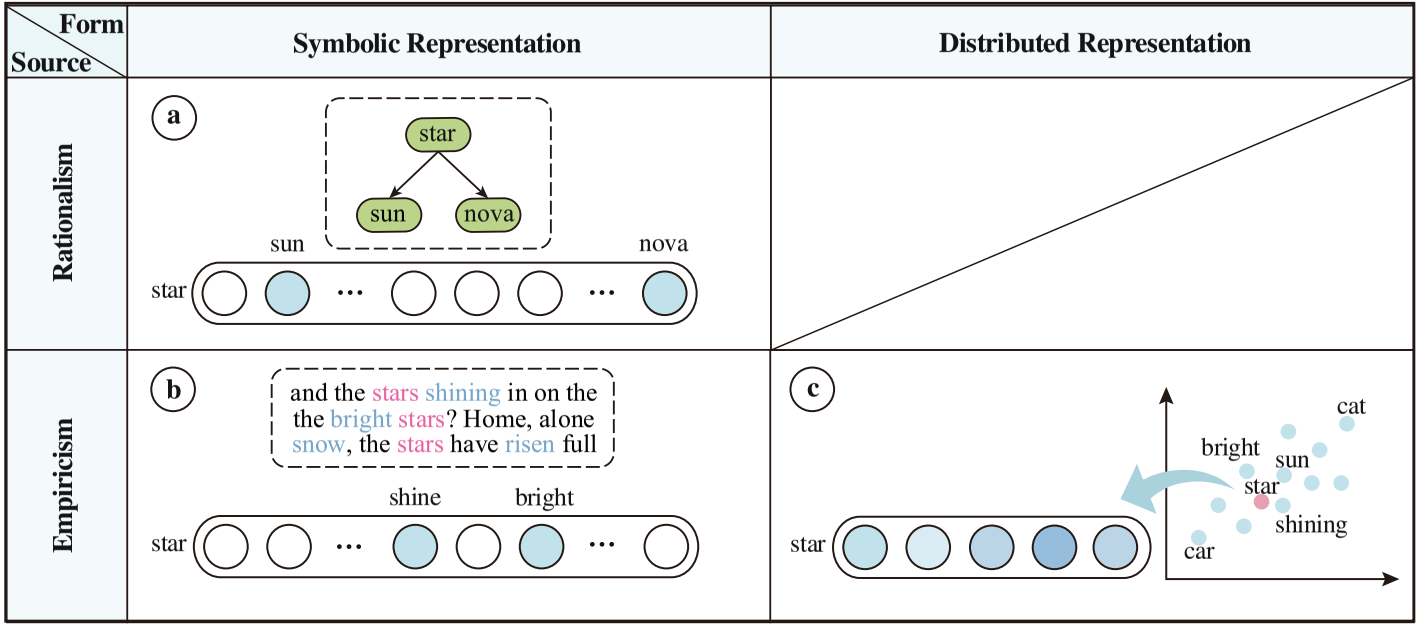


Fig. 2.1 The word representations can be divided according to their form of representation and source of the semantics: (**a**) shows the symbolic representations that use the knowledge base as the source, which is adopted by conventional linguistics; (**b**) shows the symbolic representations that adopt the distributional hypothesis as the foundation of the semantic source; (**c**) shows the distributed representation learned from large-scale corpora based on the distributional hypothesis, which is the mainstream of nowadays word representation learning

of distributed word representation such as word2vec [48] and GloVe [57]. These methods typically assign a fixed vector for each word and learn from text corpora. To address those words with multiple meanings under different contexts, researchers further propose contextualized word representation to capture sophisticated word semantics dynamically. The idea also inspires subsequent pre-trained models, which will be introduced in Chap. 5.

Many efforts have been devoted to constructing more *informative* word representations by encoding more information, such as multilingual data, internal character information, morphology information, syntax information, document-level information, and linguistic knowledge, as introduced in Sect. 2.4. Moreover, it would be a bonus if some degree of *interpretability* is added to word representation, and we will also briefly describe improvements in interpretable word representation.

Word representation learning has been widely used in many applications in NLP and other areas. In NLP, word representations can be applied to word-level tasks such as word similarity and analogy and simple downstream tasks such as sentiment analysis. We note that, with the advancement of deep learning and pre-trained models, word representations are less used in isolation in NLP but more as building blocks of neural language models, as shown in Chaps. 3, 4, and 5. Meanwhile, word representations play indispensable roles in interdisciplinary fields such as computational social sciences for studying social bias and historical change.

2.2 Symbolic Word Representation

Since the ancient days of knotted strings, human ancestors have used symbols to record and share information. As time progressed, isolated symbols gradually merged to form a symbol system. This system is human language. In fact, human language is probably the most complex and systematic symbol system that humans have ever built. In human language, each word is a discrete symbol that contains a wealth of semantic meaning. Therefore, ancient linguists also regard each word as a discrete symbol.

This common practice can also apply to NLP in modern computer science. In this section, we introduce three traditional symbolic approaches to word representations, i.e., one-hot word representation, linguistic KB-based word representation, and corpus-based word representation.

2.2.1 One-Hot Word Representation

One-hot representation is the simplest way for symbol-based word representation, which can be formalized as follows. Given a finite set of word vocabulary $V = \{w^{(1)}, w^{(2)}, \ldots, w^{(|V|)}\}$, where $|V|$ is the vocabulary size, one-hot representation represents an i-th word $w^{(i)}$ with a $|V|$-dimensional vector $\mathbf{w}^{(i)}$, where only the i-th

dimension has a value 1 while all other dimensions are 0. That is, each dimension $\mathbf{w}_j^{(i)}$ is defined as:

$$\mathbf{w}_j^{(i)} = \begin{cases} 1, & \text{if } j = i, \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

In essence, the one-hot word representation maps each word to an index of the vocabulary. However, it can only distinguish between different words and does not contain any syntactic or semantic information. For any two words, their one-hot vectors are orthogonal to each other. That is, the cosine similarity between *cat* and *dog* is the same as the similarity between *cat* and *sun*, which are both zeros.

Although we do not have much to talk about one-hot word representation itself, it is the foundation of bag-of-words models for document representations, which are widely used in information retrieval and text classification. Readers can refer to document representation learning methods in Chap. 4.

As mentioned, there is no internal semantic structure in the one-hot representation. To incorporate semantics in the representation, we will present two methods with different sources of semantics: linguistic KB and natural corpus.

2.2.2 Linguistic KB-based Word Representation

As we introduced in Chap. 1, *rationalism* regards the introspective reasoning process as the source of knowledge. Therefore, the researchers construct a complex word-to-word network by reflecting on the relationship between words. For example, human linguists manually annotate the synonyms and hypernyms[1] of each word. In the well-known linguistic knowledge base WordNet [52], the hypernyms and hyponyms of *dog* are annotated as Fig. 2.2. To represent a word, we can use the vector forms just like one-hot representation as follows:

$$\mathbf{w}_j^{(i)} = \begin{cases} 1 & \text{if } (w^{(i)}, \texttt{has\_hypernym}, w^{(j)}), \\ -1 & \text{if } (w^{(i)}, \texttt{has\_hyponym}, w^{(j)}), \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

But it is clear that this representation has limited expressive power, where the similarity of two words without common hypernyms and hyponyms is 0. It would be better to directly adopt the original graph form, where the similarity between the two words can be derived using metrics on the graph. For synonym networks, we can

[1] Hypernyms are words whose meaning includes a group of other words, which are instances of the former. Word u is a hyponym of word v if and only if word v is a hypernym of word u.

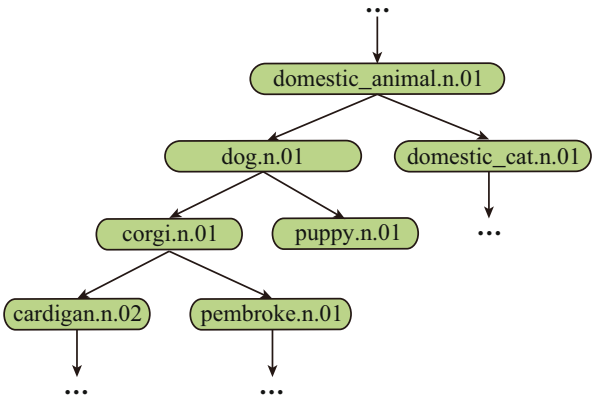


Fig. 2.2 Hypernyms and hyponyms of *dog* in WordNet [52]. The *dog.n.01* denotes the first synset of *dog* used as a noun

calculate the distance between two words on the network as their semantic similarity (i.e., the shortest path length between the two words). Hierarchical information can be utilized to better measure the similarity for hypernym-hyponym networks. For example, the information content (IC) approach [61] is proposed to calculate the similarity based on the assumption that the lower the frequency of the closest hypernym of two words is, the closer the two words are.

Formally, we define the similarity s as follows:

$$\mathrm{s}(w_1, w_2) = \max_{w \in C(w_1, w_2)}[-\log P(w)], \tag{2.3}$$

where $C(w_1, w_2)$ is the common hypernym set of w_1 and w_2 and $P(w)$ is the probability of word w's appearance in the corpus.[2] Intuitively, $P(w)$ is the generality of the word w. It indicates that if all common hypernyms of w_1 and w_2 are very general, then $\mathrm{s}(w_1, w_2)$ will be very small. But if some hypernyms of w_1 and w_2 are specific, $\mathrm{s}(w_1, w_2)$ will have a higher score, which indicates that these two words are closely related to each other. A vivid example is shown in Fig. 2.3.

2.2.3 Corpus-based Word Representation

The process of constructing a linguistic KB is labor-intensive. In contrast, it is much easier to collect a corpus. This motivation is also supported by *empiricism*, which emphasizes knowledge from naturally produced data.

[2] Estimated by dividing the frequency of a word by the total number of words in the corpus.

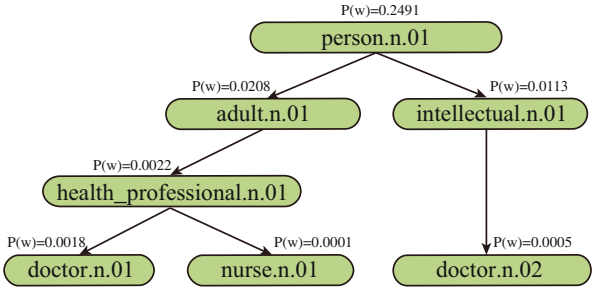


Fig. 2.3 *doctor.n.01* and *nurse.n.01* share a rare ancestor *health_professional.n.01*, their similarity is large. But the closest common ancestor of *doctor.n.02* and *nurse.n.01* is *people.n.01*, which is common. Therefore the similarity between them is small

The correctness of automatically derived representations from a corpus relies on the linguistic hypothesis behind them. We start with the **bag-of-words hypothesis**. To illustrate this hypothesis, we temporarily shift our attention to document representation. This hypothesis states that we can ignore the order of words in a document and simply treat the document as a bag (i.e., a multiset[3]) of words. Then the frequencies of the words in the bag can reflect the content of the document [66]. In this way, a document is represented by a *row vector* in which each element indicates the presence or frequency of a word in the document. For example, the value of the entry corresponding to word *cat* being 3 means that *cat* occurs three times in the document, and an entry corresponding to a word being 0 means that the word is not in the document [67]. In this way, we have automatically constructed a representation of the document.

How does this inspire us to construct the word representations of greater interest in this chapter? In fact, as we stack the row vectors of each document to form a document (row)-word (column) matrix, we can shift our attention from rows to columns [17]. Each column now represents the occurrence of a word in a stack of documents. Intuitively, if the words *rat* and *cat* tend to occur in the same documents, their statistics in the columns will be similar.

In the above approach, a document can be considered as the *context* of a word. Actually, more flexibility can be added in defining the context of a word to obtain other kinds of representations. For example, we can define a fixed-size window centered on a word and use the words inside the window as the context of that word. This corresponds to the well-known **distributional hypothesis** that the meaning of a word is described by its companions [24]. Then we count the words that appear in a word's neighborhood and use a dictionary as a word representation, where each key is a context word whose value is the frequency of the occurrence of that context word within a certain distance.

[3] A multiset is a set where duplicated elements are allowed, e.g., (1,2,2,3) and (2,2,3,1) are the same multiset.

To further extend the context of a word, several works propose to include dependency links [56] or links induced by argument positions [21]. Interested readers can refer to a summary of various contexts used for corpus-based distributional representations [65].

In summary, in symbolic representations, each entry of the representation has a clear and interpretable meaning. The clear interpretable meaning can correspond to a specific word, synset, or term, and that is why we call it "symbolic representation."

2.3 Distributed Word Representation

Although simple and interpretable, symbolic representations are not the best choice for computation. For example, the very sparse nature of the symbolic representation makes it difficult to compute word-to-word similarities. Methods like information content [61] cannot naturally generalize to other symbolic representations.

The difficulty of symbolic representation is solved by the distributed representation.[4] Distributed representation represents a subject (here is a word) as a fixed-length real-valued vector, where no clear meaning is assigned to every single dimension of the vector. More specifically, semantics is scattered over all (or a large portion) of the dimensions of the representation, and one dimension contributes to the semantics of all (or a large proportion) of the words.

We must emphasize that the "*distributed representation*" is completely different from and orthogonal to the "*distributional representation*" (induced by "*distributional hypothesis*"). Distributed representation describes the form of a representation, while distributional hypothesis (representation) describes the source of semantics.

2.3.1 Preliminary: Interpreting the Representation

Although each dimension is uninterpretable in distributed representation, we still want ways to interpret the meaning conveyed by the representation approximately. We introduce two basic computational methods to understand distributed word representation: similarity and dimension reduction.

Suppose the representations of two words are $\mathbf{u} = [u_1, \ldots, u_d]$ and $\mathbf{v} = [v_1, \ldots, v_d]$,[5] we can calculate the similarity or perform dimension reduction as follows.

[4] Models of distributed representations are also called vector space models (VSMs).

[5] In the following sections, we use the row vector for distributed word representation.

Similarity The Euclidean distance is the $L2$-norm of the difference vector of $\mathbf{u}$ and $\mathbf{v}$.

$$d_{\text{Euclidean}}(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{d} |u_i - v_i|^2}. \tag{2.4}$$

Then the Euclidean similarity can be defined as the inverse of distance, i.e.,

$$s_{\text{Euclidean}}(\mathbf{u}, \mathbf{v}) = 1 \Big/ \sqrt{\sum_{i=1}^{d} |u_i - v_i|^2}. \tag{2.5}$$

Cosine similarity is also common. It measures the similarity by the angle between the two vectors:

$$s_{\text{cosine}}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|_2 \|\mathbf{v}\|_2}. \tag{2.6}$$

Dimension Reduction Distributed representations, though being lower dimensional than symbolic representations, still exist in manifolds higher than three dimensions. To visualize them, we need to reduce the dimension of the vector to 2 or 3. Many methods have been proposed for this purpose. We will briefly introduce principal component analysis (PCA).

PCA transforms the vectors into a set of new coordinates using an orthogonal linear transformation. In the new coordinate system, an axis is pointed in the direction which explains the data's most variance while being orthogonal to all other axes. Under this construction, the later constructed axes explain less variance and therefore are less important to fit the data. Then we can use only the first two to three axes as the principal components and omit the later axes. A case of PCA on two-dimensional data is in Fig. 2.4. Formally, denoting the new axes by a set of unit row vectors $\{\mathbf{d}_j | j = 1, \ldots, k\}$, where k is the number of unit row vectors. An original vector $\mathbf{u}$ of the sample u can be represented in the new coordinates by

$$\mathbf{u} = \sum_{j=1}^{k} a_j \mathbf{d}_j \approx \sum_{j=1}^{r} a_j \mathbf{d}_j, \tag{2.7}$$

where a_j is the weight of the vector $\mathbf{d}_j$ for representing $\mathbf{u}$, and $\mathbf{a} = [a_1, \ldots, a_r]$ forms the new vector representation of u. In practice, we only set $r = 2$ or 3 for visualization.

The set of new coordinates $\{\mathbf{d}_j | j = 1, \ldots, k\}$ can be computed by *eigen-decomposition* of the covariance matrix or using *singular value decomposition (SVD)*. Then we introduce SVD-based PCA and present its resemblance to latent

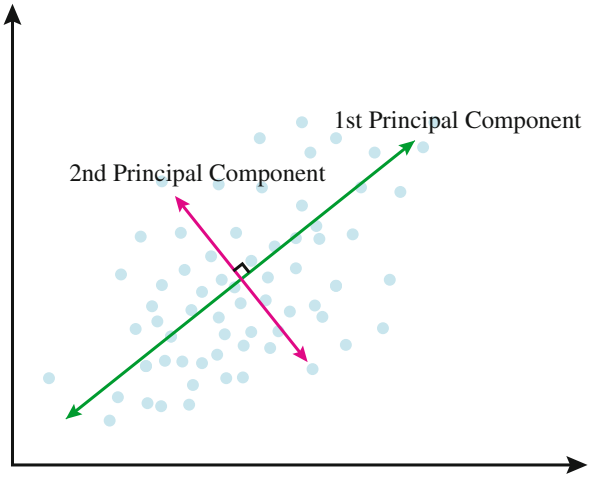


Fig. 2.4 The PCA specifies new axis directions (called principal components), and the axes are ordered from largest to smallest variance in their directions so that keeping the coordinates on the first few axes retains most of the distribution information

semantic analysis (LSA) in the next subsection. In SVD, a real-valued matrix can be decomposed into

$$\mathbf{U} = \mathbf{W}\Sigma\mathbf{D}, \tag{2.8}$$

such that Σ is a diagonal matrix of positive real numbers, i.e., the singular values, and $\mathbf{W}$ and $\mathbf{D}$ are singular matrices formed by orthogonal vectors. For a data sample (e.g., i-th row in $\mathbf{U}$):

$$\mathbf{U}_{i,:} = \sum_{j=1}^{k} \sigma_j \mathbf{W}_{i,j} \mathbf{D}_{j,:} \,. \tag{2.9}$$

Thus in Eq. (2.7), $a_j = \sigma_j \mathbf{W}_{i,j}$ and $\mathbf{d}_j = \mathbf{D}_{j,:}$.

Although widely adopted in high-dimensional data visualization, PCA is unable to visualize representations that form nonlinear manifolds. Other dimensionality reduction methods, such as t-SNE [73], can solve this problem.

2.3.2 Matrix Factorization-based Word Representation

Distributed representations can be transformed from symbolic representations by matrix factorization or neural networks. In this subsection, we introduce the matrix factorization-based methods. We introduce latent semantic analysis (LSA), its probabilistic version PLSA, and latent Dirichlet allocation (LDA) as the representative approaches. Readers who are only interested in neural networks can jump to the next section to continue reading.

Latent Semantic Analysis (LSA) LSA [17] utilizes singular value decomposition (SVD) to perform the transformation from matrices of symbolic representations. Suppose we have a word-document matrix $\mathbf{M} \in \mathbb{R}^{n \times d}$, where n is the number of words and d is the number of documents. By linear algebra, it can be uniquely[6] decomposed into the multiplication of three matrices $\mathbf{W} \in \mathbb{R}^{n \times n}$, $\Sigma \in \mathbb{R}^{n \times d}$, and $\mathbf{D} \in \mathbb{R}^{d \times d}$:

$$\mathbf{M} = \mathbf{W}\Sigma\mathbf{D}^\top, \tag{2.10}$$

such that Σ is a diagonal matrix of positive real numbers, i.e., the singular values, and columns of $\mathbf{W}, \mathbf{D}$ are left-singular vectors and right-singular vectors, respectively.[7]

Now let's try to interpret the two orthogonal matrices. The i-th row of the matrix $\mathbf{M}$ that represents the i-th word's symbolic representation (denoted by $\mathbf{M}_{i,:}$) is decomposed into:

$$\mathbf{M}_{i,:} = \mathbf{W}_{i,:}\Sigma\mathbf{D}^\top. \tag{2.11}$$

From Eq. (2.11), we can see only the i-th row of $\mathbf{W}$ contributes to the i-th word's symbolic representation. More importantly, since $\mathbf{D}$ is an orthogonal matrix, the similarity between words w_i and w_j is given by:

$$\mathbf{M}_{i,:}\mathbf{M}_{j,:}^\top = \mathbf{W}_{i,:}\Sigma^2\mathbf{W}_{j,:}^\top. \tag{2.12}$$

Thus we can take $\mathbf{W}_{i,:}\Sigma$ as the distributed representation for word w_i. Note that taking $\mathbf{W}_{i,:}\Sigma$ or $\mathbf{W}_{i,:}$ as the distributed representation is both ok because $\mathbf{W}_{i,:}\Sigma$ is $\mathbf{W}_{i,:}$ stretched along each axis j with ratio $\Sigma_{j,j}$, and the relative positions of points in the two spaces are similar.

Suppose we arrange the eigenvalues in descending order. In that case, the largest K singular values (and their singular vectors) contribute the most to matrix $\mathbf{M}$. Thus we only use them to approximate $\mathbf{M}$. Now Eq. (2.11) becomes:

$$\mathbf{M}_{i,:} \approx \mathbf{W}_{i,:K} \odot [\sigma_1, \sigma_2, \ldots, \sigma_K]\mathbf{D}_{:,:K}^\top, \tag{2.13}$$

where $\odot$ is the element-wise multiplication and σ_i is the i-th diagonal element of Σ.

To sum up, in LSA, we perform SVD to the counting statistics to get the distributed representation $\mathbf{W}_{i,:K}$. Usually, with a much smaller K, the approximation can be sufficiently good, which means the semantics in the high-dimensional

[6] Up to permutations of rows, columns, and signs.

[7] In fact, the mathematical backbone of LSA is the same as PCA. We repeat it here for the convenience of those readers who skipped the previous section.

symbolic representation are now compressed and distributed into a much lower-dimensional real-valued vector. LSA has been widely used to improve the recall of query-based document ranking in information retrieval since it supports ambiguous semantic matching.

One challenge of LSA comes from the computational cost. A full SVD on an $n \times d$ matrix requires $\mathcal{O}(\min\{n^2 d, nd^2\})$ time, and the parallelization of SVD is not trivial. A solution is random indexing [36, 64] that overcomes the computational difficulties of SVD-based LSA and avoids expensive preprocessing of a huge word-document matrix. In random indexing, each document is assigned a randomly-generated high-dimensional sparse ternary vector (named as *index vector*). Note that random vectors in high-dimensional space should be (nearly) orthogonal, analogous to the orthogonal matrix $\mathbf{D}$ in SVD-based LSA. For each word in a document, we add the document's index vector to the word's vector. After passing the whole text corpora, we can get accumulated word vectors. Random indexing is simple to parallelize and implement, and its performance is comparable to the SVD-based LSA [64].

Probabilistic LSA (PLSA) LSA further evolves into PLSA [34]. To understand PLSA based on LSA, we can treat the $\mathbf{W}_{i,:K}$ as a distribution over latent factors $\{z_k | k = 1, \ldots, K\}$, where

$$\mathbf{W}_{i,k} = P(w_i | z_k), \quad k = 1, \ldots, K. \tag{2.14}$$

We can understand these factors as "topics" as we will assign meanings to them later. Similarly, $\mathbf{D}_{i,:K}$ can also be regarded as a distribution, where

$$\mathbf{D}_{j,k} = P(d_j | z_k), \quad k = 1, \ldots, K. \tag{2.15}$$

And the $\{\sigma_k | k = 1, \ldots, K\}$ are the prior probabilities of factors z_k, i.e., $P(z_k) = \sigma_k$. Thus, the word-document matrix becomes a joint probability of word w_j and document d_j:

$$\mathbf{M}_{i,j} = P(w_i, d_j) = \sum_{k=1}^{K} P(w_i | z_k) P(z_k) P(d_j | z_k). \tag{2.16}$$

With the help of Bayes' theorem and Eq. (2.16), we can compute the conditional probability of w_i given a document d_j is

$$P(w_i | d_j) = \frac{P(w_i, d_j)}{P(d_j)} = \sum_{k=1}^{K} P(w_i | z_k) \frac{P(d_j | z_k) P(z_k)}{P(d_j)} = \sum_{k=1}^{K} P(w_i | z_k) P(z_k | d_j). \tag{2.17}$$

Now we can see a generative process is defined from Eq. (2.17). To generate a word in the document d_j, we first sample a latent factor z_k from $P(z_k | d_j)$ and

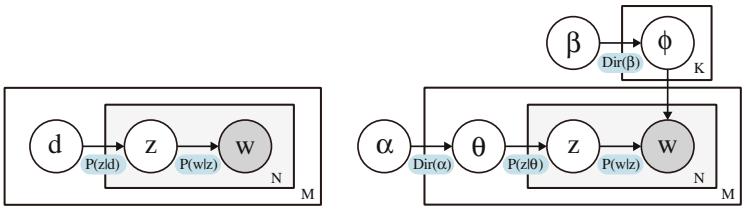


Fig. 2.5 The generative process of words in documents in the PLSA model (left) and LDA model (right). N is the number of words in a document, M is the number of documents, and K is the number of topics. w is the only observed variable, and we need to estimate the other white variables based on the observation. The figure is redrawn according to the Wikipedia entry "Latent Dirichlet allocation"

then sample a word w_i from the conditional probability $P(w_i|z_k)$. The process is represented in Fig. 2.5.

Note that to make Eq. (2.14)∼Eq. (2.17) rigorous, the elements of $\mathbf{W}, \Sigma, \mathbf{D}$ have to be nonnegative. To do optimization under such probabilistic constraints, a different loss from SVD is used [34]:

$$\mathcal{L} = -\sum_j \sum_i \mathbf{M}_{i,j} \log P(w_i, d_j). \tag{2.18}$$

The following works prove that optimizing the above objective is the same as nonnegative matrix factorization [20, 38]. We omit the mathematical details here.

Latent Dirichlet Allocation (LDA) PLSA is further developed into latent Dirichlet allocation (LDA) [10], a popular topic model that is widely used in document retrieval. LDA adds hierarchical Bayesian priors to the generative process defined by Eq. (2.14). The generative process for words in document j becomes:

1. Choose $\boldsymbol{\theta}_j \in \mathbb{R}^K \sim \text{Dir}(\boldsymbol{\alpha})$, where $\text{Dir}(\boldsymbol{\alpha})$ is a Dirichlet distribution (typically each dimension of $\boldsymbol{\alpha} < 1$), where K is the number of topics. This is the probability distribution of topics in the document d_j.
2. Choose $\boldsymbol{\phi}_z \in \mathbb{R}^{|V|} \sim \text{Dir}(\boldsymbol{\beta})$ for each topic z, where $|V|$ is the size of the vocabulary. Typically each dimension of $\boldsymbol{\beta}$ is less than 1. This is the probability distribution of words produced by topic z.
3. For each word w_i in the document d_j:
   a. Choose a topic $z_{i,j} \sim \text{Multinomial}(\boldsymbol{\theta}_j)$.
   b. Choose a word from $w_{i,j} = P(w_j|d_j) \sim \text{Multinomial}(\boldsymbol{\phi}_{z_{i,j}})$.

The generative process in LDA is in Fig. 2.5 (right). We will not dive into the mathematical details of LDA.

We would like to emphasize two points about LDA: (1) the hyper-parameter $\boldsymbol{\alpha}, \boldsymbol{\beta}$ in Dirichlet prior is typically set to be less than 1, resulting in a "sparse prior", i.e., most dimensions of the sampled $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are close to zero, and the mass of the

distribution will be concentrated in a few values. This is consistent with our common sense that a document will always have only a few topics and that a topic will only produce a small number of words. Moreover, the total number of topics K is predefined as a relatively small integer. The sparsity and interpretability make LDA essentially a kind of *symbolic representation*, and LDA can be seen as *a bridge between distributed representations and symbolic representations*. (2) Although PLSA and LDA are more often used in document retrieval, the distribution of a word over different topics (latent factors) $P(w|z_i)$ can be used as an effective word representation, i.e., $\mathbf{w} = [P(w|z_1), \ldots, P(w|z_K)]$.

However, the information source, i.e., counting matrix $\mathbf{M}$, of matrix factorization-based methods is still based on the bag-of-words hypothesis. These methods lose the word order information in the documents, so their expressiveness capability remains limited. Therefore, these classical methods are less often used when neural network-based methods that can model word order information emerge.

2.3.3 Word2vec and GloVe

The neural networks, revived in the 2010s, are similar to the neurons of the human brain, where the neurons inside a neural network perform distributed computation. One neuron is responsible for the computation of multiple pieces of information, and one input activates multiple neurons at the same time. This property coincides with distributed representation. Hence, distributed representation plays a dominant role in the era of neural networks. Moreover, neural models are optimized on large-scale data. The data dependency makes the distributional hypothesis particularly important in optimizing such distributed representations. In the following section, we first present **word2vec** [48], a milestone work of distributional distributed word representation using neural approaches. After that, we introduce **GloVe** [57] that improves word2vec with a global word-occurrence matrix.

Word2vec Word2vec adopts the distributional hypothesis but does not take a count-based approach. It directly uses gradient descent to optimize the representation of a word toward its neighbors' representations. Word2vec has two specifications, namely, **continuous bag-of-words (CBOW)** and **skip-gram**. The difference is that CBOW predicts a center word based on multiple context words, while skip-gram predicts multiple context words based on the center word.

CBOW predicts the center word given a window of context. Figure 2.6 shows the idea of CBOW with a window of $five$ words.

Formally, CBOW predicts w_i according to its contexts as:

$$P(w_i|w_{i-l}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+l}) = \text{Softmax}\left(\mathbf{W}\left(\sum_{i-l \le j \le i+l, j \ne i} \mathbf{w}_j^\top\right)\right), \tag{2.19}$$

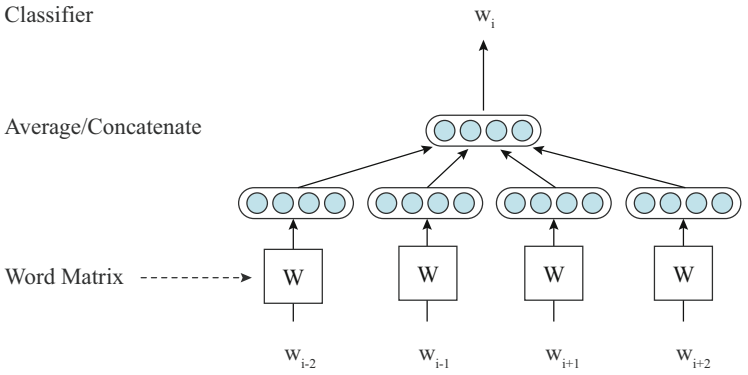


Fig. 2.6 The architecture of the CBOW model. (The figure is redrawn according to Fig. 1 from Mikolov et al. [49])

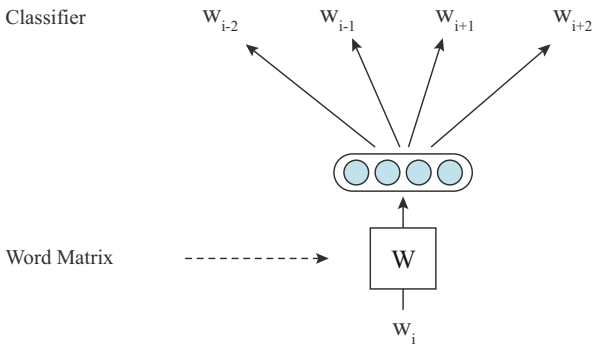


Fig. 2.7 The architecture of the skip-gram model. (The figure is redrawn according to Fig. 1 from Mikolov et al. [49])

where $P(w_i|w_{i-l},\ldots,w_{i-1},w_{i+1},\ldots,w_{i+l})$ is the probability of word w_i given its contexts, $2l+1$ is the size of training contexts, $\mathbf{w}_j$ is the word vector of word w_j, $\mathbf{W}$ is the weight matrix in $\mathbb{R}^{|V|\times m}$, V indicates the vocabulary, and m is the dimension of the word vector.

The CBOW model is optimized by minimizing the sum of the negative log probabilities:

$$\mathcal{L} = -\sum_i \log P(w_i|w_{i-l},\ldots,w_{i-1},w_{i+1},\ldots,w_{i+l}). \tag{2.20}$$

Here, the window size l is a hyper-parameter to be tuned. A larger window size may lead to higher accuracy as well as longer training time.

Contrary to CBOW, *skip-gram* predicts the context given the center word. Figure 2.7 shows the model.

Formally, given a word w_i, skip-gram predicts its context as:

$$P(w_j|w_i) = \text{Softmax}(\mathbf{W}\mathbf{w}_i^\top), \quad \text{where} \quad i-l \le j \le i+l, j \ne i, \tag{2.21}$$

where $P(w_j|w_i)$ is the probability of context word w_j given w_i and $\mathbf{W}$ is the weight matrix. The loss function is similar to CBOW but needs to sum over multiple context words:

$$\mathcal{L} = -\sum_i \sum_{i-l \le j \le i+l, j \ne i} P(w_j|w_i). \tag{2.22}$$

In the early stages of the deep learning renaissance, computational resources are still limited, and it is time-consuming to optimize the above objectives directly. The most time-consuming part is the softmax layer since the softmax layer uses the scores of predicting all words in the vocabulary V in the denominator:

$$P(w_j|w_i) = \text{Softmax}(\mathbf{W}\mathbf{w}_i^\top) = \frac{\exp(\mathbf{W}_{j,:}\mathbf{w}_i^\top)}{\sum_{j,w_j \in V} \exp(\mathbf{W}_{j,:}\mathbf{w}_i^\top)}. \tag{2.23}$$

An intuitive idea to improve efficiency is obtaining a reasonable but faster approximation of the softmax score. Here, we present two typical approximation methods, including **hierarchical softmax** and **negative sampling**. We explain these two methods using CBOW as an example.

The idea of hierarchical softmax is to build hierarchical classes for all words and to estimate the probability of a word by estimating the conditional probability of its corresponding hierarchical classes. Figure 2.8 gives an example. Each internal node of the tree indicates a hierarchical class and has a feature vector, while each leaf node of the tree indicates a word. The conditional probabilities, e.g., p_0 and p_1 in Fig. 2.8, of two child nodes are computed by the feature vector of each node and the context vector. For example,

$$p_0 = \frac{\exp(\mathbf{w}_0\mathbf{w}_c^\top)}{\exp(\mathbf{w}_0\mathbf{w}_c^\top) + \exp(\mathbf{w}_1\mathbf{w}_c^\top)}, \tag{2.24}$$

$$p_1 = 1 - p_0, \tag{2.25}$$

where $\mathbf{w}_c$ is the context vector, $\mathbf{w}_0$ and $\mathbf{w}_1$ are the feature vectors.

Then, the probability of a word can be obtained by multiplying the probabilities of all nodes on the path from the root node to the corresponding leaf node. For example, the probability of the word *the* is $p_0 \times p_{01}$, while the probability of *cat* is $p_0 \times p_{00} \times p_{001}$.

The tree of hierarchical classes is generated according to the word frequencies, which is called the Huffman tree. Through this approximation, the computational complexity of the probability of each word is $\mathcal{O}(\log|V|)$.

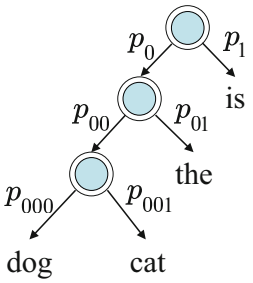


Fig. 2.8 An illustration of hierarchical softmax

Negative sampling is a more straightforward technique. It directly samples k words as negative samples according to the word frequency. Then, it computes a softmax over the $k + 1$ words (1 for the positive sample, i.e., the target word) to approximate the conditional probability of the target word.

GloVe The word2vec and matrix factorization-based methods have complementary advantages and disadvantages. In terms of learning efficiency and scalability, word2vec is superior because word2vec uses an online learning (or batch learning paradigm in deep learning) approach and is able to learn over large corpora. However, considering the preciseness of distribution modeling, the matrix factorization-based methods can exploit global co-occurrence information by building a global co-occurrence matrix. In comparison, word2vec is a local window-based method that cannot see the frequency of word pairs in a global corpus in a single optimization step. Therefore, GloVe [57] is proposed to combine the advantages of word2vec and matrix factorization-based methods.

To learn from global count statistics, GloVe firstly builds a co-occurrence matrix $\mathbf{M}$ over the entire corpus but does not directly factorize it. Instead, it takes each entry in the co-occurrence matrix $\mathbf{M}_{ij}$ and optimizes the following target:

$$\mathcal{L} = \sum_{i,j=1}^{|V|} f(\mathbf{M}_{ij}) d(\mathbf{w}_i, \mathbf{w}_j, \mathbf{M}_{ij}). \tag{2.26}$$

The $d(\mathbf{w}_i, \mathbf{w}_j, \mathbf{M}_{ij})$ is a metric that compares the distributed representations of word w_i and w_j with the ground-truth statistics $\mathbf{M}_{ij}$. $f(\mathbf{M}_{ij})$ is a weight term measuring the importance of the word pair w_i, w_j. Specifically, GloVe adopts the following form as the metric d:

$$d(\mathbf{w}_i, \mathbf{w}_j, \mathbf{M}_{ij}) = (\mathbf{w}_i \mathbf{w}_j^\top + b_i + b_j - \log \mathbf{M}_{ij})^2, \tag{2.27}$$

where b_i and b_j are bias terms for word w_i and w_j. Interested readers can read the original paper [57] for the derivation details.

For the weight term $f(\mathbf{M}_{ij})$, most previous approaches set the weight of all word pairs to 1. However, common word pairs may have too weak semantics, so

we should lower their weights and increase the weights of rare word pairs slightly. Thus, GloVe observes that it should satisfy three constraints:

- $f(0) = 0$ and $\lim_{x \to 0} f(x) \log^2 x$ is finite.
- A nondecreasing function.
- Truncated for large values of x to avoid overfitting to common words (stop words).

A possible choice is the following, where α is taken as $\frac{3}{4}$ in the original GloVe paper:

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha}, & \text{if } x < x_{\max}, \\ 1, & \text{otherwise.} \end{cases} \tag{2.28}$$

In summary, GloVe uses weighted squared loss to optimize the representation of words based on the elements in the global co-occurrence matrix. Compared with word2vec, it captures global statistics. Compared with matrix factorization, it (1) reasonably reduces the weights of the most frequent words at the level of matrix entries, (2) reduces the noise caused by non-discriminative word pairs by implicitly optimizing the ratio of co-occurrence frequencies, and (3) enables fitting on large corpus by iterative optimization. Since the number of nonzero elements of the co-occurrence matrix is much smaller than $|V|^2$, the efficiency of GloVe is ensured in practice.

Word2vec as Implicit Matrix Factorization It seems so far that the neural network-based methods like word2vec and matrix factorization-based methods are two distinct paradigms for deriving distributed representation. But in fact, they have close theoretical connections. Omer et al. [41] prove that word2vec is factorizing pointwise mutual information matrix (PMI), where

$$\text{PMI}_{i,j} = \log \frac{P(w_i, w_j)}{P(w_i)P(w_j)}. \tag{2.29}$$

Omer et al. [41] then compare the performance of factoring the PMI matrix using SVD and skip-gram with the negative sampling (SGNS) model. SVD achieves a significantly better objective value when the embedding size is smaller than 500 dimensions and the number of negative samples is 1. With more negative samples and higher embedding dimensions, SGNS gets a better objective value. For downstream tasks, under several conditions, SVD achieves slightly better performance on word analogy and word similarity. In contrast, skip-gram with negative sampling achieves better performance by 2% on syntactical analogy.

2.3.4 Contextualized Word Representation

In natural language, the semantic meaning of an individual word can usually be different with respect to its context in a sentence. For example, in the two sentences: "*willows lined the bank of the stream.*", "*a bank account.*",[8] although the word *bank* is always the same, their meanings are different. This phenomenon is prevalent in any language. However, most of the traditional word embeddings (CBOW, skip-gram, GloVe, etc.) cannot well understand the different nuances of the meanings of words in the different surrounding texts. The reason is that these models only learn a unique and specific representation for each word. Therefore these models cannot capture how the meanings of words change based on their surrounding contexts.

Matthew et al. [58] propose ELMo to address this issue, whose word representation is a function of the whole input. More specifically, rather than having a look-up table of word embedding matrix, ELMo converts words into low-dimensional vectors on-the-fly by feeding the word and its context into a deep neural network. ELMo utilizes a bidirectional language model to conduct word representation. Formally, given a sequence of N words $(w_1, \ldots, w_N)$, a forward language model (LM)[9] models the probability of the sequence by predicting the probability of each word w_k according to the historical context:

$$P(w_1, w_2, \ldots, w_N) = \prod_{k=1}^{N} P(w_k \mid w_1, \ldots, w_{k-1}). \tag{2.30}$$

The forward LM in ELMo is a multilayer long short-term memory (LSTM) network [33], which is a kind of widely used neural network for modeling sequential data, and the j-th layer of the LSTM-based forward LM will generate the context-dependent word representation $\overrightarrow{\mathbf{h}}_{k,j}^{\mathrm{LM}}$ for the word w_k. The backward LM is similar to the forward LM. The only difference is that it reverses the input word sequence to $(w_N, w_{N-1}, \ldots, w_1)$ and predicts each word according to the future context:

$$P(w_1, w_2, \ldots, w_N) = \prod_{k=1}^{N} P(w_k \mid w_{k+1}, \ldots, w_N). \tag{2.31}$$

Similar to the forward LM, the j-th backward LM layer generates the representations $\overleftarrow{\mathbf{h}}_{k,j}^{\mathrm{LM}}$ for the word w_k.

When used in a downstream task, ELMo combines all layer representations of the bidirectional LM into a single vector as the *contextualized word representation*. The way to do the combination is flexible. For example, the final representation can be the weighting of all bidirectional LM layer's hidden representation, and the

[8] Examples are taken from the Oxford Dictionary of English [69].

[9] The details of the language model are in Chap. 5.

weights are task-specific:

$$\mathbf{w}_k = \alpha^{\text{task}} \sum_{j=0}^{L} s_j^{\text{task}} \mathbf{h}_{k,j}^{\text{LM}}, \tag{2.32}$$

where $\mathbf{s}^{\text{task}} = [s_1^{\text{task}}, \ldots, s_L^{\text{task}}]$ are softmax-normalized weights and α^{task} allows the task to scale the whole representation, and $\mathbf{h}_{k,j}^{\text{LM}} = \text{concat}(\overrightarrow{\mathbf{h}}_{k,j}^{\text{LM}}; \overleftarrow{\mathbf{h}}_{k,j}^{\text{LM}})$.

Due to the superiority of contextualized representations, a group of works led by ELMo [58], BERT [19], and GPT [59] has started to emerge since 2017, eventually leading to a unified paradigm of pre-training-fine-tuning across NLP. Please refer to Chap. 5 for further reading.

2.4 Advanced Topics

In the previous section, we introduced the basic models of word representation learning. These studies promoted more work on pursuing better word representations. In this section, we introduce the improvement from different aspects. Before we dive into the specific methods, let's first discuss the essential features of a **good** word representation.

Informative Word Representation A key point where representation learning differs from traditional prediction tasks is that when we construct representations, we do not know what information is needed for downstream tasks. Therefore, we should compress as much information as possible into the representation to facilitate various downstream tasks. From the development of one-hot representations to distributional and contextualized representations, the information in the representations is indeed increasing. And we still expect to incorporate more information into the representations.

Interpretable Word Representation For distributed representations, a single dimension is not responsible for explaining the factors of semantic change, and the semantics is entangled in multiple dimensions. As a result, distributed representations are difficult to interpret. As Bengio et al. [9] pointed out, a good distributed representation should "*disentangle the factors of variation*." There is always a desire for an interpretable distributed representation. Although PLSA and LDA have already increased interpretability, we would like to see more developments in this direction.

In this section, we will introduce the efforts that enhance the distributed word representations in terms of the above criteria.

2.4.1 Informative Word Representation

To make the representations informative, we can learn word representations from universal training data, including multilingual corpus. Another key direction for being informative is incorporating as much additional information into the representation as possible. From small to large information granularity, we can utilize character, morphological, syntactic, document, and knowledge base information. We will describe the related work in detail.

Multilingual Word Representation There are thousands of languages in the world. Making the vector space applicable for multiple languages not only improves the performance of word representation in low-resource languages but also can absorb information from the corpora of multiple languages. The bilingual word embedding model [78] proposes to make use of the word alignment pairs available in machine translation. It maps the embeddings of the source language to the embeddings of the target language and vice versa.

Specifically, a set of source words' representations that are trained on monolingual source language corpus is used to initialize the words in the target language:

$$\mathbf{w}_{\text{t-init}} = \sum_{s=1}^{S} \frac{N_{ts} + 1}{N_t + S} \mathbf{w}_s, \tag{2.33}$$

where $\mathbf{w}_s$ is the trained embeddings of the source word and $\mathbf{w}_{\text{t-init}}$ is the initial embedding of the target word, respectively. N_{ts} is the number of times that the target word t is aligned with the source word s. N_t is the total times of word t in the target corpus. The add-on terms $+1$ and $+S$ are the Laplace smoothing. S is the number of source words. Intuitively, the initialization of target word embedding is the weighted average of the aligned words in the source corpus, which ensure the two sets of embeddings are in the same space initially.

Then the source and target representation is optimized on their unlabeled corpora with target $\mathcal{L}_s$ and $\mathcal{L}_t$, respectively. To improve the alignment during training, alignment matrices $\mathbf{N}_{t\to s}$ and $\mathbf{N}_{s\to t}$ are used, where each element $\mathbf{N}_{ij}$ denotes the count of a word w_i is aligned with source word w_j normalized across all source words. Then a translation equivalence objective is used:

$$\begin{aligned} \mathcal{L}_{s\to t} &= \|\mathbf{w}_t - \mathbf{N}_{t\to s}\mathbf{w}_s\|_2^2, \\ \mathcal{L}_{t\to s} &= \|\mathbf{w}_s - \mathbf{N}_{s\to t}\mathbf{w}_t\|_2^2. \end{aligned} \tag{2.34}$$

Thus the unified objective becomes $\mathcal{L}_s + \lambda_1\mathcal{L}_{t\to s}\mathcal{L}_t + \lambda_2\mathcal{L}_{s\to t}$ for source words and target words, respectively, where λ_1 and λ_2 are the coefficients to weight the different sub-objectives.

However, this model performs poorly when the seed lexicon is small. Some works introduce virtual alignment between languages to tackle this limitation. Let's

take Zhang et al. [77] as an example. In addition to monolingual word embedding learning and bilingual word embedding alignment based on seed lexicon, this work proposes an integer latent variable vector $\mathbf{m} \in \mathbb{N}^{V^T}$ (V^T is the size of target vocabulary, and $\mathbb{N}$ is the set of natural numbers) representing which source word is linked by a target word w_t. So $\mathbf{m}_t \in \{0, 1, \ldots, V^S\}$. $\mathbf{m}$ is randomly initialized and then optimized through an expectation-maximization algorithm [18] together with the word representations. In the E-step, the algorithm fixes the current word representation and finds the best matching $\mathbf{m}$ that can align the source and target representations. And in the M-step, it treats the mapping as fixed and known, just like Zou et al. [78], and optimizes the source and target word representations.

Character-Enhanced Word Representation Many languages, such as Chinese and Japanese, have thousands of characters compared to other languages containing only dozens of characters. And the words in Chinese and Japanese are composed of several characters. Characters in these languages have richer semantic information. Hence, the meaning of a word can be learned not only from its context but also from the composition of characters. This intuitive idea drives Chen et al. [14] to propose a joint learning model for character and word embeddings (CWE). In CWE, a word representation $\mathbf{w}$ is a composition of the original word embedding $\mathbf{w}_0$ trained on corpus and its character embeddings $\mathbf{c}_i$. Formally,

$$\mathbf{w} = \mathbf{w}_0 + \frac{1}{|w|} \sum_i \mathbf{c}_i, \tag{2.35}$$

where $|w|$ is the number of characters in the word. Note that this model can be integrated with various models such as skip-gram, CBOW, and GloVe.

Further, position-based and cluster-based methods are proposed to address the issue that characters are highly ambiguous. In the position-based approach, each character is assigned three vectors that appear in *begin*, *middle*, and *end* of a word, respectively. Since the meaning of a character varies when it appears in the different positions of a word, this method can significantly resolve the ambiguity problem. However, characters that appear in the same position may also have different meanings. In the cluster-based method, a character is assigned K different vectors for its different meanings, in which a word's context is used to determine which vector to be used for the characters in this word.

Introducing character embeddings can significantly improve the representation of low-frequency words. Besides, this method can deal with new words while other methods fail. Experiments show that the joint learning method can perform better on both word similarity and analogy tasks.

Morphology-Enhanced Word Representation Many languages, such as English, have rich morphological information and plenty of rare words. However, most word representation models ignore the rich morphology information. This is a limitation because a word's affixes can help infer a word's meaning. Moreover, in traditional models, word representation is independent of each other. So when facing rare

words without enough context to learn the representations, the representations tend to be inaccurate.

Fortunately, in morphology-enhanced word representation, morphologies' representation can enrich word embeddings and are shared among words to assist the representation of rare words. Piotr et al. [11] propose to represent a word as a bag of morphology n-grams. This model substitutes word vectors in skip-gram with the sum of morphology n-gram vectors. When creating the dictionary of morphology n-grams, they select all morphology n-grams with a length greater or equal to 3 and smaller or equal to 6. To distinguish prefixes and suffixes from other affixes, they also add special characters to indicate the beginning and the end of a word. This model is efficient and straightforward, which achieves good performance on word similarity and word analogy tasks, especially when the training set is small. Ling et al. [44] further use a bidirectional LSTM to generate word representation by composing morphologies. This model significantly reduces the number of parameters since only the morphology representations and the weights of LSTM need to be stored.

Syntax-Enhanced Word Representation Continuous word embeddings should combine the semantic and syntactic information of words. However, existing word representation models depend solely on linear contexts and have more semantic information than syntactic information. To inject the embeddings with more syntactic information, the dependency-based word embedding [40] uses the dependency-based context. Dependency-based representation contains less topical information than the original skip-gram representation and shows more functional similarity. It considers the dependency parsing tree's information when learning word representations. The contexts of a target word w are the modifiers m_i of this word, i.e., $(m_1, r_1), \ldots, (m_k, r_k)$, where r_i is the type of dependency relation between the target node and the modifier. During training, the model optimizes the probability of dependency-based contexts rather than neighboring contexts. This model gains some improvements on word similarity benchmarks compared with skip-gram. Experiments also show that words with syntactic similarity are more similar in the vector space.

Document-Enhanced Word Representation Word embedding methods like skip-gram simply consider the context information within a window to learn word representation. However, the information in the whole document, e.g., the topics of the document, could also help word representation learning. Topical word embedding (TWE) [45] introduces topic information generated by latent Dirichlet allocation (LDA) to help distinguish different meanings of a word. The model is defined to minimize the following objective:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{i-l\le j\le i+l, j\neq i}\left(\log P(\mathbf{w}_j|\mathbf{w}_i) + \log P(\mathbf{w}_j|\mathbf{z}_i)\right), \tag{2.36}$$

where $\mathbf{w}_i$ is the word embedding and $\mathbf{z}_i$ is the topic embedding of w_i. Each word w_i is assigned a unique topic, and each topic has a topic embedding. The topical word embedding model shows the advantage of contextual word similarity and document classification tasks.

TopicVec [42] further improves the TWE model. TWE simply combines the LDA with word embeddings and lacks statistical foundations. Moreover, the LDA topic model needs numerous documents to learn semantically coherent topics. TopicVec encodes words and topics in the same semantic space. It can learn coherent topics when only one document is presented.

Knowledge-Enhanced Word Representation People have also annotated many knowledge bases that can be used in word representation learning as additional information. Yu et al. [76] introduce relational objectives into the CBOW model. With the objective, the embeddings can predict their contexts and words with relations. The objective is to minimize the sum of the negative log probability of all relations as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{w\in R_{w_i}} \log P(w|w_i), \tag{2.37}$$

where R_{w_i} indicates a set of words that have a relation with w_i. The external information helps train a better word representation, showing significant improvements in word similarity benchmarks.

Moreover, retrofitting [22] introduces a post-processing step that can introduce knowledge bases into word representation learning. It is more modular than other approaches which consider knowledge base during training. Let the word embeddings learned by existing word representation approaches be $\hat{\mathbf{W}}$. Retrofitting attempts to find a knowledgeable embedding space $\mathbf{W}$, which is close to $\hat{\mathbf{W}}$ but considers the relations in the knowledge base. It optimizes $\mathbf{W}$ toward $\hat{\mathbf{W}}$ and simultaneously shrinks the distance between knowledgeable representations of words w_i, w_j with relations. Formally,

$$\mathcal{L} = \sum_i \left(\alpha_i \|\mathbf{w}_i - \hat{\mathbf{w}}_i\|_2 + \sum_{(w_i,w_j)\in R} \beta_{ij}\|\mathbf{w}_i - \mathbf{w}_j\|_2\right), \tag{2.38}$$

where α and β are hyper-parameters indicating the strength of the associations, and R is a set of relations in the knowledge base. With knowledge bases such as the paraphrase database [28], WordNet [52], and FrameNet [3], this model can achieve consistent improvement on word similarity tasks. But it may also reduce the performance of the analogy of syntactic relations if it emphasizes semantic knowledge. Since it is a post-processing approach, it is compatible with various distributed representation models.

In addition to the aforementioned synonym-based knowledge bases, there are also sememe-based knowledge bases, in which the sememe is defined as the minimum semantic unit of word meanings. Due to the importance of sememe in computational linguistics, we introduce it in detail in Chap. 10.

2.4.2 Interpretable Word Representation

Although distributed word representation achieves ground-breaking performance on numerous tasks, it is less interpretable than conventional symbolic word representations. It would be a bonus if the distributed representations also enjoy some degree of interpretability. We can improve the interpretability from three directions. The first is to increase the interpretability of the vector representation among its neighbors. Since a word has multiple meanings, especially those polysemy words, the vectors of different meanings should locate in different neighborhoods. Therefore, we introduce work on disambiguated word representations. Another direction is to increase the interpretability of each dimension of the representation. A group of nonnegative and sparse word representations is shown to be well interpretable in each dimension. The third direction is to increase the interpretability of the embedding space by introducing more spatial properties in addition to the translational semantics in word2vec. In this section, we illustrate related work in these three directions.

Disambiguated Word Representation Using only one single vector to represent a word is problematic due to the ambiguity of words. A single vector that implies multiple meanings is naturally difficult to interpret, and distinguishing different meanings can lead to a more accurate representation.

In the multi-prototype vector space model, Banerjee et al. [5] use clustering algorithms to cluster different word meanings. Formally, it assigns a different word representation $\mathbf{w}_i(w_1)$ to the same word w_1 in each different cluster i. When the multi-prototype embedding is used, the similarity between two words w_1, w_2 is computed by comparing each pair of prototypes, i.e.,

$$\begin{aligned} \text{AvgSim}(w_1, w_2) &= \frac{1}{K^2}\sum_{i=1}^{K}\sum_{j=1}^{K} s(\mathbf{w}_i(w_1), \mathbf{w}_j(w_2)), \\ \text{MaxSim}(w_1, w_2) &= \max_{1\le i,j\le K} s(\mathbf{w}_i(w_1), \mathbf{w}_j(w_2)), \end{aligned} \tag{2.39}$$

where K is a hyper-parameter indicating the number of clusters and $s(\cdot)$ is a similarity function of two vectors, such as cosine similarity. When contexts are available, the similarity can be computed more precisely as

$$\begin{aligned} \text{AvgSimC}(w_1, w_2) &= \frac{1}{K^2}\sum_{i=1}^{K}\sum_{j=1}^{K} s_{c,w_1,i}\, s_{c,w_2,j}\, s(\mathbf{w}_i(w_1), \mathbf{w}_j(w_2)), \\ \text{MaxSimC}(w_1, w_2) &= s(\hat{\mathbf{w}}(w_1), \hat{\mathbf{w}}(w_2)), \end{aligned} \tag{2.40}$$

where $s_{c,w_1,i} = s(\mathbf{w}(c), \mathbf{w}_i(w_1))$ is the likelihood of context c belonging to cluster i, $\mathbf{w}(c)$ is the context representation, and $\hat{\mathbf{w}}(w_1) = \mathbf{w}_{\arg\max_{1\le i\le K} s_{c,w_1,i}}(w_1)$ is the maximum likelihood cluster for w_1 in context c. With multi-prototype

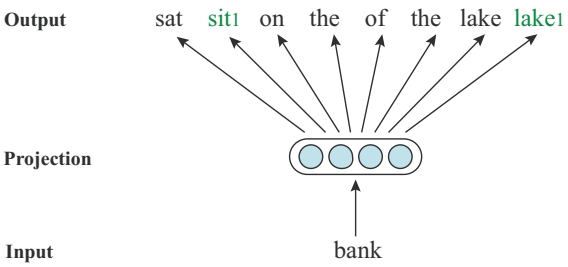


Fig. 2.9 The framework of Chen et al. [13]. We use the center word to predict both the context word and the context word's sense. The figure is redrawn according to Fig. 1 from Chen et al. [13]

embeddings, the accuracy of the word similarity task is significantly improved, but the performance is still sensitive to the number of clusters.

The multi-prototype embedding method can effectively cluster different meanings of a word via its contexts. However, the clustering is offline, and the number of clusters is fixed and needs to be pre-defined. It is difficult for a model to select an appropriate amount of meanings for different words; to adapt to new senses, new words, or new data; and to align the senses with prototypes. A unified model is proposed for both word representation and word sense disambiguation [13]. It uses available knowledge bases such as WordNet [52] to provide the list of possible senses of a word, perform the disambiguation based on the original word vectors, and update the word vectors and sense vectors. More specifically, as shown in Fig. 2.9, it first initalizes the word vectors to be the skip-gram vectors learned from large-scale corpora. And then, it aggregates the words in the definition of the sense (provided by knowledge bases) to form the sense initialization, where only the words in the definition whose similarity with the original word is larger than a threshold are considered. After initialization, it uses the sense vector to update the context vectors. For example, in the sentence "*He sat on the bank of the lake*," the "*bank*" which means "*the land alongside the lake*" is closer to the context vectors formed by words "*sat, bank, lake*," and then the representation of $bank_1$ is utilized to update the context vectors. The process is repeated for all words with multiple senses. After the disambiguation, a joint objective is used to optimize the senses and word vectors together

$$\mathcal{L} = -\sum_{i} \sum_{i-l \le j \le i+l, j \ne i} P(w_j|w_i) P(M(w_j)|w_i)), \tag{2.41}$$

where $M(w_j)$ is the disambiguated sense of word w_j and $2l+1$ is the window size. With the joint learning framework, not only performance on word representation learning tasks are enhanced, but the representations of concrete senses are more interpretable than the representations of polysemy words.

Nonnegative and Sparse Word Representation Another aspect of interpretability comes from the interpretability of each dimension of distributed word representations. Murphy et al. [53] introduce nonnegative and sparse embeddings (NNSE), where each dimension indicates a unique concept. This method factorizes the corpus

statistics matrix $\mathbf{M} \in \mathbb{R}^{|V| \times |D|}$ into a word embedding matrix $\mathbf{W} \in \mathbb{R}^{|V| \times m}$ and a document statistics matrix $\mathbf{D} \in \mathbb{R}^{m \times |D|}$, where $|V|$, $|D|$ and m are the vocabulary size, the number of documents, and the dimension of the distributed representation, respectively. Its training objective is

$$\begin{aligned} &\arg\min_{\mathbf{W},\mathbf{D}} \frac{1}{2} \sum_{i=1}^{|V|} \|\mathbf{M}_{i,:} - \mathbf{W}_{i,:}\mathbf{D}\|_2 + \lambda \|\mathbf{W}_{i,:}\|_1, \\ &\text{s.t. } \mathbf{D}_{i,:}\mathbf{D}_{i,:}^{\top} \leq 1, \text{ for } 1 \leq i \leq m, \\ &\quad \mathbf{W}_{i,j} \geq 0, \text{ for } 1 \leq i \leq |V|, 1 \leq j \leq m. \end{aligned} \tag{2.42}$$

The sparsity is ensured by $\lambda \|\mathbf{W}_{i,:}\|_1$, and non-negativity is guaranteed by $\mathbf{W}_{i,j} \geq 0$. By iteratively optimizing $\mathbf{W}$ and $\mathbf{D}$ via gradient descent, this model can learn nonnegative and sparse embeddings for words. Since embeddings are nonnegative, words with the highest scores on each dimension show high similarity to more words. Therefore, they can be regarded as superordinate concepts of more specific words. Again, since embeddings are sparse and only a few words correspond to each dimension, each dimension can be interpreted as the concept (word) with the highest value in that dimension.

A word intrusion task is designed to assess the interpretability of the word representation. For each dimension, we pick the N words with the largest value for that dimension as the positive words. Then we select the noisy words with the value of that dimension in the small half. Finally, we let human annotators pick out these noise words. The performance of the human annotators in all dimensions is the interpretability score of the model.

Fyshe et al. [27] further improve NNSE by enforcing the compositionality of the interpretable dimensions. For a phrase p composed of words w_i and w_j, the following constraint can be applied:

$$\mathbf{W}_{p,:} = f(\mathbf{W}_{i,:}, \mathbf{W}_{j,:}). \tag{2.43}$$

Therefore, the objective becomes

$$\begin{aligned} &\arg\min_{\mathbf{W},\mathbf{D}} \frac{1}{2} \sum_{i=1}^{|V|} \|\mathbf{M}_{i,:} - \mathbf{W}_{i,:}\mathbf{D}\|_2 + \lambda_1 \|\mathbf{W}_{i,:}\|_1 \\ &+ \lambda_2 \sum_{\text{phrase} p=(w_i,w_j)} (\mathbf{W}_{p,:} - f(\mathbf{W}_{i,:}, \mathbf{W}_{j,:}))^2, \end{aligned} \tag{2.44}$$

where λ_1 and λ_2 are the coefficients to weigh different sub-objectives.

The f has many choices. The authors define it to be a weighted addition between $\mathbf{W}_{i,:}$ and $\mathbf{W}_{j,:}$, i.e.,

$$\mathbf{W}_{p,:} = \alpha \mathbf{W}_{i,:} + \beta \mathbf{W}_{j,:}. \tag{2.45}$$

The resulting word representations are more interpretable since the multiple dimensions can form compositional meanings.

The above method applies to the matrix factorization paradigm, which encounters difficulty when the corpus and global co-occurrence is large. Can we apply the same nonnegative regularization for neural word representations such as word2vec? Luo et al. [47] present a nonnegative skip-gram model OIWE (online interpretable word embeddings), which adds constraints to the gradient descent process. Specifically, the update rule of the parameter is

$$\mathbf{w}_k \leftarrow \max(0, \mathbf{w}_k + \gamma \nabla \mathcal{L}(\mathbf{w}_k)), \tag{2.46}$$

where $\mathbf{w}$ is the word representation that needs to be updated, k is its k-th dimension, and γ is the learning rate.

However, directly using this update rule leads to unstable optimization because the update deviates from the true update too much. What we need is to make fewer dimensions of $\mathbf{w}_k + \gamma \nabla f(\mathbf{w}_k)$ less than 0. To achieve this goal, we use a dynamic learning rate. The learning rate is chosen to make the following violation ratio small:

$$R(\gamma) = \frac{|\{k | \mathbf{w}_k > 0, \mathbf{w}_k + \gamma \nabla \mathcal{L}(\mathbf{w}_k) < 0\}|}{K}, \tag{2.47}$$

where K is the number of dimensions.

The resulting word representation exceeds NNSE in both word similarity and word intrusion detection tasks.

Non-Euclidean Word Representation Interpretability also comes from an embedding space with comprehensible spatial properties. For example, the translation property of word2vec makes the difference between *male* and *female* interpretable (i.e., relation *gender*). Therefore, we are looking for more interpretable spatial properties. We introduce two special embedding spaces, i.e., Gaussian distribution space and hyperbolic space. Both of them enjoy hierarchical spatial properties that are understandable by humans. Vilnis et al. [74] propose to encode words into Gaussian distribution $N(x; \mu, \Sigma)$, where the mean μ of the Gaussian distribution is similar to traditional word embedding and the variance Σ becomes the uncertainty of the word meaning. The similarity between two representations can be defined either using asymmetric similarity (e.g., the KL divergence) or symmetric similarity (e.g., the continuous inner product between two Gaussian distributions):

$$\int_{x \in \mathbb{R}^n} N(x; \mu_i, \Sigma_i) N(x; \mu_j, \Sigma_j) dx = N(0; \mu_i - \mu_j, \Sigma_i + \Sigma_j), \tag{2.48}$$

where n is the dimension of vectors.

Note that the focus of Vilnis et al. [74] is on the uncertainty estimation of word meanings, which increases the interpretability of word meanings in terms of uncertainty estimation. But on the other hand, it is very easy to define entailment relations between two Gaussian embeddings of different sizes (variances), thus natural to encode the hierarchy into the representation, which increases the interpretability of the embedding in terms of ontology. This line of work further develops into representations based on the Gaussian mixture model [2] and elliptical word embedding [54].

Another line of work focuses on hyperbolic embeddings. Hyperbolic spaces $\mathbb{H}^n$ are spaces with constant negative curvature. The volume of the circle in hyperbolic space grows exponentially with radius. This property makes it suitable for encoding the tree structure, where the number of nodes grows exponentially with depth. Hence, it is a suitable space for encoding hierarchical structures. For example, Nickel et al. [55] use a special hyperbolic space, namely, Poincaré Ball, as the embedding space. They propose to encode word relations such as those explicitly given by the hierarchy in WordNet using supervised learning. A subsequent work [72] successfully applies Poincaré embeddings in a completely unsupervised manner. Specifically, they propose Poincaré GloVe, a modified target of GloVe, for encoding hyperbolic geometry. Considering the GloVe target in Eq. (2.26), it can be generalized into more general metrics as follows:

$$\begin{aligned} d(\mathbf{w}_i, \mathbf{w}_j, \mathbf{M}_{ij}) &= (\mathbf{w}_i \mathbf{w}_j^\top + b_i + b_j - \log \mathbf{M}_{ij})^2 \\ &= (-\frac{1}{2}\|\mathbf{w}_i - \mathbf{w}_j\|_2^2 + b_i' + b'_j - \log \mathbf{M}_{ij})^2 \\ &= (-\frac{1}{2} h_{\text{Euclidean}}(\|\mathbf{w}_i - \mathbf{w}_j\|_2) + b_i' + b'_j - \log \mathbf{M}_{ij})^2, \end{aligned} \tag{2.49}$$

where $\mathbf{M}_{ij}$ is the global co-occurrence matrix and $h_{\text{Euclidean}} = (\cdot)^2$ is the metric for accessing the similarity of the two embeddings. Now it can be substituted with $h_{\text{hyperbolic}} = \cosh^2(\cdot)$. The embedding is optimized using Riemannian optimization [8]. The use of this word vector for inference (e.g., word analogy tasks) requires using the corresponding hyperbolic space operators, which we omit the details.

In summary, encoding more information and improving interpretability have been pursued by researchers. With such efforts, word representations have become the basis of modern NLP and are widely used in many practical tasks.

2.5 Applications

Word representation, as a milestone breakthrough in NLP, has not only spawned subsequent work in NLP itself but has also been widely applied in other disciplines, catalyzing many highly influential interdisciplinary works. Therefore, in this

section, we first introduce the applications of NLP itself, and then we introduce interdisciplinary works such as in psychology, history, and social science.

2.5.1 NLP

In the early stages of the introduction of neural networks into NLP, research on the application of word representations was very vigorous. For example, word representations are helpful in word-level tasks such as word similarity, word analogy, and ontology construction. They can also be applied to simple higher-level downstream tasks such as sentiment analysis. The performance of word representations on these tasks can measure the quality of word representations, so they can also be considered as *evaluation tasks of word representations*. Next, we introduce word similarity, word analogy, ontology construction, and sentence-level tasks.

Word Similarity and Relatedness Word similarity and relatedness both measure how close a word is to another word. Similarity means that the two words express similar meanings. And relatedness refers to a close linguistic relationship between the two words. Words that are not semantically similar could still be related in many ways, such as meronymy (*car* and *wheel*) or antonymy (*hot* and *cold*).

To measure word similarity and relatedness, researchers collect a set of word pairs and compute the correlation between human judgment and predictions made by word representations. So far, many datasets have been collected and made public. Some datasets focus on word similarity, such as RG-65 [62] and SimLex-999 [32]. Other datasets concern word relatedness, such as MTurk [60]. WordSim-353 [23] is a very popular dataset for word representation evaluation, but its annotation guideline does not differentiate similarity and relatedness. Agirre et al. [1] conduct another round of annotation based on WordSim-353 and generate two subsets, one for similarity and the other for relatedness. We summarize some information about these datasets in Table 2.1.

After collecting the datasets, quantitative evaluations of the word representations for the datasets are needed. Researchers usually select cosine similarity as the metric. After that, Spearman's correlation coefficient ρ is then used to evaluate

Table 2.1 Datasets for evaluating word similarity/relatedness

Dataset	Similarity type
RG-65 [62]	Word similarity
WordSim-353 [23]	Word similarity and relatedness
WordSim-353 REL [1]	Word relatedness
WordSim-353 SIM [1]	Word similarity
MTurk-287 [60]	Word relatedness
SimLex-999 [32]	Word similarity

the coherence between human annotators and the word representation. A higher Spearman's correlation coefficient indicates they are more similar.

Given the datasets and evaluation metrics, different kinds of word representations can be compared. Agirre et al. [1] address different word representations with different advantages. They point out that linguistic KB-based methods perform better on similarity than on relatedness, while distributed word representation shows similar performance on both. With the development of distributed representations, a study [68] in 2015 compares a series of word representations on a wide variety of datasets and gives the conclusion that distributed representations achieve state of the art in both similarity and relatedness.

Besides evaluation with deliberately collected datasets, the word similarity measurement can come in an alternative format, the TOEFL synonyms test. In this test, a cue word is given, and the test is required to choose one from four words that are the synonym of the cue word. The exciting part of this task is that the performance of a system could be compared with human beings. Landauer et al. [39] evaluate the system with the TOEFL synonyms test to address the knowledge inquiring and representing of LSA. The reported score is 64.4%, which is very close to the average rating of the human test-takers. On this test set with 80 queries, Sahlgren et al. [63] report a score of 72.0%. Freitag et al. [25] extend the original dataset with the help of WordNet and generate a new dataset (named as WordNet-based synonymy test) containing thousands of queries.

Word Analogy Besides word similarity, the word analogy task is an interesting task to infer words and serves as an alternative way to measure the quality of word representations. This task gives three words w_1, w_2, and w_3, and then it requires the system to predict a word w_4 such that the relation between w_1 and w_2 is the same as that between w_3 and w_4. This task has been used since the proposal of word2vec [49, 51] to exploit the structural relations among words. Here, word relations can be divided into two categories, including semantic relations and syntactic relations. Word analogy quickly becomes a standard evaluation metric once the dataset is released. Unlike the TOEFL synonyms test, the prediction is chosen from the whole vocabulary instead of the provided options. This test favors distributed word representations because it emphasizes the structure of word space. The comparison between different models on the word analogy task measured by accuracy could be found in [7, 68, 70, 75].

Ontology Construction Another usage of word representation is to construct the ontology knowledge bases. Section 2.4.1 talked about injecting knowledge base information into word representations. But conversely, learned word embeddings also help build the knowledge base. Since word representation is better at common words than rare words, word representations are more suitable for building ontology graphs[10] than building factual knowledge graphs.

[10] Ontology graph connects different abstract concepts in a graph according to their semantic relationships.

In ontologies, perhaps the most important relation is the `is_a` relation. Traditional word2vec models are good at expressing analogous relations, such as *man-woman* $\approx$ *king-queen* but not good at hierarchical relations, such as *mammal-cat* $\not\approx$ *celestial body-sun*. To model such relationships, Fu et al. [26] propose to use a linear projection rather than a simple embedding offset to represent the relationship. The model optimizes the projection as

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \sum_{i,j} \|\mathbf{w}_i \mathbf{W} - \mathbf{w}_j\|_2, \tag{2.50}$$

where $\mathbf{w}_i$ and $\mathbf{w}_j$ are hypernym and hyponym embeddings and $\mathbf{W}$ is the transformation matrix.

The non-Euclidean word representations introduced in Sect. 2.4.2 also help build the ontology network. Another knowledge base that word embedding can help is the sememe knowledge introduced in Chap. 10.

Sentence-Level Tasks Besides word-level tasks, word representations can also be used alone in some simple sentence-level tasks. However, word representations trained under purely co-occurrence objectives may not be optimal for a given task, and we can include task-relevant objectives in training. Take sentiment analysis as an example. Most word representation methods capture syntactic and semantic information while ignoring the sentiment of the text. This is questionable because words with similar syntactic polarity but opposite sentiment polarity obtain close word vectors. Tang et al. [71] propose to learn sentiment-specific word embeddings (SSWE). An intuitive idea is to jointly optimize the sentiment classification model using word embeddings as its feature, and SSWE minimizes the cross entropy loss to achieve this goal. To better combine the unsupervised word embedding method and the supervised discriminative model, they further use the words in a window rather than a whole sentence to classify sentiment polarity. To get massive training data, they use distant supervision to generate sentiment labels for a document. On sentiment classification tasks, sentiment embeddings outperform other strong baselines, including SVM [15] and other word embedding methods. SSWE also shows strong polarity consistency, where the closest words of a word are more likely to have the same sentiment polarity compared with existing word representation models. This sentiment-specific word embedding method provides us with a general way to learn task-specific word embeddings, which is to design a joint loss function and generate massive labeled data automatically.

Interestingly, as subsequent research in NLP progressed, including the development of sentence representations (Chap. 4) and the introduction of pre-trained models (Chap. 5), simple word vectors gradually ceased to be used alone. We point out the following reasons for this:

- High-level (e.g., sentence level) semantic units require combinations between words, and simple arithmetic operations between word representations are not sufficient to model high-level semantic models.

- Most word representation models do not consider word order and cannot model utterance probabilities, much less generate language.

We recommend that readers continue reading subsequent chapters to become familiar with more advanced methods.

2.5.2 Cognitive Psychology

In cognitive psychology, a famous behavioral test examines the correlation in the subconscious mind, named the implicit association test (IAT) [30]. This test is widely used to detect biases, such as gender biases, religion biases, occupation biases, etc.

IAT is based on a hypothesis. The hypothesis says that people's reaction time decreases when faced with similar concepts and increases when faced with conflicting concepts. For example, given *target words* (*woman*, *man*) and *attribute words* (*beautiful*, *strong*), we want to test a subject's perspective: which attribute is associated more closely with each target. If a person believes that *woman* and *beautiful* are close and *man* and *strong* are close, then, when faced with category *A* (*woman*, *beautiful*) and category *B* (*man*, *strong*), he/she will quickly categorize the word *charming* into the former group. Whereas if he/she is faced with two categories *A* (*woman*, *strong*) and *B* (*man*, *beautiful*), then faced with the word *charming*, he/she will hesitate between the two pairs of words, thus increasing the reaction time substantially, although the correct answer is clear that *charming* should be grouped into *B* since it's an attribute word, and thus should be classified according to *beautiful* or *strong*. A part of an IAT is shown in Fig. 2.10.

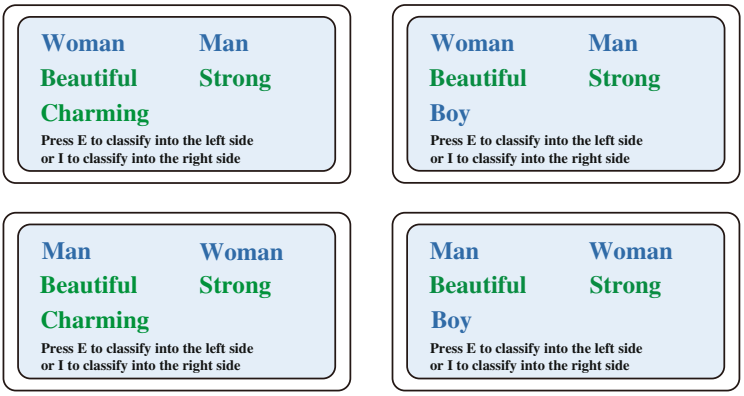


Fig. 2.10 Parts of the IAT. The box in the second row is considered to be harder than the first row for people who have an implicit association between *women* and *beautiful* (*man* and *strong*)

IAT can detect implicit thoughts and biases in the human mind. Considering that bias is so prevalent in people's perceptions, it will likely be reflected in the texts written by humans. A pioneering article [12] in Science magazine proposes WEAT (word-embedding association test) to detect bias in texts. Given two sets X, Y as the target words and A, B as two sets of attribute words, WEAT defines a difference between the target sets to the two sets of attribute words as follows:

$$s(X, Y; A, B) = \sum_{x \in X} s(x; A, B) - \sum_{y \in Y} s(y; A, B), \tag{2.51}$$

and

$$s(x; A, B) = \text{mean}_{a \in A} \cos(\mathbf{x}, \mathbf{a}) - \text{mean}_{a \in B} \cos(\mathbf{x}, \mathbf{b}), \tag{2.52}$$

where $\mathbf{x}, \mathbf{y}, \mathbf{a}, \mathbf{b}$ are the vector representation of word x, y, a, b. $\cos(\mathbf{x}, \mathbf{a})$ that can be treated as the response time in the IAT. And $\text{mean}(\cdot)$ is the average function. Thus, $s(x; A, B)$ measures the closeness of x to two sets of attributes. And $s(X, Y; A, B)$ measures the bias difference between X and Y to A and B. If X, Y are not biased differently for A and B, then they should not exceed at least the bias difference of (X_i, Y_i), where $X \cap Y$ is randomly divided into two sets X_i, Y_i of equal size. Hence, we test whether the following metric is small:

$$P[s(X_i, Y_i; A, B) > s(X, Y; A, B)]. \tag{2.53}$$

After some statistical derivations, we are able to calculate the above probabilities. In their experiments, WEAT is capable of capturing the occupational gender bias, where the occupational gender association calculated from the word representation is highly correlated with the publicly available proportion of female workers in each industry. For the name gender association, WEAT can find a similar pattern.

2.5.3 History and Social Science

It is possible to detect human thoughts without conducting live experiments using tests built on texts. This makes it very helpful to study the thoughts of ancient people. That is, we can explore the thoughts of the ancients through the texts they wrote, and this is precisely the important role of word representations in history and social sciences. In this section, we talk about how to use word representations to study the changes in history and society across time.

In order to track the chronological changes in word meanings, we first need to have a corpus of different chronologies. Google NGram Book Corpus [43] is a relatively early chronological corpus. This dataset counts the words/phrases' frequency used during the last five centuries and includes 6% of all published books, which is a very large dataset. Another COHA dataset [16] has 400 million

words, documenting the development of American English between 1810 and 2009, contains a wide variety of genres, and is relatively balanced across genres.

The work on tracking word sense changes [31, 37] divides these datasets into bins of equal size by time. They then train word representation models on the text within each period. For example, in the work [31], the SVD decomposition of the PPMI (positive pointwise mutual information) matrix and SGNS (skip-gram with negative sampling) are used as two base models. As mentioned earlier, the dimensions are not aligned for different groups of distributed representations, even if they are derived from the same counting matrix. Therefore, the authors propose to optimize a mapping matrix that maps the word vectors of the previous period to the word vector space of the latter period.

Aligning the word vectors after training them usually does not yield satisfying results because simple transformations cannot always align the two vector spaces, and complex transformations carry the risk of overfitting. **Time-sensitive word representation** is developed to address these issues. Bamler et al. [4] propose a dynamic skip-gram model which connects several Bayesian skip-gram models [6] using Kalman filters [35]. In this model, the embeddings of words in different periods could affect each other. For example, a word that appears in the 1990s document can affect the embeddings of that word in the 1980s and 2000s. Moreover, this model puts all the embeddings into the same semantic space, significantly improving against other methods and making word embeddings in different periods comparable. Experimental results show that the cosine distance between two words changes much more smoothly in this model than in those that simply divide the corpus into bins.

We can arrive at interesting *societal observations* using word representations from different periods. Hamilton et al. [31] perform two analyses, the first of which computes the time series formed by the cosine value of a word pair over time. They use Spearman correlation coefficients of the time series against time to estimate whether this change is an upward or downward trend and how significant the trend is. For the second analysis, the authors track the degree of change in the word vector of the same word over time to see the semantic drifts of a word across periods. They have come to some interesting conclusions. (1) Some established word sense shifts can be confirmed from the corpus. For example, the shift of *gay* change from *happy* to *homosexual* is observed from the word representation. (2) The authors also find the ten words that changed the most from 1900 to 1990. (3) Combining some experimental observations, the authors found two statistical rules of semantic variation. The first is that common words change their meanings more slowly, and rare words change their meanings more quickly. The second is that words with multiple meanings change their meanings more quickly.

The follow-up work [29] is based on the same diachronic word vectors as Hamilton et al. [31] but makes some observations with more depth in social science. Specifically, it compares trends in gender and race stereotypes over 100 years. To get the stereotype information from word representations, this article computes the difference in the association scores of an attribute word (e.g., *intelligent*) to two groups of words (e.g., *woman*, *female* versus *man*, *male*). The association score

can be calculated by either cosine similarity or Euclidean distance. This is similar to the WEAT mentioned in Sect. 2.5.2. The work further compares the association score with publicly available data about the gender per occupation statistics over the year. They find the two trends match almost exactly. When studying the association of adjectives to genders, a clear phase shift is found. The similarity of adjectives' association scores to genders is similar within the 1910s∼1960s and the 1960s∼1990s, respectively, but differs substantially between the two time periods. The phase shift in the 1960s corresponds to the US women's movement in history.

2.6 Summary and Further Readings

In this chapter, we focus on words, which are the basic semantic units. We introduce the representative methods in symbolic representation and distributed representation. Some well-known models are introduced, such as one-hot representation, LSA, PLSA, LDA, word2vec, GloVe, and ELMo. We also present the methods to make the representations more informative and interpretable. At last, the applications of word representations are introduced, where interdisciplinary applications are emphasized. For further readings, firstly, we encourage the readers to read Chaps. 3 and 4 for higher-level representations as they are more widely used in practical tasks. We also encourage the readers to review historical research, such as the review paper on representation learning by Yoshua Bengio [9], and the review paper on pre-trained distributed word representations by Tomas Mikolov, the author of word2vec [50].

Acknowledgments Zhiyuan Liu, Yankai Lin, and Maosong Sun designed the overall architecture of this chapter; Shengding Hu drafted this chapter. Zhiyuan Liu and Yankai Lin proofread and revised this chapter.

We also thank Ning Ding, Yujia Qin, Si Sun, Yusheng Su, Zhitong Wang, Xingyu Shen, Zheni Zeng, and Ganqu Cui for proofreading the chapter and Lei Xu for preparing the initial draft materials for the first edition.

This is the word representation learning chapter about of the second edition of the book *Representation Learning for Natural Language Processing*, with its first edition published in 2020 [46]. Compared to the first edition of this chapter, the main changes include the following: (1) we rewrote the sections before Sect. 2.4 by systematically restructuring the works, (2) we restructured and summarized the advanced topics into two directions and polish the writing of advanced topics, and (3) we added a new section to introduce word representation's applications.

References

1. Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. A study on similarity and relatedness using distributional and WordNet-based approaches. In *Proceedings of NAACL-HLT*, 2009.
2. Ben Athiwaratkun and Andrew Wilson. Multimodal word distributions. In *Proceedings of ACL*, 2017.

3. Collin F Baker, Charles J Fillmore, and John B Lowe. The berkeley framenet project. In *Proceedings of ACL*, 1998.
4. Robert Bamler and Stephan Mandt. Dynamic word embeddings via skip-gram filtering. *arXiv preprint arXiv:1702.08359*, 2017.
5. Arindam Banerjee, Inderjit S Dhillon, Joydeep Ghosh, and Suvrit Sra. Clustering on the unit hypersphere using von Mises-Fisher distributions. *Journal of Machine Learning Research*, 2005.
6. Oren Barkan. Bayesian neural word embedding. In *Proceedings of AAAI*, 2017.
7. Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of ACL*, 2014.
8. Gary Bécigneul and Octavian-Eugen Ganea. Riemannian adaptive optimization methods. In *Proceedings of ICLR*, 2019.
9. Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.
10. David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. In *Proceedings of NeurIPS*, 2001.
11. Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 2017.
12. Aylin Caliskan, Joanna J Bryson, and Arvind Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 2017.
13. Xinxiong Chen, Zhiyuan Liu, and Maosong Sun. A unified model for word sense representation and disambiguation. In *Proceedings of EMNLP*, 2014.
14. Xinxiong Chen, Lei Xu, Zhiyuan Liu, Maosong Sun, and Huanbo Luan. Joint learning of character and word embeddings. In *Proceedings of IJCAI*, 2015.
15. Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 1995.
16. Mark Davies. The corpus of historical American english: 400 million words, (1810-2009), 2010.
17. Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 1990.
18. Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1977.
19. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.
20. Chris Ding, Tao Li, and Wei Peng. On the equivalence between non-negative matrix factorization and probabilistic latent semantic indexing. *Computational Statistics & Data Analysis*, 2008.
21. Katrin Erk and Sebastian Padó. A structured vector space model for word meaning in context. In *Proceedings of EMNLP*, 2008.
22. Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A Smith. Retrofitting word vectors to semantic lexicons. In *Proceedings of NAACL-HLT*, 2015.
23. Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. In *Proceedings of WWW*, 2001.
24. John R Firth. A synopsis of linguistic theory, 1930-1955. *Studies in Linguistic Analysis*, 1957.
25. Dayne Freitag, Matthias Blume, John Byrnes, Edmond Chow, Sadik Kapadia, Richard Rohwer, and Zhiqiang Wang. New experiments in distributional representations of synonymy. In *Proceedings of CoNLL*, 2005.
26. Ruiji Fu, Jiang Guo, Bing Qin, Wanxiang Che, Haifeng Wang, and Ting Liu. Learning semantic hierarchies via word embeddings. In *Proceedings of ACL*, 2014.

27. Alona Fyshe, Leila Wehbe, Partha Pratim Talukdar, Brian Murphy, and Tom M Mitchell. A compositional and interpretable semantic space. In *Proceedings of NAACL-HLT*, 2015.
28. Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. PPDB: The paraphrase database. In *Proceedings of NAACL-HLT*, 2013.
29. Nikhil Garg, Londa Schiebinger, Dan Jurafsky, and James Zou. Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, 2018.
30. Anthony G Greenwald and Shelly D Farnham. Using the implicit association test to measure self-esteem and self-concept. *Journal of Personality and Social Psychology*, 2000.
31. William L. Hamilton, Jure Leskovec, and Dan Jurafsky. Diachronic word embeddings reveal statistical laws of semantic change. In *Proceedings of ACL*, 2016.
32. Felix Hill, Roi Reichart, and Anna Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 2015.
33. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.
34. Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of SIGIR*, 1999.
35. Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 1960.
36. Pentti Kanerva, Jan Kristofersson, and Anders Holst. Random indexing of text samples for latent semantic analysis. In *Proceedings of CogSci*, 2000.
37. Yoon Kim, Yi-I Chiu, Kentaro Hanaki, Darshan Hegde, and Slav Petrov. Temporal analysis of language through neural language models. In *Proceedings of ACL Workshop*, 2014.
38. Da Kuang, Jaegul Choo, and Haesun Park. Nonnegative matrix factorization for interactive topic modeling and document clustering. *Partitional Clustering Algorithms*, page 215, 2014.
39. Thomas K Landauer and Susan T Dumais. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 1997.
40. Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of ACL*, 2014.
41. Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Proceedings of NeurIPS*, 2014.
42. Shaohua Li, Tat-Seng Chua, Jun Zhu, and Chunyan Miao. Generative topic embedding: a continuous representation of documents. In *Proceedings of ACL*, 2016.
43. Yuri Lin, Jean-Baptiste Michel, Erez Aiden Lieberman, Jon Orwant, Will Brockman, and Slav Petrov. Syntactic annotations for the Google Books NGram corpus. In *Proceedings of the ACL*, 2012.
44. Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramón Fermandez, Silvio Amir, Luis Marujo, and Tiago Luís. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of EMNLP*, 2015.
45. Yang Liu, Zhiyuan Liu, Tat-Seng Chua, and Maosong Sun. Topical word embeddings. In *Proceedings of AAAI*, 2015.
46. Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation Learning for Natural Language Processing*. Springer, 2020.
47. Hongyin Luo, Zhiyuan Liu, Huanbo Luan, and Maosong Sun. Online learning of interpretable word embeddings. In *Proceedings of EMNLP*, 2015.
48. T Mikolov and J Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of NeurIPS*, 2013.
49. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of ICLR*, 2013.
50. Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of LREC*, 2018.
51. Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of NAACL-HLT*, 2013.
52. George A Miller. WordNet: a lexical database for English. *Communications of the ACM*, 1995.

53. Brian Murphy, Partha Talukdar, and Tom Mitchell. Learning effective and interpretable semantic models using non-negative sparse embedding. In *Proceedings of COLING*, 2012.
54. Boris Muzellec and Marco Cuturi. Generalizing point embeddings using the Wasserstein space of elliptical distributions. In *Proceedings of NeurIPS*, 2018.
55. Maximillian Nickel and Douwe Kiela. Poincare embeddings for learning hierarchical representations. In *Proceedings of NeurIPS*, 2017.
56. Sebastian Padó and Mirella Lapata. Dependency-based construction of semantic space models. *Computational Linguistics*, 2007.
57. Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of EMNLP*, 2014.
58. Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, 2018.
59. Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. 2018.
60. Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. A word at a time: computing word relatedness using temporal semantic analysis. In *Proceedings of WWW*, 2011.
61. Philip Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of artificial intelligence research*, 1999.
62. Herbert Rubenstein and John B Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 1965.
63. Magnus Sahlgren. Vector-based semantic analysis: Representing word meanings based on random labels. In *Proceedings of SKAC Workshop*, 2001.
64. Magnus Sahlgren. An introduction to random indexing. In *Proceedings of TKE*, 2005.
65. Magnus Sahlgren. *The Word-Space Model: Using Distributional Analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces*. PhD thesis, Institutionen för lingvistik, 2006.
66. Gerard Salton. *The SMART retrieval system—experiments in automatic document processing*. Prentice-Hall, Inc., 1971.
67. Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 1975.
68. Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of EMNLP*, 2015.
69. Angus Stevenson. *Oxford dictionary of English*. 2010.
70. Fei Sun, Jiafeng Guo, Yanyan Lan, Jun Xu, and Xueqi Cheng. Learning word representations by jointly modeling syntagmatic and paradigmatic relations. In *Proceedings of ACL*, 2015.
71. Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning sentiment-specific word embedding for Twitter sentiment classification. In *Proceedings of ACL*, 2014.
72. Alexandru Tifrea, Gary Bécigneul, and Octavian-Eugen Ganea. Poincaré glove: Hyperbolic word embeddings. *arXiv preprint arXiv:1810.06546*, 2018.
73. Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 2008.
74. Luke Vilnis and Andrew McCallum. Word representations via Gaussian embedding. In *Proceedings of ICLR*, 2015.
75. Dani Yogatama, Manaal Faruqui, Chris Dyer, and Noah A Smith. Learning word representations with hierarchical sparse coding. In *Proceedings of ICML*, 2015.
76. Mo Yu and Mark Dredze. Improving lexical embeddings with semantic knowledge. In *Proceedings of ACL*, 2014.
77. Meng Zhang, Haoruo Peng, Yang Liu, Huan-Bo Luan, and Maosong Sun. Bilingual lexicon induction from non-parallel data with minimal supervision. In *Proceedings of AAAI*, 2017.
78. Will Y Zou, Richard Socher, Daniel M Cer, and Christopher D Manning. Bilingual word embeddings for phrase-based machine translation. In *Proceedings of EMNLP*, 2013.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Chapter 3
Representation Learning for Compositional Semantics

Ning Ding, Yankai Lin, Zhiyuan Liu, and Maosong Sun

Abstract Many important applications in NLP fields rely on understanding complex language units composed of words, such as phrases, sentences, and documents. A key problem is semantic composition, i.e., how to represent the semantic meanings of complex language units by composing the semantic meanings of those words in them. Semantic composition is a much more complicated process than a simple sum of the parts, and compositional semantics remains a core task in computational linguistics and NLP. In this chapter, we first introduce representative approaches for binary semantic composition, including additive models and multiplicative models, to demonstrate the nature of natural languages. After that, we present typical modeling methods for N-ary semantic composition, including sequential order, recursive order, and convolutional order methods, which follows a similar track for sentence and document representation learning and will be introduced in detail in the next chapter.

3.1 Introduction

Following the distributional hypothesis, one could project the semantic meaning of a word into a low-dimensional real-valued vector according to its context information. Here comes a further problem: how to compress a higher semantic unit, such as a phrase, into a vector or other kinds of mathematical representations like a matrix or a tensor? In this chapter, we introduce the representation learning approach to model semantic composition functions from the linguistic perspective.

Compositionality enables natural languages to construct complicated semantic meanings from the combinations of basic semantic elements with particular rules.

N. Ding · Z. Liu (✉) · M. Sun
Department of Computer Science and Technology, Tsinghua University, Beijing, China
e-mail: dingn18@mails.tsinghua.edu.cn; liuzy@tsinghua.edu.cn; sms@tsinghua.edu.cn

Y. Lin
Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China
e-mail: yankailin@ruc.edu.cn

© The Author(s) 2023

Z. Liu et al. (eds.), *Representation Learning for Natural Language Processing*,
https://doi.org/10.1007/978-981-99-1600-9_3

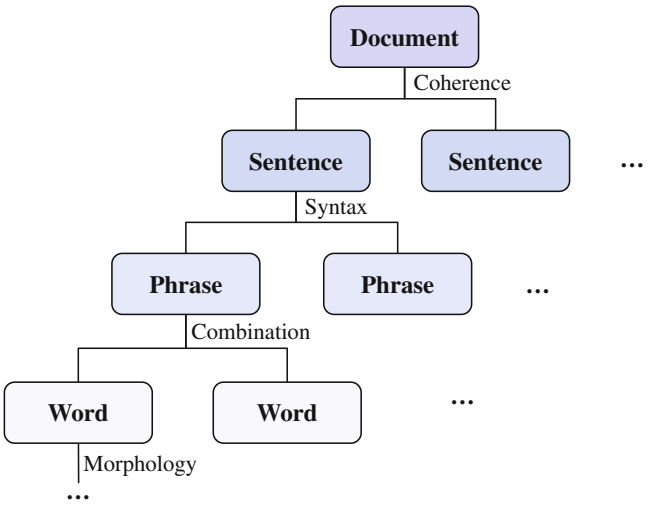


Fig. 3.1 Higher-level linguistic units are composed of basic linguistic units guided by certain rules

The basic principle is the semantic meaning of a whole is a function of the semantic meanings of its several parts. Therefore, the semantic meanings of complex structures will depend on how their semantic elements combine. There are many previous studies dedicated to the representation learning of compositional semantics. Among them, the article *composition in distributional models of semantics* [7] proposes a comprehensive framework of compositional semantics and becomes a rather representative summarization of this line of work. In this chapter, we adopt the framework to introduce compositional semantics along with our understanding and discussion (Fig. 3.1).

Here, we consider two basic semantic units and use $\mathbf{u}$ and $\mathbf{v}$ to denote them, respectively. And the most intuitive way to define the joint representation could be formulated by directly building a mapping function:

$$\mathbf{p} = f(\mathbf{u}, \mathbf{v}), \tag{3.1}$$

where $\mathbf{p}$ corresponds to the representation of the joint semantic unit $(\mathbf{u}, \mathbf{v})$. Generally, $\mathbf{u}$ and $\mathbf{v}$ could denote words, phrases, sentences, paragraphs, or even higher-level semantic units.

However, given the representations of two semantic constituents, it is not enough to derive their joint embedding without syntactic information. For instance, although the phrase *machine learning* and *learning machine* have the same vocabulary, they contain different meanings: *machine learning* refers to a research field in artificial intelligence, while *learning machine* means some specific learning algorithms. That is to say, the way how the units are mixed is also an essential part of the semantic composition. This phenomenon stresses the importance of syntactic and order

information in a compositional unit. Hence, an improved version of the framework is to take the role of syntactic and order information into consideration [9]. Specifically, in terms of formulation, we can introduce an $\mathcal{R}$ to represent the relationship between units.

The complex semantics of a combined unit in the real world can also be influenced by human background knowledge. In other words, some sentences are difficult to understand by merely paying attention to the constituent units and the syntax. For example, the sentence *Tom and Jerry is one of the most popular comedies in that style.* needs two main backgrounds: firstly, *Tom and Jerry* is a special noun phrase or knowledge entity that indicates a cartoon comedy, rather than two ordinary people. The other prior knowledge should be *that style*, which needs further explanation in the previous sentences. Hence, a full understanding of compositional semantics needs to take external knowledge into account. We consummate the formulation by adding another item $\mathcal{K}$ to denote such background knowledge. The complete composition function in Eq. (3.1) is redefined to combine the syntactic relationship rule $\mathcal{R}$ between the semantic units $\mathbf{u}$ and $\mathbf{v}$ and human background knowledge $\mathcal{K}$:

$$\mathbf{p} = f(\mathbf{u}, \mathbf{v}, \mathcal{R}, \mathcal{K}). \tag{3.2}$$

From the perspective of computational linguistics, the meaning of a word is not isolated from the context. That is, the semantics of the combined unit are constituted from its components, and the meanings of the components are meanwhile derived from the combined unit [3]. This also echoes the concept of structuralism stated in Chap. 1. Compositionality is also a matter of degree instead of an either-or issue [8]. We could divide linguistic structures into several groups according to the degree of compositionality. For example, the fully compositional structure means that the combined high-level semantics is completely composed of the independent semantics of basic units (e.g., white fur). In partly compositional expressions, basic units still have separate meanings, but when combined together, they derive additional semantics (e.g., take care). In non-compositional idioms or multi-word expressions, the combined semantics have little relationship with the semantics of basic units (e.g., at large).

From the above equations formulating composition function, it could be concluded that composition could be viewed as more than a specific binary operation. The syntactic information could help to indicate a particular approach, while background knowledge helps to explain some obscure words or specific context-dependent entities such as pronouns. To this end, we could realize that the complexity of language comes from the nearly infinite combination of finite elements. This chapter can be seen as the transition from word representation to sentence and document representation, aiming to introduce the basic concepts and methods for dealing with compositional semantics from a linguistic point of view. We will first explain basic binary composition functions in Sect. 3.2, including the additive model and multiplicative model and then give a brief introduction to modeling methods for more complex N-ary composition in Sect. 3.3, including

sequential order, recursive order, and convolutional order modeling. We will introduce the specific methods of learning sentence and document representation in modern NLP in detail in the next chapter.

3.2 Binary Composition

The goal of compositional semantics is to construct vector representations for higher-level linguistic units with basic units via binary composition. Without loss of generality, we assume that each constituent of a phrase (or even higher-level linguistic units) is embedded into a computable vector which will be further used to generate a representation vector for the phrase.

In this section, we focus on binary composition, where two objects will be involved in each operation. We now consider phrases consisting of two components: a head and a modifier or complement. If we cannot model the binary composition (or phrase representation), it is almost impossible to build more complex compositional representations for higher-level linguistic units. Even in today's age of neural networks, the concept of binary composition is still important. For example, in the transformer architecture, the calculation of the attention of two units could be regarded as a type of binary operation.

Given a phrase with two constituent words *machine learning*, as well as the representations $\mathbf{u}$ and $\mathbf{v}$ representing the words *machine* and *learning*, respectively, our primary goal is to construct a representation vector $\mathbf{p}$ of the phrase according to the representations of the words. With a simple semantic space where each vector is represented by five integers, we let the hypothetical vectors for *machine* and *learning* be $[0, 3, 1, 5, 2]$ and $[1, 4, 2, 2, 0]$, respectively. And if we simply use the add operator to represent the phrase *machine learning*, it becomes $[0, 3, 1, 5, 2] + [1, 4, 2, 2, 0] = [1, 7, 3, 7, 2]$. The key to this problem is designing a primitive composition function as a binary operator. Based on this function, one could apply it to a word sequence recursively to derive composition for longer text.

Modeling the binary composition function is a well-studied but still challenging problem. There are mainly two perspectives on this question, including the additive model and the multiplicative model according to the basic operators. We will introduce the basic concepts and computation principles of these two approaches in this section.

3.2.1 Additive Model

The additive model, as the name implies, is a modeling method with addition as the basic operation. Recall that in the introductory part, we have derived a formulation that may contain complex relationships between units and external background knowledge, which would make our discussion exceedingly broad. In this section, to narrow the space of our considered function and establish

fundamental understandings of compositional semantics, we start by simplifying the formula to $\mathbf{p} = f(\mathbf{u}, \mathbf{v})$ and omitting the relationship and background items. Naturally, if we aim to perform addition correctly, $\mathbf{p}$, $\mathbf{u}$, and $\mathbf{v}$ should lie in the same semantic space.

One of the simplest ways is to directly use the sum to represent the joint representation:

$$\mathbf{p} = \mathbf{u} + \mathbf{v}. \tag{3.3}$$

As computed in the foregoing part, the sum of the two vectors representing *machine* and *learning* would be $\mathbf{w}(machine) + \mathbf{w}(learning) = [1, 7, 3, 7, 2]$. It assumes that the composition of different constituents is a symmetric function where $\mathbf{p} = \mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$. That is, it does not consider the order of constituents. Although having lots of drawbacks such as a lack of the ability to model word orders and the absence of background syntactic or knowledge information, this approach still provides a relatively strong baseline [6].

To overcome the word order issue, one easy variant is applying a weighted sum instead of uniform weights. This is, the composition has the following form:

$$\mathbf{p} = \alpha\mathbf{u} + \beta\mathbf{v}, \tag{3.4}$$

where α and β correspond to different weights for two vectors. Under this setting, two sequences (u, v) and (v, u) have different representations when $\alpha \neq \beta$, which is consistent with real language phenomena. For example, *machine learning* and *learning machine* have different meanings and require different representations. To this end, we assign different importance scores to different components. For instance, we set α to 0.3 and β to 0.7, the $0.3 \times \mathbf{w}(machine) = [0, 0.9, 0.3, 1.5, 0.6]$ and $0.7 \times \mathbf{w}(learning) = [0.7, 2.8, 1.4, 1.4, 0]$, and *machine learning* is represented by their addition $0.3 \times \mathbf{w}(machine) + 0.7 \times \mathbf{w}(learning) = [0.7, 3.7, 1.7, 2.9, 0.6]$.

Now, we attempt to incorporate prior knowledge and syntax information into the additive model in a straightforward way. To achieve that, one could combine K nearest neighborhood semantics into composition, deriving:

$$\mathbf{p} = \mathbf{u} + \sum_{i=1}^{L} \mathbf{m}_i + \mathbf{v} + \sum_{i=1}^{K} \mathbf{n}_i, \tag{3.5}$$

where $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_L$ denote semantic neighbors (i.e., synonyms) of $\mathbf{u}$, and $\mathbf{n}_1, \mathbf{n}_2, \ldots, \mathbf{n}_K$ denote semantic neighbors of $\mathbf{v}$. To this end, this method could ensemble such synonyms of the component as a smoothing factor into the composition function, which reduces the variance of language. For example, if in the composition of *machine* and *learning*, the chosen neighbors are *computer* and *optimizing* with $\mathbf{w}(computer) = [1, 0, 0, 0, 1]$ and $\mathbf{w}(optimizing) = [1, 5, 3, 2, 1]$, respectively. This leads to the situation that the representation of *machine learning* becomes $\mathbf{w}(machine) + \mathbf{w}(computer) + \mathbf{w}(learning) + \mathbf{w}(optimizing) =$

[3, 12, 6, 9, 4]. Although it is a simple strategy, the use of synonyms to improve the robustness of language models is still a very effective practice in modern NLP.

When it comes to the measurement of similarity between representations, the cosine function is a natural approach in the semantic space. We will take a closer look to understand the additive model by computing the cosine similarity between $\mathbf{p} = \mathbf{u} + \mathbf{v}$ (we go back to the naive additive model for computation simplicity) and an arbitrary word $\mathbf{w}$. The cosine similarity, denoted as $s(\cdot)$ could be derived as:

$$s(\mathbf{p}, \mathbf{w}) = \frac{\mathbf{p} \cdot \mathbf{w}}{\|\mathbf{p}\| \cdot \|\mathbf{w}\|} = \frac{(\mathbf{u} + \mathbf{v})\mathbf{w}}{\|\mathbf{u} + \mathbf{v}\|\|\mathbf{w}\|} \tag{3.6}$$

$$= \frac{\|\mathbf{u}\|}{\|\mathbf{u} + \mathbf{v}\|} s(\mathbf{u}, \mathbf{w}) + \frac{\|\mathbf{v}\|}{\|\mathbf{u} + \mathbf{v}\|} s(\mathbf{v}, \mathbf{w}). \tag{3.7}$$

From the derivation ahead, it could be concluded that this composition function composes of both magnitude and directions of two component vectors. And the composition similarity of two linguistic units could be viewed as a linear combination of the similarity of two components. In other words, if one vector dominates the magnitude, it will also dominate the similarity. For example, if $\|\mathbf{u}\| = 10^3$ and $\|\mathbf{v}\| = 10^{-3}$, the similarity between $\mathbf{p}$ and $\mathbf{w}$ will be mostly determined by the semantics of $\mathbf{u}$. This could happen if $\mathbf{u}$ is an entity with a strong specific meaning like *Europe* while $\mathbf{v}$ is an empty word like *there*. Further disassembly, in terms of the norm of compositional semantics, we have:

$$\|\mathbf{p}\| = \|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|. \tag{3.8}$$

This lemma suggests that the semantic unit with a deeper-rooted parsing tree could determine the joint representation when combined with a shallow unit. That is, the closer the unit to the final semantic combined unit, the more likely it is to exert a greater influence on the overall semantics.

3.2.2 Multiplicative Model

Though the additive model achieves considerable success in semantic composition, the simplification may also restrict it from performing more complex interactions. Different from the additive model that regards composition as a simple linear transformation, the three-order multiplicative model aims to make higher-order interactions by using multiplication as the basic operator. Among all models from this perspective, the most intuitive approach tried to apply the pairwise product as a composition function approximation. In this method, the composition function is shown as the following:

$$\mathbf{p} = \mathbf{u} \odot \mathbf{v}, \tag{3.9}$$

where, $\mathbf{p}_i = \mathbf{u}_i \cdot \mathbf{v}_i$, which implies each dimension of the output only depends on the corresponding dimension of two input vectors. However, similar to the simplest additive model, this model is also suffering from the lack of the ability to model word order and the absence of background syntactic or knowledge information.

In the additive model, we have $\mathbf{p} = \alpha\mathbf{u} + \beta\mathbf{v}$ to alleviate the word order issue by assigning different weights to different items. Here, α and β are two scalars, which can also be naturally changed to two matrices. The composition function could be represented as:

$$\mathbf{p} = \mathbf{W}_\alpha \cdot \mathbf{u} + \mathbf{W}_\beta \cdot \mathbf{v}, \tag{3.10}$$

where $\mathbf{W}_\alpha$ and $\mathbf{W}_\beta$ are weight matrices that indicate the importance of components $\mathbf{u}$ and $\mathbf{v}$ to the combined unit $\mathbf{p}$. With this expression, the composition could be more expressive and flexible, although much harder to train.

By generalizing the multiplicative model ahead, another approach is to utilize tensors as a multiplicative descriptor, and the composition function could be viewed as:

$$\mathbf{p} = \overrightarrow{\mathbf{W}} \cdot \mathbf{uv}, \tag{3.11}$$

where $\overrightarrow{\mathbf{W}}$ denotes a three-order tensor, i.e., the formula above could be written as $\mathbf{p}_k = \sum_{i,j} \mathbf{W}_{ijk} \cdot \mathbf{u}_i \cdot \mathbf{v}_j$. Hence, this model makes sure that each element of $\mathbf{p}$ could be influenced by all elements of both $\mathbf{u}$ and $\mathbf{v}$, with a relationship of linear combination by assigning each (i, j) a unique weight.

Starting from this simple but general baseline, some researchers proposed to make the function not symmetric to consider word order in the sequence, paying more attention to the first element. The composition function could be:

$$\mathbf{p} = \overrightarrow{\mathbf{W}} \cdot \mathbf{uuv}, \tag{3.12}$$

where $\overrightarrow{\mathbf{W}}$ denotes a four-order tensor. This method could be understood as replacing the linear transformation of $\mathbf{u}$ and $\mathbf{v}$ to a quadratic in $\mathbf{u}$ asymmetrically. So this is a variant of the tensor multiplicative compositional model.

Different from expanding a simple multiplicative model to complex ones, other kinds of approaches are proposed to reduce the parameter space. With the reduction of parameter size, people could make compositions much more efficient rather than having an $O(n^3)$ time complexity in the tensor-based model. Thus, some compression techniques could be applied to the original tensor model. One representative instance is the circular convolution model, which could be shown as:

$$\mathbf{p} = \mathbf{u} \circledast \mathbf{v}, \tag{3.13}$$

where $\circledast$ represents the circular convolution operation with the following definition:

$$\mathbf{p}_i = \sum_j \mathbf{u}_j \cdot \mathbf{v}_{i-j}. \tag{3.14}$$

If we assign each pair with unique weights, the composition function will be:

$$\mathbf{p}_i = \sum_j \mathbf{W}_{ij} \cdot \mathbf{u}_j \cdot \mathbf{v}_{i-j}. \tag{3.15}$$

The circular convolution model could be viewed as a special instance of a tensor-based composition model. If we write the circular convolution in the tensor form, we have $\mathbf{W}_{ijk} = 0$, where $k \neq i + j$. Thus, the parameter number could be reduced from n^3 to n^2, while maintaining the interactions between each pair of dimensions in the input vectors.

Both in the additive and multiplicative models, the basic condition is all components lie in the same semantic space as the output. Nevertheless, different modeling types of words in different semantic spaces could bring us different perspectives. For instance, given $(\mathbf{u}, \mathbf{v})$, the multiplicative model could be reformulated as:

$$\mathbf{p} = \mathbf{W} \cdot (\mathbf{u} \cdot \mathbf{v}) = \mathbf{U} \cdot \mathbf{v}. \tag{3.16}$$

This implies that each left unit could be treated as an operation on the representation of the right one. In other words, each remaining unit could be formulated as a transformation matrix, while the right one should be represented as a semantic vector. This argument could be meaningful, especially for some kinds of phrase compositions. Baroni et al. [2] argue that for *adj-noun* phrases, the joint semantic information could be viewed as the conjunction of the semantic meanings of two components. Given a phrase *red car*, its semantic meaning is the conjunction of all red things and all different kinds of cars. Thus, *red* could be formulated as an operator on the vector of *car*, deriving the new semantic vector, which expressed the meaning of *red car*. These observations lead to another genre of semantic compositional modeling: semantic matrix-composition space.

3.3 *N*-ary Composition

In real-world NLP tasks, the input is typically a sequence of multiple words or tokens rather than just a pair of words. Therefore, besides designing a suitable binary compositional operator, the order to apply binary operations is also important. In this section, we will introduce mainstream strategies in N-ary composition by taking language modeling as an example. To illustrate the language modeling task more clearly, the composition problem to model a sentence or even a document could be formulated as follows. Given a sentence/document consisting of a word

sequence $\{w_1, w_2, ..., w_N\}$, we aim to design the following functions to obtain the joint semantic representation of the whole sentence/document:

1. A semantic representation method like semantic vector space or compositional matrix space.
2. A binary compositional operation function $f(\mathbf{u}, \mathbf{v})$ like we introduced in the previous sections. Here the input $\mathbf{u}$ and $\mathbf{v}$ denote the representations of two constitute semantic units, while the output is also the representation in the same space.
3. An order to apply the binary function in step 2. To describe in detail, we could use a bracket to identify the order to apply the composition function. For instance, we could use $((w_1, w_2), w_3)$ to represent the sequential order from beginning to end.

Methods to model sentence semantics and tackle the above problems could be classified by word-level order: sequential order and convolution order. These composition methods can be particularly implemented by neural networks with corresponding structures. We will introduce the fundamental concepts of the modeling and leave the specific neural network methods to the next chapter.

Sequential Order To design orders to apply binary compositional functions, the most intuitive method is utilizing sequentiality. Namely, the sequence order should be $s_n = (s_{n-1}, w_n)$, where s_{n-1} is the order of the first $n - 1$ words. In this case, the most suitable neural network is the recurrent neural network (RNN). An RNN applies the composition function sequentially and derives the representations of hidden semantic units. Based on these hidden semantic units, we could use them on some specific NLP tasks like sentiment analysis or text classification. Also, note that basic RNNs only utilize the sequential information from head to tail of a sentence/document. To improve the representation ability, RNNs could be enhanced by considering sequential and reverse-sequential information. In RNNs, each hidden state is controlled by the previous hidden state and the input embeddings at the current timestep, thereby forming the composition function of the sequential order.

Convolutional Order In addition to the sequential and recursive order from linguistic intuition, we can also model high-level semantics from the convolutional order. Naturally, this is implemented by a convolutional neural network (CNN), which extracts local features by a convolution layer and then integrates local features via pooling operations to produce sentence-level representations. The starting point of such methods is also to model local features for basic units and then synthesize the universal representation of the entire input. The difference from the previous approaches is that it does not follow the sequence order or syntactic structure but lets the convolutional layer complete this combination automatically.

For the sake of simplicity, this chapter ignores the relationship $\mathcal{R}$ and external knowledge $\mathcal{K}$ of compositional semantics when introducing them. And these two items are challenging to be heuristically defined and applied in traditional computational linguistics. However, the modern NLP, typically based on deep neural

networks, brings a twist to the situation. The tremendous capacity enables neural networks to model almost arbitrarily complex semantic structures in an implicit way, which could be regarded as modeling the $\mathcal{R}$ item (will be introduced in Chap. 4). And advances in knowledge representation learning and knowledge-guided NLP could be naturally seen as a process to model the $\mathcal{K}$ item (will be introduced in Chap. 9).

3.4 Summary and Further Readings

In this chapter, we first introduce the semantic space for compositional semantics. Afterward, we take phrase representation as an example to introduce representative models for binary semantic composition, including additive models and multiplicative models. Finally, we introduce typical methods for N-ary semantic composition. We use fundamental principles and concepts to illustrate the core idea of compositional semantics: to build complex semantics with the combinations of basic components. For further understanding of compositional semantics, readers can refer to some recommended surveys and books that comprehensively introduce the area. For example, the framework applied in this chapter is from the inspiring article of Pelletier et al. [10].

For better modeling compositional semantics, some directions require further efforts in the future. For example, neurobiology-inspired compositional semantics is a promising research topic that explores the neurobiological insights of compositional semantics [11]. The analysis of how language builds meaning and lays out directions in neurobiological research may bring some instructive reference for modeling compositional semantics in representation learning. It is valuable to design novel compositional forms inspired by recent neurobiological advances. There are also studies that attempt to consider discrete symbols in deep neural networks [1, 4], triggering new research issues on the combination of neural models and symbolic models.

Generally speaking, modeling complex semantics distributed in sentences and even documents could be extremely difficult. It may be difficult for us to complete the modeling through heuristic methods. At this point, the powerful fitting and generalization capability of the neural networks are needed to play a role. In the next chapter, we will introduce concepts, methodologies, and applications of sentence and document representation and particularly put focus on the neural network approaches.

Acknowledgments Zhiyuan Liu, Yankai Lin, and Maosong Sun designed the overall architecture of this chapter; Ning Ding and Yankai Lin drafted the chapter. Zhiyuan Liu and Yankai Lin proofread and revised this chapter.

We thank Yang Liu for providing some initial materials in the first edition. We thank Ganqu Cui, Yuan Yao, Shi Yu, Yulin Chen, Xingtai Lv, and Suyuan Zhao for proofreading this chapter.

This is the representation learning for compositional semantics chapter of the second edition of the book *Representation Learning for Natural Language Processing*, with its first edition published

in 2020 [5]. As compared to the first edition of this chapter, main changes include the following: (1) we reorganized the structure and moved the detailed introduction of neural networks (e.g., RNNs and CNNs) to the next chapter; (2) we polished and rewrote the content of the introduction and binary composition; and (3) we removed the semantic space section of the first edition to make this chapter more focused on the basic concepts of semantic composition modeling.

References

1. Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of CVPR*, 2016.
2. Marco Baroni and Roberto Zamparelli. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *Proceedings of EMNLP*, 2010.
3. Gottlob Frege. Die grundlagen derarithmetik. *Eine logisch mathematische Untersuchung u'ber den Begrijfder Zahl. Breslau: Koebner*, 1884.
4. Chen Liang, Jonathan Berant, Quoc Le, Kenneth Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of ACL*, 2017.
5. Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation Learning for Natural Language Processing*. Springer, 2020.
6. Jeff Mitchell and Mirella Lapata. Vector-based models of semantic composition. In *Proceedings of ACL*, 2008.
7. Jeff Mitchell and Mirella Lapata. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429, 2010.
8. Geoffrey Nunberg, Ivan A Sag, and Thomas Wasow. Idioms. *Language*, pages 491–538, 1994.
9. Barbara Partee. Lexical semantics and compositionality. *An Invitation to Cognitive Science: Language*, 1:311–360, 1995.
10. Francis Jeffry Pelletier. The principle of semantic compositionality. *Topoi*, 13(1):11–24, 1994.
11. Liina Pylkkänen. The neural basis of combinatory syntax and semantics. *Science*, 366(6461):62–66, 2019.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Chapter 4
Sentence and Document Representation Learning

Ning Ding, Yankai Lin, Zhiyuan Liu, and Maosong Sun

Abstract Sentence and document are high-level linguistic units of natural languages. Representation learning of sentences and documents remains a core and challenging task because many important applications of natural language processing (NLP) lie in understanding sentences and documents. This chapter first introduces symbolic methods to sentence and document representation learning. Then we extensively introduce neural network-based methods for the far-reaching language modeling task, including feed-forward neural networks, convolutional neural networks, recurrent neural networks, and Transformers. Regarding the characteristics of a document consisting of multiple sentences, we particularly introduce memory-based and hierarchical approaches to document representation learning. Finally, we present representative applications of sentence and document representation, including text classification, sequence labeling, reading comprehension, question answering, information retrieval, and sequence-to-sequence generation.

4.1 Introduction

A natural language sentence is a linguistic unit that conveys complete semantic information, which is composed of words and phrases guided by grammatical rules. Although all the elements in a sentence come from a finite set, they could constitute almost infinite semantics with complex sequential and hierarchical structures. Transforming sentence-level information into computable numerical representations is an intriguing and meaningful research issue for broad tasks of natural language processing.

N. Ding · Z. Liu (✉) · M. Sun
Department of Computer Science and Technology, Tsinghua University, Beijing, China
e-mail: dingn18@mails.tsinghua.edu.cn; liuzy@tsinghua.edu.cn; sms@tsinghua.edu.cn

Y. Lin
Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China
e-mail: yankailin@ruc.edu.cn

© The Author(s) 2023

Z. Liu et al. (eds.), *Representation Learning for Natural Language Processing*,
https://doi.org/10.1007/978-981-99-1600-9_4

In the early stage before deep learning, symbolic strategies are widely adopted to represent sentences. Following the bag-of-words assumption, sentences could be represented as one-hot or term frequency-inverse document frequency (TF-IDF) vectors. However, such methods would bring the computational efficiency problem since the dimension of such representation vectors is usually up to thousands or millions. And these methods also neglect the syntactic structure of a sentence, which is the core of constituent words to express different semantics. By contrast, the n-gram probabilistic language model that assigns probabilities to sequences of words could consider the context while modeling sentences. Despite the simpleness of the probabilistic language model, it inspires the subsequent state-of-the-art neural language models that are based on deep neural networks, such as convolutional neural networks and recurrent neural networks, etc. Compared with conventional symbolic sentence representations, deep neural networks can capture the internal structures of sentences, e.g., sequential and dependency information, through convolutional, recurrent, or self-attention operations, yielding significant success in sentence modeling and NLP tasks.

Documents, usually regarded as the highest-level linguistic unit of natural language, are constituted when there are enough sentences, and they are organized in a particularly logical way. With the rapid development of the Internet, how to effectively retrieve and mine the vast new-coming information from massive amounts of online text becomes a crucial problem for natural language processing. Therefore, document representation plays a vital role in a series of real-world applications and becomes an intriguing research problem. In principle, the aforementioned symbolic or neural-based methods for sentence representation learning could also be applied to documents. But it is also easy to see that the coherence between sentences provides space for more complex combinations to form document-level semantics, thereby producing new challenges. Common approaches to tackle document representation include memory-based and hierarchical methods.

In this chapter, we first introduce symbolic sentence representation learning methods in Sect. 4.2, including the bag-of-words model and probabilistic language models. Then we detail the techniques of neural language models in Sect. 4.3, including feed-forward neural networks, recurrent neural networks, convolutional neural networks, and Transformers. Memory-based and hierarchical methods to model document-level information are elaborated in Sect. 4.4. Finally, we comprehensively introduce representative applications of sentence and document representation in Sect. 4.5, including text classification, sequence labeling, reading comprehension, question answering, information retrieval, recommendation, etc.

4.2 Symbolic Sentence Representation

When words and phrases form sentences, they obtain complete semantics. Similar to word representations in Chap. 2, sentences can also be represented symbolically. But with a slight difference, the sentence is not the smallest unit in this recipe.

A symbol-based sentence representation is composed of multiple symbolic word representations. In this section, we introduce the bag-of-words model and the probabilistic language model for symbolic sentence representation learning.

4.2.1 Bag-of-Words Model

As introduced in Chap. 2, one-hot representation is the most straightforward symbolic method for words and phrases. This approach represents each word with a fixed-length binary vector. For a vocabulary $V = \{w_1, w_2, \ldots, w_{|V|}\}$, the one-hot representation of word w is $\mathbf{w} = [0, \ldots, 0, 1, 0, \ldots, 0]$. Based on the one-hot word representation and the vocabulary, it can be extended to represent a sentence $s = \{w_1, w_2, \ldots, w_N\}$ based on the bag-of-words hypothesis. Bag-of-words model represents sentences as a multiset of its words while ignoring the order and other grammatical rules:

$$\mathbf{s} = \sum_{i=1}^{N} \mathbf{w}_i, \tag{4.1}$$

where N indicates the length of the sentence s. The sentence representation $\mathbf{s}$ is the sum of the one-hot representations of N words within the sentence, i.e., each element in $\mathbf{s}$ represents the term frequency (TF) of the corresponding word. In practice, to prevent it from being biased toward longer texts, it is usually normalized according to the number of words in the whole text.

However, TF alone cannot properly represent a sentence or document since not all the words are equally important. For example, the function words such as *a*, *an*, and *the* usually appear in almost all sentences and reserve little semantics that could represent the sentence or document. Therefore, the inverse document frequency (IDF) is developed to measure the prior importance of w_i in V as follows:

$$\text{idf}_{w_i} = \log \frac{|D|}{\text{df}_{w_i}}, \tag{4.2}$$

where $|D|$ is the number of all sentences or documents in the corpus D and df_{w_i} represents the document frequency (DF) of w_i, which is the number of documents that w_i appears. With the importance of each word, the sentences are represented more precisely as follows:

$$\hat{\mathbf{s}} = \mathbf{s} \odot \text{idf}, \tag{4.3}$$

where $\odot$ is the element-wise product.

Here, $\hat{\mathbf{s}}$ is the TF-IDF representation of the sentence s, and it could be naturally applied to both the sentence and document levels. The insight behind it is that the

more frequently a word appears and the less it appears in other texts, the more it represents the uniqueness of the current text and thus will be assigned more weight. TF-IDF is one of the most popular methods in information retrieval and recommender system [76, 81].

4.2.2 Probabilistic Language Model

One-hot sentence representation identifies important terms to construct the representation and neglects the structural information in a sentence. In this section, we introduce the probabilistic language model, a symbolic sentence representation approach that takes context into account.

A standard probabilistic language model defines the probability of a sentence $s = \{w_1, w_2, \ldots, w_N\}$ by the chain rule of probability:

$$P(s) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots P(w_N|w_1, \ldots, w_{N-1}) \tag{4.4}$$

$$= \prod_{i=1}^{N} P(w_i|w_1, \ldots, w_{i-1}). \tag{4.5}$$

The probability of each word is determined by all the preceding words. And the conditional probabilities of all the words jointly compute the probability of the sentence. However, the model indicated in the Eq. (4.5) is not practicable due to its enormous parameter space for long texts. This is where the n-gram model comes to play, whose core idea is not to use all previous words but $n - 1$ words to predict the current word. We show an example of the n-gram model in Fig. 4.1.

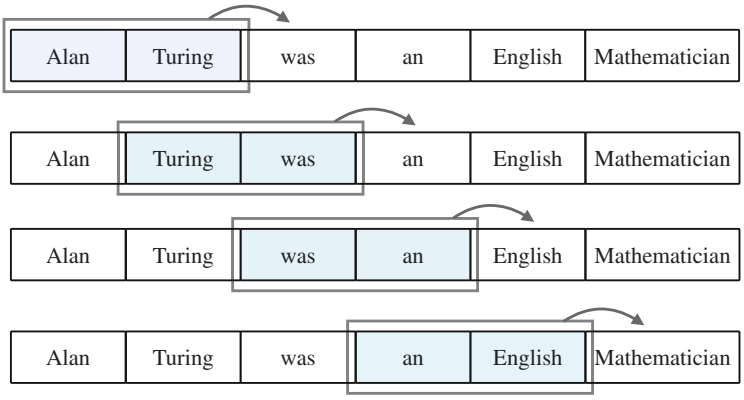


Fig. 4.1 An example of the n-gram language model, where $n = 3$

In practice, we set such $n-1$-sized context windows in the probabilistic language model, assuming that the probability of word w_i only depends on $\{w_{i-n+1} \cdots w_{i-1}\}$. More specifically, an n-gram language model predicts word w_i in the sentence s based on its previous $n-1$ words:

$$P(w_i|w_1, \ldots, w_{i-1}) \approx P(w_i|w_{i-n+1}, \ldots, w_{i-1}). \tag{4.6}$$

After simplifying the language model problem, how to estimate the conditional probability is crucial. In practice, a common approach is maximum likelihood estimation (MLE), which is generally in the following form:

$$P(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \frac{P(w_{i-n+1}, \ldots, w_i)}{P(w_{i-n+1}, \ldots, w_{i-1})}. \tag{4.7}$$

In this equation, the denominator and the numerator can be estimated by counting the frequencies in the corpus. To avoid the probability of some n-gram sequences from being zero, researchers also adopt several types of smoothing approaches, which assign some of the total probability mass to unseen words or n-grams, such as "add-one" smoothing, Good-Turing discounting [31], or back-off models [45].

n-gram model is a typical probabilistic language model for predicting the next word in an n-gram sequence, which follows the Markov assumption that the probability of the target word only relies on the previous $n-1$ words. The idea is employed by most current sentence modeling methods, where the n-gram language model serves as an approximation of the true language model. This hypothesis is crucial because it substantially simplifies the problem of learning the parameters of language models from data. Recent works on word representation learning [1, 69, 72] are mainly based on the n-gram language model.

The introductory part of this chapter states that the semantic information of a sentence not only exists in constituent words but is also closely related to its flexible syntactic structure. Obviously, despite its simplicity, symbolic approaches treat constituent words as independent symbols and are not capable of representing rich semantic information. Symbolic methods for sentence representation learning have been extensively introduced by many classical textbooks [42]. In this chapter, we mainly focus on sentence representations based on neural networks, which is a common practice in modern NLP.

4.3 Neural Language Models

Although the aforementioned symbolic methods are cornerstones to represent sentences in inchoate NLP, they still face challenges in modeling rich semantic information and universal information distributed in flexible structures of sentences.

To this end, a set of more powerful modeling tools, neural networks, are developed for language modeling. Different from symbolic methods, neural language models use continuous representations to represent all words, which enjoy better generalization and modeling capability for longer texts.

A neural network could also be viewed as an estimator of the language model function, and the architecture could be flexible in this setting. Similar to n-gram probabilistic language models, neural language models are constructed and trained to model a probability distribution of a target word conditioned on previous words:

$$P(s) = \prod_{i=1}^{N} P(w_i | w_{i-n+1}, \ldots, w_{i-1}), \tag{4.8}$$

where the conditional probability of the selecting word w_i can be calculated by multiple kinds of neural networks and the common choices include the feed-forward neural network, recurrent neural network, convolutional neural network, etc. The training of neural language models is achieved by optimizing the cross-entropy loss function:

$$\mathscr{L} = -\sum_{i=1}^{N} \log P(w_i | w_{i-n+1}, \ldots, w_{i-1}). \tag{4.9}$$

The parameters of the language model will be iteratively optimized during training and result in a language model that could predict the next word based on the context. In the following sections, we will detail these neural language models.

4.3.1 Feed-Forward Neural Network

Whether it is a probabilistic language model or a neural language model, the primary goal is to estimate the conditional probability $P(w_i | w_1, \ldots, w_{i-1})$. And as stated, adopting the idea of n-gram to approximate the conditional probability is a common approach, where each word is determined by its $n-1$ context words, i.e., $P(w_i | w_1, \ldots, w_{i-1}) \approx P(w_i | w_{i-n+1}, \ldots, w_{i-1})$. In this section, we first introduce language modeling with the feed-forward neural network (FFN).

The architecture of the FFN language model is proposed by Bengio et al. [1] (illustrated in Fig. 4.2). Although more sophisticated neural architectures could be applied to the problem, the FFN language model first elaborates on the methodology of neural-based language modeling. To evaluate the conditional probability of the word w_i, it first projects its $n-1$ context-related words to their word vector representations $[\mathbf{w}_{i-n+1}, \ldots, \mathbf{w}_{i-1}]$ and concatenate the representations

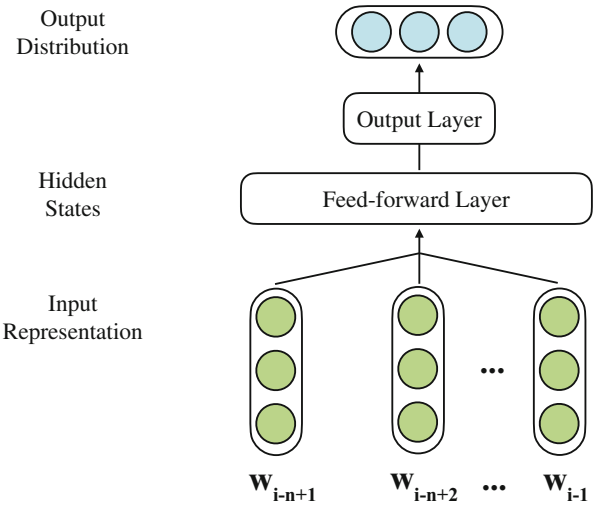


Fig. 4.2 The architecture of the feed-forward neural network

$\mathbf{x} = \text{concat}(\mathbf{w}_{i-n+1}; \ldots; \mathbf{w}_{i-1})$ to feed them into a FFN. The formulation can be generally written as follows:

$$\mathbf{h} = \mathbf{M} f(\mathbf{W}_1 \mathbf{x} + \mathbf{b}) + \mathbf{W}_2 \mathbf{x} + \mathbf{d}, \tag{4.10}$$

where $f(\cdot)$ is an activation function, $\mathbf{W}_1, \mathbf{W}_2$ are weighted matrices to transform word vectors into hidden representations, $\mathbf{M}$ is a weighted matrix for the connections between the hidden layer and the output layer, and $\mathbf{b}$, $\mathbf{d}$ are bias terms. And then, the conditional probability of the word w_i can be calculated by a Softmax function:

$$P(w_i | w_{i-n+1}, \ldots, w_{i-1}) = \text{Softmax}(\mathbf{h}). \tag{4.11}$$

4.3.2 Convolutional Neural Network

Convolutional neural networks (CNNs) use convolutional layers to conduct the basic operation. This type of neural network layer represents the context by extracting hierarchical information from it [23]. For the input words $\{w_1, \ldots, w_l\}$, we first obtain their word embeddings $[\mathbf{w}_1, \ldots, \mathbf{w}_N]$. Let d denote the dimension of the hidden states. The convolutional layer involves a sliding window with the size of k of the input vectors centered on each word vector using a kernel matrix $\mathbf{W}_c$. And the hidden representation could be calculated by

$$\mathbf{h} = f(\mathbf{X} * \mathbf{W}_c + \mathbf{b}), \tag{4.12}$$

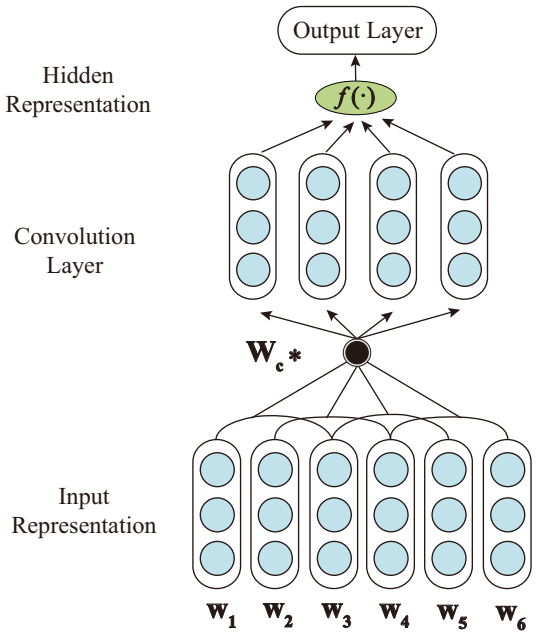


Fig. 4.3 The architecture of the convolutional neural network

where $*$ is the convolution operation, $f(\cdot)$ is a nonlinear activation function (e.g., a sigmoid or tangent function), $\mathbf{X} \in \mathbb{R}^{l \times d}$ is the matrix of word embeddings, $\mathbf{W}_c \in \mathbb{R}^{k \times d \times d'}$ (d' is the kernel size), and $\mathbf{b} \in \mathbb{R}^{d'}$ are learned parameters. The sliding window prevents the model from seeing the subsequent words so that $\mathbf{h}$ does not learn information from future words. For each sliding step, the hidden state of the current word is computed based on the previous k words and then further fed to an output layer to calculate the probability of the present word. The architecture of a CNN is shown in Fig. 4.3. In practice, we can use distinct lengths of sliding windows to form multi-channel operations to learn local information with different scales.

4.3.3 Recurrent Neural Network

To address the lack of ability to model long-term dependency in the FFN language model, Mikolov et al. [70] propose a recurrent neural network (RNN) language model which applies an RNN in language modeling. RNNs are different from FFNs in a fundamental way in that they operate in an internal state space where representations can be sequentially processed. Therefore, the RNN language model can deal with those sentences of arbitrary length. At every time step, its input is the vector of its previous word instead of the concatenation of vectors of its $n - 1$

previous words. The information of all other previous words can be considered by its internal state.

Given the input word embeddings $\mathbf{x} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N]$, at timestep t, the current hidden state $\mathbf{h}_t$ is computed based on the current input $\mathbf{w}_t$ and the hidden state of the last timestep $\mathbf{h}_{t-1}$. Formally, the RNN language model can be defined as

$$\mathbf{h}_t = f(\mathbf{W}\,\text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}), \tag{4.13}$$

$$\mathbf{y} = \text{Softmax}(\mathbf{M}\mathbf{h}_t + \mathbf{d}), \tag{4.14}$$

where $f(\cdot)$ is a nonlinear activation function, $\mathbf{y}$ represents a probability distribution over the given vocabulary, $\mathbf{W}$ and $\mathbf{M}$ are weighted matrices and $\mathbf{b}$, $\mathbf{d}$ are bias terms. As the increase of the length of the sequence, a common issue of the RNN language model is the vanishing gradients problem. The architecture of the RNN language model is shown in Fig. 4.4. Here, the RNN unit can also be implemented in other variants of recurrent neural networks, e.g., long short-term memory (LSTM) and gated recurrent unit (GRU).

LSTM Since the raw RNN only utilizes the simple tangent function, it is hard to obtain the long-term dependency of a long sentence/document. Hochreiter et al.

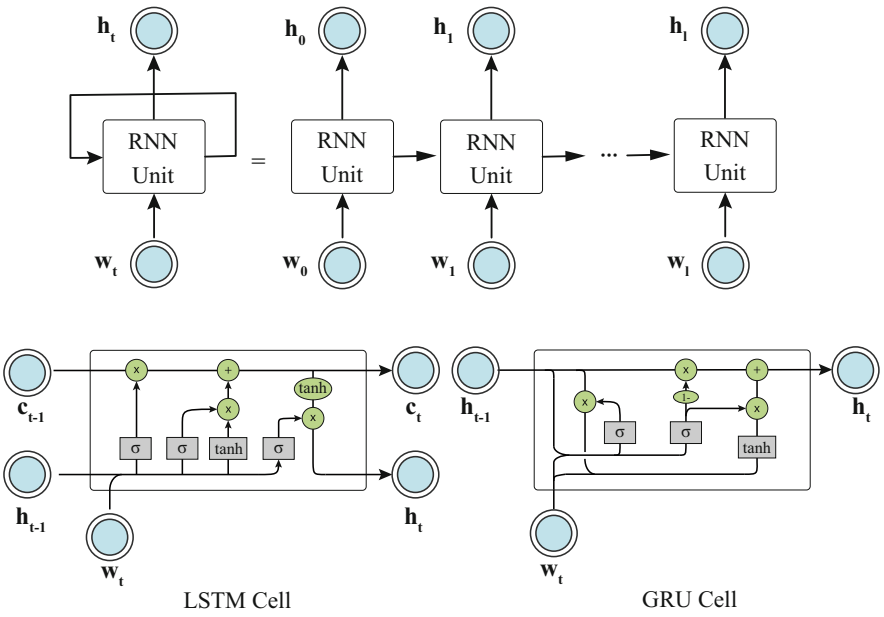


Fig. 4.4 The architecture of recurrent neural networks. The figure is re-drawn according to the blog for introducing LSTM models (https://colah.github.io/posts/2015-08-Understanding-LSTMs/)

[37] propose long short-term memory (LSTM) networks to strengthen the ability to model long-term semantic dependency in RNN.

LSTM introduces a cell state $\mathbf{c}_t$ to represent the current information at timestep t, which is computed from the cell state at the last timestep $\mathbf{c}_{t-1}$ and the candidate cell state of the current timestep $\tilde{\mathbf{c}}_t$. And the representation of the current timestep $\mathbf{h}_t$ is calculated based on $\mathbf{c}_t$. Formally,

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}_c), \tag{4.15}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \tag{4.16}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{4.17}$$

where $\odot$ is the element-wise multiplication operation, $\mathbf{W}_c$ and $\mathbf{b}_c$ are learnable parameters and $\mathbf{f}_t$, $\mathbf{i}_t$, and $\mathbf{o}_t$ are different gates introduced in LSTM to control the information flow. Specifically, $\mathbf{f}_t$ is the forgetting gate to determine how much information of the cell state at the last timestep $\mathbf{c}_{t-1}$ should be forgotten, $\mathbf{i}_t$ is the input gate to control how much information of the candidate cell state at the current timestep $\tilde{\mathbf{c}}_t$ should be reserved, and $\mathbf{o}_t$ is the output gate to control how much information of the current cell state $\mathbf{c}_t$ should be output to the representation $\mathbf{h}_t$. And all these gates are computed by the representation of the last timestep $\mathbf{h}_{t-1}$ and the current input $\mathbf{w}_t$. Formally, it could be written as

$$\mathbf{f}_t = \text{Sigmoid}(\mathbf{W}_f \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}_f), \tag{4.18}$$

$$\mathbf{i}_t = \text{Sigmoid}(\mathbf{W}_i \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}_i), \tag{4.19}$$

$$\mathbf{o}_t = \text{Sigmoid}(\mathbf{W}_o \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}_o), \tag{4.20}$$

where $\mathbf{W}_f$, $\mathbf{W}_i$, $\mathbf{W}_o$ are weight matrices and $\mathbf{b}_f$, $\mathbf{b}_i$, $\mathbf{b}_o$ are bias terms in different gates. It is generally believed that LSTM could model longer text than the vanilla RNN model.

GRU To simplify LSTM and obtain more efficient algorithms, Chung et al. [19] propose to utilize a simple but comparable RNN architecture, named gated recurrent unit (GRU), which also utilizes the gating mechanism to handle information flow. But compared to several gates with different functionalities, GRU uses an update gate $\mathbf{z}_t$ to control the information flow. And a reset gate $\mathbf{r}_t$ is adopted to control how much information from the last step hidden state $\mathbf{h}_{t-1}$ would flow into the candidate hidden state of the current step $\tilde{\mathbf{h}}$. Formally, the computation flow of GRU is as follows:

$$\mathbf{z}_t = \text{Sigmoid}(\mathbf{W}_z \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}_z), \tag{4.21}$$

$$\mathbf{r}_t = \text{Sigmoid}(\mathbf{W}_r \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}_r), \tag{4.22}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h \text{concat}(\mathbf{w}_t; \mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h), \tag{4.23}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t, \tag{4.24}$$

where $\mathbf{W}_z$, $\mathbf{W}_r$, $\mathbf{W}_h$ are weight matrices and $\mathbf{b}_z$, $\mathbf{b}_r$, $\mathbf{b}_h$ are bias terms. The update gate z in GRU simultaneously manages the historical and current information. Moreover, the model also omits cell modules $\mathbf{c}$ in LSTM and directly uses hidden states $\mathbf{h}$ in the computation. GRU has fewer parameters, which brings higher efficiency and could be seen as a simplified version of LSTM.

Generally, compared to CNNs, RNNs are more suitable for the sequential characteristic of textual data. However, the nature of each step's hidden state is dependent on the previous step also makes RNNs difficult to perform parallel computation and thus slower in training.

4.3.4 Transformer

Since 2017, a more powerful neural architecture, the Transformer [96] model, which is equipped with a self-attention mechanism, has received extensive attention from the NLP community. Compared to RNNs, Transformers could handle sequential data in parallel instead of processing a word at a timestep. The Transformer model has become a mainstream choice of neural networks to model natural language and pre-trained language models based on deep Transformers have achieved state-of-the-art results on various NLP tasks. In this section, we introduce the mechanism of the Transformer model. We will use the next chapter to introduce the progress and research issues of representation learning brought by pre-trained models.

Structure A Transformer is a nonrecurrent encoder-decoder architecture with a series of attention-based blocks. For the encoder, there are multiple layers, and each layer is composed of a multi-head attention sublayer and a position-wise feed-forward sublayer. And there is a residual connection and layer normalization of each sublayer. The decoder also contains multiple layers, and each layer is slightly different from the encoder. First, sublayers of multi-head attention and feed-forward with identical structures with the encoder are adopted. And the input of the multi-head attention sublayer is from both the encoder and the previous sublayer, which is additionally developed. This sublayer is also a multi-head attention sublayer that performs self-attention over the outputs of the encoder. And the sublayer adopts a masking operation to prevent the decoder from seeing subsequent tokens. The architecture of the Transformer is shown in Fig. 4.5.

Attention There are several attention heads in the multi-head attention sublayer. A *head* represents a scaled dot-product attention structure, which takes the query

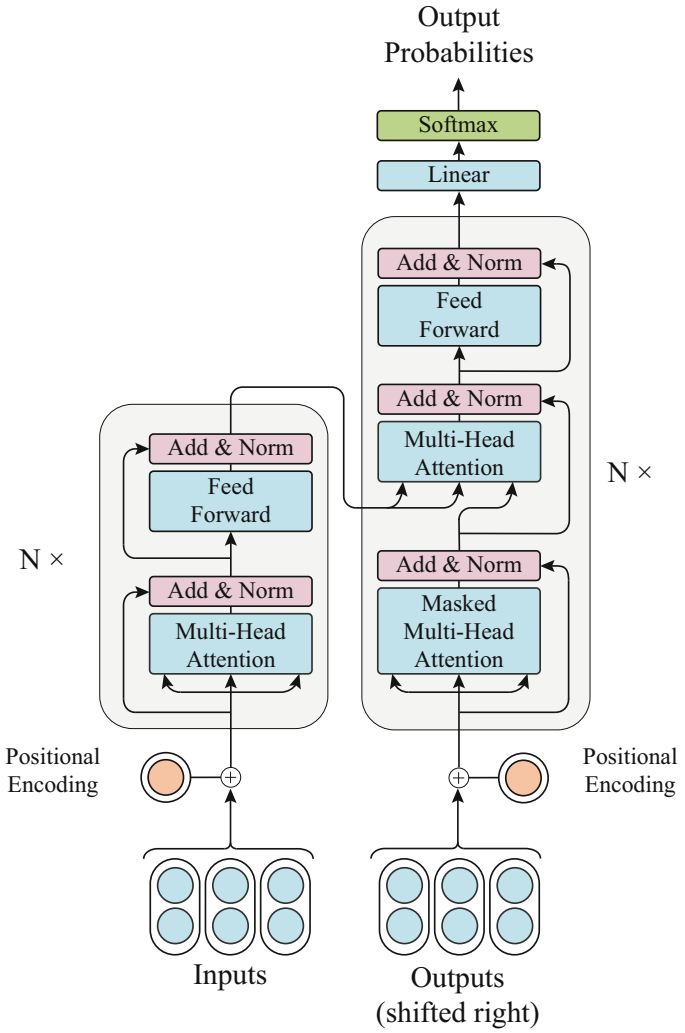


Fig. 4.5 The architecture of a Transformer. This figure is re-drawn according to Fig. 1 from Google's Transformer paper [96]

matrix $\mathbf{Q}$, the key matrix $\mathbf{K}$, and the value matrix $\mathbf{V}$ as the inputs, and the output is computed by

$$\mathrm{ATT}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_k}}\right)\mathbf{V}, \tag{4.25}$$

where d_k is the dimension of the query matrix; note that in language models, $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ usually come from the same source, i.e., the input sequences. Specifically, they are obtained by the multiplication of the input embedding $\mathbf{H}$ and three weight matrices $\mathbf{W}^Q$, $\mathbf{W}^K$, and $\mathbf{W}^V$, respectively. The dimensions of query, key, and value vectors are d_k, d_k, and d_v, respectively. The computation in Eq. (4.25) is typically known as the self-attention mechanism.

The multi-head attention sublayer linearly projects the input hidden states $\mathbf{H}$ several times into the query matrix, the key matrix, and the value matrix for h heads. The multi-head attention sublayer could be formulated as follows:

$$\text{Multihead}(\mathbf{H}) = [\text{head}_1, \text{head}_2, \ldots, \text{head}_h]\mathbf{W}^O, \tag{4.26}$$

where $\text{head}_i = \text{ATT}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$ and $\mathbf{W}_i^Q$, $\mathbf{W}_i^K$, and $\mathbf{W}_i^V$ are linear projections. $\mathbf{W}^O$ is also a linear projection for the output.

After operating self-attention, the output would be fed into a fully connected position-wise feed-forward sublayer, which contains two linear transformations with ReLU activation:

$$\text{FFN}(\mathbf{x}) = \mathbf{W}_2 \max(0, \mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2. \tag{4.27}$$

Input Tokenization Tokenization is a crucial step in NLP to process the raw input sequences. Generally, tokenization converts the input sequence into "tokens" and feeds them to subsequence processing modules. A simple approach is to directly regard a word as a token, whereas such a method cannot well handle unknown out-of-vocabulary (OOV) words and cannot grasp the correlations of similar words. For example, it is more intuitive to tokenize "apples" into "apple" and "s" than a separate token independent of "apple." In modern NLP, more mature methods like byte pair encoding (BPE) and wordpiece are extensively applied to Transformer-based models. Taking BPE as an example, it iteratively replaces two adjacent units with a new unit, which ensures that common words will remain as a whole and uncommon words are split into multiple subwords. Practically, BPE is applied to many pre-trained models such as RoBERTa [64] and GPT-2 [79], and wordpiece is used to pre-train BERT [24].

Positional Encoding Positional encoding indicates the position of each token in an input sequence. The self-attention mechanism of Transformers does not involve positional information. Thus, the model needs to represent positional information of the input sequence additionally. Transformers do not use integers to represent positions because the value range varies with the input length. For example, positional values may become very large if the model process a long text, which will restrain the generalization over texts of different lengths.

Specifically, each position is encoded to a particular vector with the same dimension d of the hidden states to represent the positional information. For the k-th

token, let $\mathbf{p}^k$ be the positional vector; the i-th element of the positional encoding $\mathbf{p}_i^k$ is calculated by

$$\mathbf{p}_i^k = \sin\left(\frac{k}{10{,}000^{\frac{2j}{d}}}\right), \text{ if } i = 2j, \tag{4.28}$$

$$\mathbf{p}_i^k = \cos\left(\frac{k}{10{,}000^{\frac{2j}{d}}}\right), \text{if } i = 2j + 1. \tag{4.29}$$

In this way, for each positional encoding vector, the frequency would decrease along with the dimension. We can imagine that at the end of each vector, $k/10{,}000^{\frac{2j}{d}}$ is near to 0 since the denominator becomes very large, which makes $\sin(k/10{,}000^{\frac{2j}{d}})$ approximates 0 and $\cos(k/10{,}000^{\frac{2j}{d}})$ approximates 1. Assuming the state of alternating 0s and 1s is a kind of "stable point," for different positions k, the "speed" to reach such a stable point is also different. That is, the later the token is (larger k), the later the value $k/10{,}000^{\frac{2j}{d}}$ will be close to 0. Moreover, no matter the text lengths the model is currently processing, the encoding values are stable and range from -1 to 1. Alternatively, learnable positional embeddings could also be applied to Transformers and could consistently yield similar performance. Pre-trained language models like BERT [24] adopt learnable position embeddings rather than sinusoidal encoding.

Although the Transformer model was proposed to tackle machine translation, the powerful capability to model sequential data makes it the most popular backbone of NLP applications. For example, it has become the standard architecture for pre-trained language models, and GPT is a representative example of using a Transformer for the language modeling task. As stated, the overall objective is $\mathscr{L} = -\sum_{i=1}^{N} \log P(w_i | w_{i-n+1}, \ldots, w_{i-1})$. Here, we use the decoder of a Transformer to adopt the self-attention mechanism to the previous $n-1$ words of the current word, and the output will be further fed into the feed-forward sublayer. After multiple layers of propagation, the final probability distribution P is computed by a softmax function acting on the hidden representation. Compared to RNNs, Transformers could better model the long-term dependency, where all tokens will be equally considered and computed during the attention operation.

4.3.5 Enhancing Neural Language Models

The foregoing parts have described representative neural language models. Next, we introduce some techniques that can further improve the performance of such models, including word classification and the caching approach.

Word Classification Researchers [9, 32] propose a class-based language model to adopt word classification to improve the performance and speed of the language model. In this class-based language model, all words are assigned to a unique class, and the conditional probability of a word given its context can be decomposed into the probability of the word's class given its previous words and the probability of the word given its class and history, which is formally defined as

$$P(w_i|w_{i-n+1},\ldots,w_{i-1}) = \sum_{c(w_i)\in C} P(w_i|c(w_i))P(c(w_i)|w_{i-n+1},\ldots,w_{i-1}), \tag{4.30}$$

where C indicates the set of all classes and $c(w_i)$ indicates the class of word w_i.

Moreover, Morin et al. [73] propose a hierarchical neural network language model, which extends word classification to hierarchical binary clustering of words in the language model. Instead of simply assigning each word a unique class, it first builds a hierarchical binary tree of words according to the word similarity obtained from WordNet. Next, it assigns a unique bit vector $[c_1(w_i), c_2(w_i), \ldots, c_N(w_i)]$ for each word, which indicates the hierarchical classes of them. And then, the conditional probability of each word can be defined as

$$\begin{aligned} &P(w_i|w_{i-n+1},\ldots,w_{i-1}) \\ &\quad = \prod_{j=1}^{N} P(c_j(w_i)|c_1(w_i),\ldots,c_{j-1}(w_i),w_{i-n+1},\ldots,w_{i-1}). \end{aligned} \tag{4.31}$$

The hierarchical neural network language model can achieve $\mathcal{O}(k/\log k)$ speed up as compared to a standard language model. However, the experimental results of [73] show that it performs worse than the standard language model. The reason is that the introduction of hierarchical architecture or word classes imposes a negative influence on word classification by neural network language models.

Caching Caching is also one of the important extensions of neural language models. A type of cache-based language model assumes that each word in a recent context is more likely to appear again [90]. Hence, the conditional probability of a word can be calculated by the information from history and caching:

$$\begin{aligned} P(w_i|w_{i-n+1},\ldots,w_{i-1}) &= \lambda P_s(w_i|w_{i-n+1},\ldots,w_{i-1}) \\ &\quad + (1-\lambda)P_c(w_i|w_{i-n+1},\ldots,w_{i-1}), \end{aligned} \tag{4.32}$$

where $P_s(w_i|w_{i-n+1},\ldots,w_{i-1})$ indicates the conditional probability generated by standard language models and $P_c(w_i|w_{i-n+1},\ldots,w_{i-1})$ indicates the conditional probability retrieved from cache, and λ is a constant.

Another cache-based language model is also used to speed up the RNN language modeling [39]. The main idea of this approach is to store the outputs and states of language models for future predictions given the same contextual history.

Neural language models are among the most powerful techniques for sentence representations, which could comprehensively model the complex syntactical structures of long texts. Even for longer documents, modern approaches are based on neural networks. And in the next chapter, we will discuss how to model documents more effectively.

4.4 From Sentence to Document Representation

The aforementioned representation learning approaches could be applied to both sentence- and document-level texts since most existing works treat documents as "longer sentences" in practice. However, the interactions of multiple sentences in a document bring more complex semantics, thereby establishing new challenges. In this section, we introduce two types of document representation learning methods. Memory-based document representation treats the document as a whole to directly learn the representation, and hierarchical document representation performs the fusion of the information of different levels of linguistic units to obtain the final document representation.

4.4.1 Memory-Based Document Representation

A direct way to learn the document representation is to regard the document as a whole. We regard this type of method as the memory-based document representation whose intuition is to use inherent modules to remember the context with critical information of the target document.

Paragraph Vector Here, we extend the idea of word2vec to the document level, which is named paragraph vector (PV) [53]. Given a target word and the corresponding contexts from the document, the training objective of this strategy is to use the paragraph vector to predict the target word. More specifically, similar to word2vec, PV has two variants: distributed memory (denoted as PV-DM) and distributed bag-of-words (denoted as PV-DBOW).

As illustrated in Fig. 4.6, PV-DM adds an additional token in each document and uses the token representation to represent the document. By extending the idea of CBOW, PV-DM predicts the target word according to historical contexts and document representation in the training phase. There are multiple choices exploiting the document representation and word representations. For example, one can directly concatenate these representations or average them. It can be seen that the additional document representation here acts as a memory module that gradually

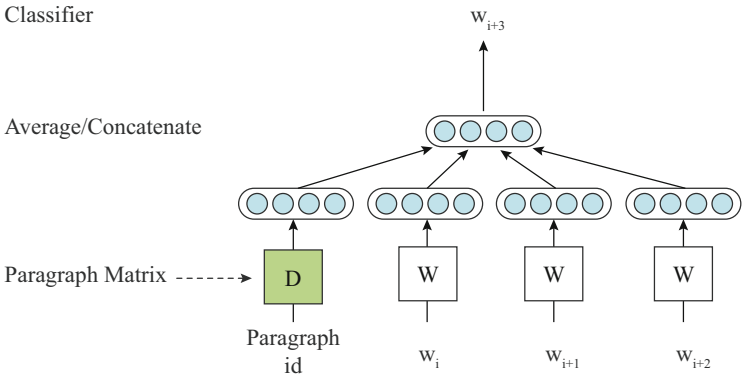


Fig. 4.6 The architecture of PV-DM model. This figure is re-drawn according to Fig. 2 from the paragraph vector paper [53]

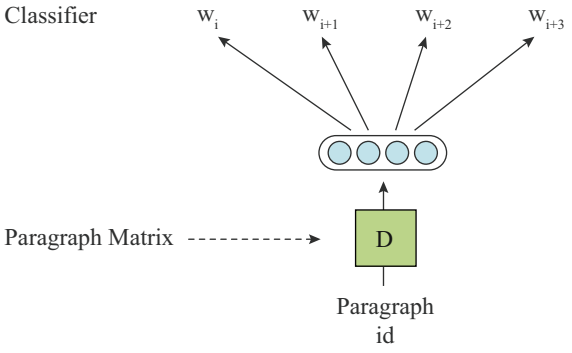


Fig. 4.7 The architecture of PV-DBOW model. This figure is re-drawn according to Fig. 3 from the paragraph vector paper [53]

captures the key semantics of the document as it participates in the training process of predicting words based on context. After training, the paragraph vectors can be regarded as the representations of the documents and be used as pre-trained document embeddings like pre-trained word embeddings.

Besides PV-DM, PV-DBOW extends the idea of skip-gram to learn document representation. As illustrated in Fig. 4.7, PV-DBOW ignores the context words in the input text and directly uses the document representation to predict the target word in a randomly sampled window. In the training phase, the model will randomly sample a window and then randomly sample a word to be the prediction target. Obviously, it is simpler in concept than PV-DM, and experiments have shown that this method is also effective for document representation.

However, when the document is too long, PV may not have enough capacity to remember enough information. These issues could be alleviated by modern deep neural networks. In the following parts, we introduce deep neural networks that

yield superior performance in storing and representing historical information as memories.

Memory Networks In the era of deep learning, memory networks [104] have become one of the representative methods for learning document representation, which uses memory units to store and maintain long-term information. Compared to standard neural networks that utilize special aggregation operations to obtain the document representation, memory networks explicitly adopt memory neural modules to store information, which could prevent information forgetting. Given an input document with n sentences $d = \{s_1, s_2, \ldots, s_n\}$, for the i-th sentence s_i, the model firstly uses a feature extractor F to transform the input into a sentence-level representation $\mathbf{h}_i^s$:

$$\mathbf{h}_i^s = F(s_i). \tag{4.33}$$

A memory unit M is responsible for storing and updating memories according to the current inputs. In this case, the memories will be updated by certain operations. For the specific update mechanism of the memory module, there are many options to define M. Here we introduce one of the most straightforward methods by using slots. The basic idea of this approach is to store the representation of each input into a separate "slot":

$$\mathbf{m}_{H(s_i)} = F(s_i), \tag{4.34}$$

where $H(\cdot)$ is used to select a particular index of a slot for the input sentence s_i. In this case, M only updates one slot with index $H(s_i)$ given the input s_i and does not interfere with any other memories.

Given memories stored through the aforementioned process, we could find the k most relevant memories given a query q and generate a final output. An output module O is adapted to select supporting memories and generate the latent representation of the output of the current query q. We take $k = 1$ as an example, where the module selects one memory index:

$$o = O(q, \mathbf{m}) = \text{argmax}_j \, s_O(q, \mathbf{m}_j), \tag{4.35}$$

where $s_O(\cdot)$ is a score function to evaluate the relevance of the query and memories. Then we can use a decoder D to generate concrete tokens. In particular, if the final output y is a single word, given query q, memory $\mathbf{m}_o$ (o is the selected index of memory produced by O), and dictionary V, we use another score function $s_y(\cdot)$ that measures the candidate word and o to produce y:

$$y = \text{argmax}_{w \in V} \, s_y([q, \mathbf{m}_o], w). \tag{4.36}$$

The framework is illustrated in Fig. 4.8.

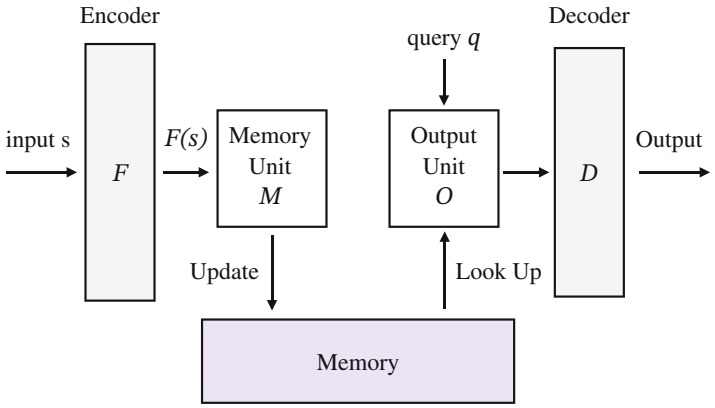


Fig. 4.8 The general architecture of memory networks

In this general framework, each module can be carefully designed to store various historical information. The framework is effective for document modeling and could be applied to many related tasks. For example, in reading comprehension question answering, we can store the representation of each sentence of the input passage as memory and then match the query against the memory to select the answer more accurately.

Variants of Memory Networks Subsequently, many memory network variants are designed from different perspectives. Generally, the improvement can be delivered from the training strategy and the memory form.

Training Strategy If the operation of each module is designed discretely, it is not easy to directly train the network via back-propagation.

The end-to-end memory network [91] presents a continuous version of this framework, which uses an RNN-based architecture (it can also be replaced with other neural backbones) to read the stored memories before outputting the results. Specifically, given a document $d = [s_1, s_2, \ldots, s_n]$, for a sentence s_i, an encoder F is adopted to obtain the representation $\mathbf{h}_i^s$, which is regarded as the raw memory for s_i. Given a query q, whose representation is $\mathbf{q}$, we need to extract relevant memories and produce the final output. As shown in Fig. 4.9, the model generates a memory vector $\mathbf{m}_i$ and an output vector $\mathbf{c}_i$ for each $\mathbf{h}_i^s$ with a trainable matrix $\mathbf{W}_m$ and another trainable matrix $\mathbf{W}_c$, respectively:

$$\mathbf{m}_i = \mathbf{W}_m \mathbf{h}_i^s, \mathbf{c}_i = \mathbf{W}_c \mathbf{h}_i^s. \tag{4.37}$$

Memory vectors are used to compute matching scores p against the query q with a softmax function. Specifically for the i-th memory vector:

$$p_i = \text{Softmax}(\mathbf{m}_i^\top \mathbf{q}). \tag{4.38}$$

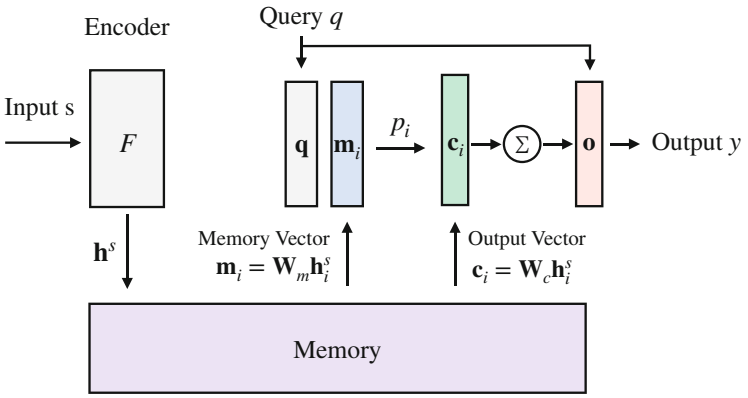


Fig. 4.9 The architecture of end-to-end memory network. This figure is re-drawn according to Fig. 1 from the end-to-end memory network paper [91]

The matching score p_i is then used as a weight of the output vector $\mathbf{c}_i$ to represent the relevance between q and $\mathbf{m}_i$. By conducting a weighted sum, we obtain a vector $\mathbf{o}$ that is responsible for the final output:

$$\mathbf{o} = \sum p_i \mathbf{c}_i . \tag{4.39}$$

Finally, the final output y given a query q could be derived from the query representation $\mathbf{q}$ and the vector $\mathbf{o}$:

$$y = \text{Softmax}(\mathbf{W}_o(\mathbf{q} + \mathbf{o})), \tag{4.40}$$

where $\mathbf{W}_o$ are trainable parameters. As we can see, in the training procedure, $\mathbf{W}_m$, $\mathbf{W}_c$, $\mathbf{W}_o$, and the encoder F will be optimized in an end-to-end manner by directly minimizing the cross-entropy loss between the prediction and the ground truth label.

Dynamic memory networks [48] present a similar methodology. After the model produces representations for all the input sentences and the current query, the query representation will trigger a retrieval procedure based on the attention mechanism. This procedure will iteratively read the stored memories and retrieve the relevant ones to produce the output. The transformation of memory networks into the end-to-end manner in terms of training strategies has further expanded its influence and inspired new research works [61, 62, 106, 108].

Memory Form In addition to the training strategy perspective, we can also improve memory networks from the perspective of the form of stored memories. It is easy to see that such a framework may be difficult to store vast amounts of information because it is hard to compute matching scores for large-scale memories. Hierarchical memory networks [10] give a solution that organizes memories in a hierarchical form. This method forms a group of memories, and then multiple

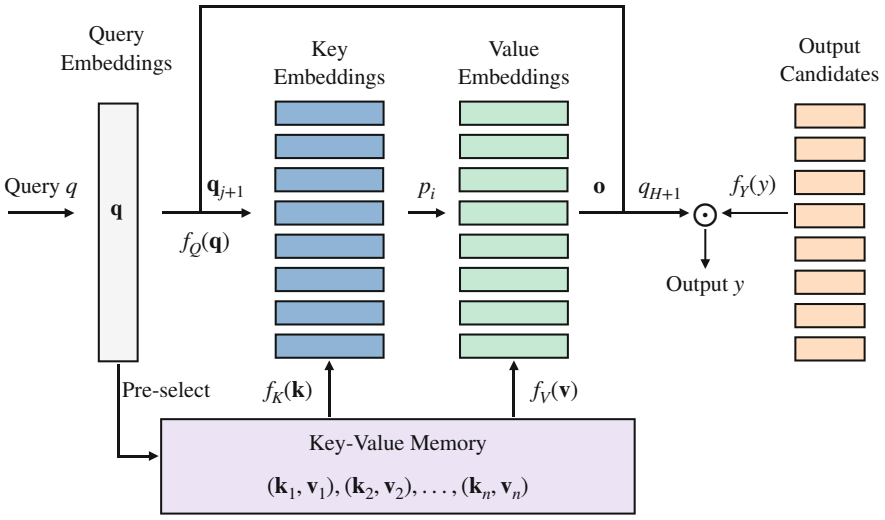


Fig. 4.10 The architecture of key-value memory network. This figure is re-drawn according to Fig. 1 from the key-value memory network paper [71]

groups can be reorganized into higher-level groups. It uses a maximum inner product search combined with the attention mechanism to efficiently retrieve desired memory. This approach effectively improves the efficiency of searching memory but may also risk losing some precision as the number of levels increases.

Key-value memory network (KV-MemNN) [71], as the name means, uses a key-value structure to store and organize memories. The design of such a structure is to boost the process of retrieving memories and could store information from different sources (e.g., text and knowledge graphs). The framework is illustrated in Fig. 4.10. Formally, the memories are pairs of key-values $(\mathbf{k}_i, \mathbf{v}_i)$. Suppose that we already have large-scale established key-value memories, given a query q and the corresponding representation $\mathbf{q}$; the model could use it to preselect a small group of memories by directly matching the words in the query and memories with the reverse index. After narrowing the search space, one can calculate the relevant score between the query and a key:

$$p_i = \text{Softmax}(f_Q(\mathbf{q}) \cdot f_K(\mathbf{k}_i)), \tag{4.41}$$

where $f_Q(\cdot)$ and $f_K(\cdot)$ are feature mapping functions. Similar to the end-to-end memory network, in the reading stage, the vector $\mathbf{o}$ responsible for the final output could be calculated by a weighted sum:

$$\mathbf{o} = \sum p_i f_V(\mathbf{v}_i), \tag{4.42}$$

where $f_V(\cdot)$ is a feature mapping function. It is noteworthy that the output vector leverages both the key and value representations. Another prominent feature of KV-MemNN is that the query can be iteratively updated during the training process. The intuition of this mechanism is that the retrieved memory key could be incorporated into the query and produce a new query to find more accurate memories. Formally, the updated query is

$$\mathbf{q}_{j+1} = \mathbf{W}_q(\mathbf{q}_j + \mathbf{o}), \tag{4.43}$$

where $\mathbf{W}_q$ is a transformation matrix. Accordingly, the relevant score of the updated query and memories is $p_i = \text{Softmax}(\mathbf{q}_{j+1}^\top f_K(\mathbf{k}_i))$. The number of updates to the query H is a fixed value, which is treated as a hyperparameter. Thus, the final prediction of the model is

$$y = \text{argmax}_k \ \text{Softmax}(\mathbf{q}_{H+1}^\top f_Y(\mathbf{y}_k)), \tag{4.44}$$

where $\mathbf{y}_k$ is an output candidate of all the possible outputs in a particular task, and $f_Y(\cdot)$ is a feature mapping function. At this time, key-value memories conduct interactions with the output candidates. In the training phase, all the aforementioned feature mapping functions and trainable parameters are optimized in an end-to-end manner. KV-MemNN could also be generalized to a variety of applications with different forms of knowledge by flexibly designing f_Q, f_K, and f_V. For example, for storing world knowledge in the form of a triplet, we can regard the head entity and the relation as the key and the tail entity as the value. For textual knowledge, we can encode sentences or words directly into both key and value in practice.

Although memory networks are proposed for better document modeling, it has profoundly influenced the academic community with this idea. We can use additional modules to store information explicitly and enhance the memory capacity of neural networks. To this day, this framework is still a common idea for modeling very long texts. There are three key points in designing such a network: representation learning of memory, the matching mechanism between memory and query, and how to perform memory retrieval based on the input efficiently.

4.4.2 Hierarchical Document Representation

As mentioned in the former sections, higher-level units in natural languages are often composed of lower-level units, and documents are composed of multiple sentences in a specific logical order. Therefore, an intuitive way to obtain sentence representations is to perform hierarchical modeling [55], where word representations are used to compose sentence representations, which in turn compose document representations. With the powerful representation capabilities of neural networks, we can explicitly develop this type of method. Here, we introduce several neural-based methods of learning document representation hierarchically.

Hierarchical Document Encoder The basic idea of the hierarchical document encoder is to use low-level representations to produce high-level representations. First, the word vectors obtained by pre-training with self-supervised methods can be directly used as the basic word representations. We can also optimize these word representations according to specific tasks. And there are various ways to get the sentence representation through the constituent word representation. For example, we can let it pass through a layer of multilayer perceptron (MLP) and then average over all the hidden states. Here we attempt to recurrently process the document and take LSTM as an example, and the input document is $d = \{s_1, \ldots, s_m\}$, where s_i is a sentence $s_i = \{w_1, \ldots, w_n\}$. We input a sentence s_j directly into LSTM (other neural networks like GRU and CNN can also be applied) and get the corresponding hidden states. In this way, according to the previous equations, the hidden state of each step is calculated from the hidden state of the previous step and the input of the current step:

$$\mathbf{h}_i^w = \text{LSTM}(\mathbf{w}_i, \mathbf{h}_{i-1}^w). \tag{4.45}$$

Thus, the hidden state of the last time step contains the semantic information of the whole sentence and can be used as a sentence representation:

$$\mathbf{s}_j = \mathbf{h}_n^w. \tag{4.46}$$

At this point, we get a representation of each sentence. Considering the sentence as a basic unit, we can build another LSTM on the sentence level to process the sentence representation sequentially. The hidden state at each sentence-level step is determined by the previous hidden state and the current sentence representation input, just like the word-level LSTM:

$$\mathbf{h}_j^s = \text{LSTM}(\mathbf{s}_j, \mathbf{h}_{j-1}^s). \tag{4.47}$$

Repeating the above operation, the hidden state of the last step of this LSTM contains all the information of the sentence representation. It thus can be regarded as a document representation:

$$\mathbf{d} = \mathbf{h}_m^s. \tag{4.48}$$

To this end, we introduce basic hierarchical modeling of document representation. When there is a supervised signal, we can use this document representation directly for neural network training with document-level classification. When there is no supervised signal, we can self-code the document representation, which can be decoded in the reverse order, i.e., first decode the document representation into a sentence representation and then generate words sequentially. The supervised and autoencoding frameworks are illustrated in Figs. 4.11 and 4.12, respectively.

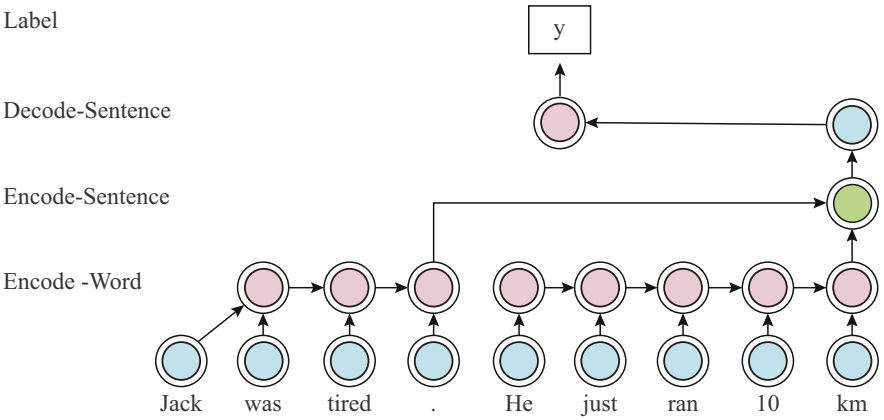


Fig. 4.11 The framework of hierarchical document representation for supervised learning. The figure is a modification of Fig. 2 of the hierarchical autoencoding paper [55]

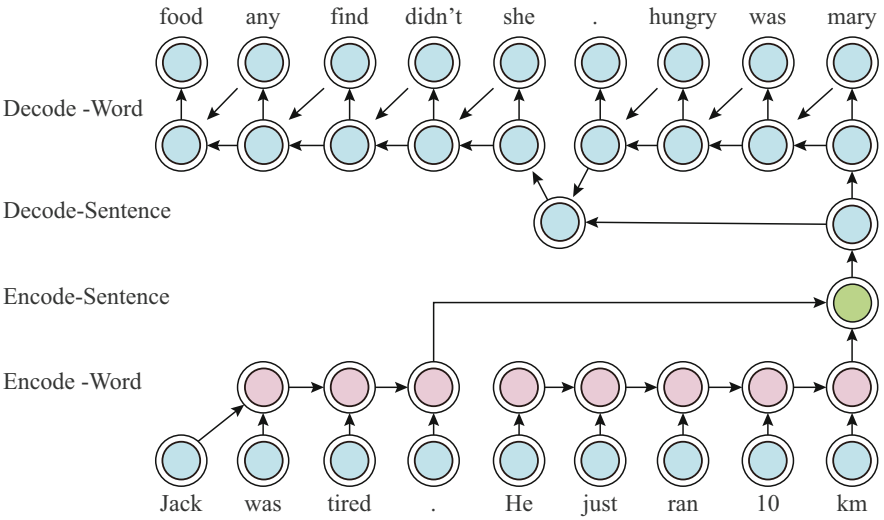


Fig. 4.12 The framework of hierarchical autoencoding of document representation. The figure is re-drawn according to Fig. 2 of the hierarchical autoencoding paper [55]

Hierarchical Attention Network Following the idea of hierarchical modeling, we can make various improvements to the model, such as replacing the LSTM with a more powerful neural network structure and adding attention mechanisms to enhance the transmission of long-dependency information. The hierarchical attention network (HAN [109]) is proposed to use attention mechanisms to capture the hierarchical correlations of documents. The key insight of this model is that while doing hierarchical modeling, different attention weights are assigned to components (words and sentences) using the attention mechanism to learn the

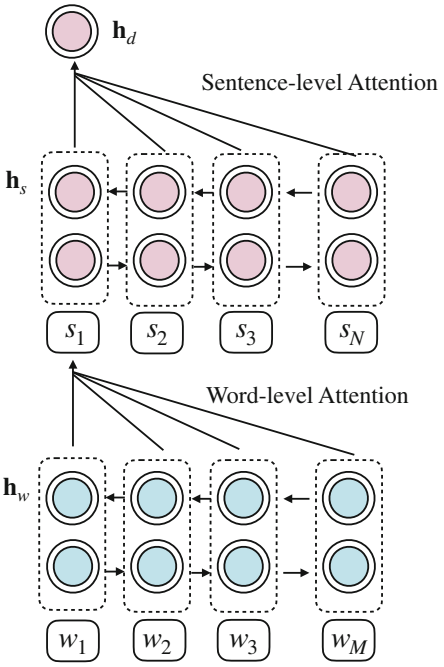


Fig. 4.13 The architecture of the hierarchical attention network. This figure is re-drawn according to Fig. 2 from the hierarchical attention network paper [109]

document's representation dynamically. The framework is illustrated in Fig. 4.13. It is worth noting that the intuition of hierarchical attention networks could be applied to various neural networks. We use GRU, another version of RNNs, as the backbone to introduce the approach.

The first step is also to model the basic linguistic units—words—using a bidirectional GRU to incorporate contextual information. The bidirectional hidden states for a word embedding $\mathbf{w}$ is computed by

$$\overrightarrow{\mathbf{h}}_w = \overrightarrow{\text{GRU}}(\mathbf{w}), \tag{4.49}$$

$$\overleftarrow{\mathbf{h}}_w = \overleftarrow{\text{GRU}}(\mathbf{w}). \tag{4.50}$$

By directly concatenating the two hidden states of both directions, we could obtain the final word representation:

$$\mathbf{h}_w = \text{concat}(\overrightarrow{\mathbf{h}}_w; \overleftarrow{\mathbf{h}}_w). \tag{4.51}$$

Then, following the spirit of hierarchical modeling, we need to construct sentence-level representations. Instead of directly feeding word-level representations to a higher-level neural network, an attention mechanism is adopted to automatically determine how important a word is to the sentence-level represen-

tation. First, we use a one-layer MLP to further extract the feature of one word:

$$\mathbf{u}_w = \tanh(\mathbf{W}\mathbf{h}_w + \mathbf{b}). \tag{4.52}$$

Then, the sentence representation is computed by

$$\mathbf{s} = \sum_i \alpha^i \mathbf{h}_w^i, \tag{4.53}$$

where α is an attention score and is computed by:

$$\alpha^i = \frac{\exp(\mathbf{u}_w^{i\top}\mathbf{u}_w)}{\sum_i \exp(\mathbf{u}_w^{i\top}\mathbf{u}_w)}. \tag{4.54}$$

Now we obtain the sentence representation $\mathbf{s}$. Logically, the foregoing procedures could be analogically applied to the sentence level and obtain the final document representation. We first still use a bidirectional GRU to capture the correlations between sentences:

$$\overrightarrow{\mathbf{h}}_s = \overrightarrow{\mathrm{GRU}}(\mathbf{s}), \tag{4.55}$$

$$\overleftarrow{\mathbf{h}}_s = \overleftarrow{\mathrm{GRU}}(\mathbf{s}). \tag{4.56}$$

Similarly, the hidden state of a sentence is the concatenation of the two directions of hidden states:

$$\mathbf{h}_s = \mathrm{concat}(\overrightarrow{\mathbf{h}}_s; \overleftarrow{\mathbf{h}}_s). \tag{4.57}$$

Then exactly the same neural network and the attention mechanism are applied as follows:

$$\mathbf{u}_s = \tanh(\mathbf{W}\mathbf{h}_s + \mathbf{b}), \tag{4.58}$$

$$\alpha^i = \frac{\exp(\mathbf{u}_s^{i\top}\mathbf{u}_s)}{\sum_i \exp(\mathbf{u}_s^{i\top}\mathbf{u}_s)}, \tag{4.59}$$

$$\mathbf{h}_d = \sum_i \alpha^i \mathbf{h}_s^i. \tag{4.60}$$

Here, we again use the hierarchical spirit, equipped with the attention technique, to construct the document representation $\mathbf{h}_d$. This representation can be fed to an output layer for document-level classification, thereby training the model.

This section introduces two primary frameworks, memory-based and hierarchical approaches, to model documents. As opposed to directly treating documents as longer sentences and then directly applying neural language modeling, such methods more accurately grasp the characteristics of documents with complex structures.

4.5 Applications

Sentence and document representations play a crucial role in multifarious downstream tasks, many of which are the cornerstone tasks of modern information processing. In this section, we introduce typical applications of sentence and document representations in real-world scenarios, which could fall into three groups: classification, sequence labeling, and generation. For classification, we introduce text classification, information retrieval, reading comprehension, open-domain question answering, sequence labeling, its three representative applications, and sequence-to-sequence generation and its typical applications.

4.5.1 Text Classification

Text classification is a typical NLP application that covers many important real-world tasks, such as parsing and semantic analysis. Therefore, it has attracted the interest of many researchers. The conventional text classification models (e.g., the LDA [5] and tree kernel [78] models) focus on capturing more contextual information and correct word order by extracting more useful and distinct features but still expose a few issues (e.g., data sparseness) which has a significant impact on the classification accuracy. With the development of deep learning in the various fields of artificial intelligence, neural models have been introduced into the text classification field, given their abilities of text representation learning. This section will introduce the three typical text classification tasks, including topic classification, sentiment classification, and natural language inference (NLI).

Topic Classification Topic classification aims to assign a sentence to an appropriate category (e.g., type of questions, type of news article), which is a fundamental task of the text classification application. Examples of topic classification are listed in Table 4.1.

Considering the effectiveness of the CNN-based models in capturing sentence semantics, many works use CNNs as representation encoders. The character-level CNN [110] is among the first few works to apply character-level information

Table 4.1 Some examples of topic classification

Sentence	Topic
One of the faculties of Stanford just won a Nobel Prize for her contributions to organic chemistry	`Sci-Tech`
After IPO, the company's share price has risen 147.4% in 2 weeks, and several media outlets are scrambling to cover the news	`Business`
The Golden State Warriors, led by Stephen Curry, won an NBA championship, and now they're eyeing contract extensions for their core players	`Sports`

modeling to topic classification. Increasing the depth of the CNNs [20] helps extract the hierarchical information from scattered characters to whole sentences. MG-CNN [111] captures multiple features from multiple sets of embeddings and concatenates them at the penultimate layer.

RNN-based models, which aim to capture the sequential information of sentences, are also widely used in sentence classification. Recurrent CNN [51] applies a recurrent structure to capture contextual information. Hierarchical attention networks [109] introduce word-level and sentence-level attention mechanisms into an RNN-based model as well as a hierarchical structure to capture the hierarchical information of the document for sentence classification. Combining an LSTM with a CNN [112] also shows better performance on text classification, as it captures both local and global features.

Sentiment Classification Sentiment classification is a particular task of the sentence classification application, whose objective is to determine the sentimental polarities of opinions a piece of text contains, e.g., favorable or unfavorable and positive or negative. This task appeals to the NLP community since it has many potential downstream applications, such as movie review suggestions. Examples of sentiment classification are illustrated in Table 4.2.

Similar to text classification, sentence representation based on neural models has also been widely explored for sentiment classification. Text-CNN [47] utilizes the CNNs trained on top of pre-trained word embeddings and achieves promising results on several sentiment classification datasets. The dynamic CNN model [44] can handle sentences of varying lengths and uses dynamic max-pooling over linear sequences, which could help the model capture both short-range and long-range semantic relations in sentences.

Xavier et al. [29] adopt a stacked denoising autoencoder in sentiment classification. Then, a series of studies based on recursive neural networks are presented to learn sentence representations for sentiment classification, including the recursive autoencoder (RAE) [88], matrix-vector recursive neural network (MV-RNN) [87], and recursive neural tensor network (RNTN) [89]. Besides, Johnson et al. [40] adopt a CNN to learn sentence-level representations and yield promising experimental results in sentiment classification.

The RNN models also benefit sentiment classification as they are able to capture sequential information. Studies [54, 93] investigate tree-structured LSTM

Table 4.2 Some examples of sentiment classification

Sentence	Sentiment
The plot and set design of this movie is breathtaking	`Positive`
He is immersed in sorrow	`Negative`
All the audience who saw the film stood up and clapped their hands, this is a masterpiece that deserves to be watched again and again	`Positive`
This book is written without any rules, and the author is very self-righteous	`Negative`

models on text classification. Hierarchical neural models are proposed to tackle the document-level sentiment classification problem [3, 94], which generate semantic representations at different levels within a document. Besides, an RNN-based multitask learning framework [63] learns across multiple sentence classification tasks and employs three different mechanisms of sharing information to model sentences with task-specific and shared layers. Moreover, the attention mechanism is also introduced into sentiment classification, which aims to determine the importance of each word contributing to the whole sentiment [109].

Natural Language Inference Natural language inference (NLI) is a classification task involving two sentences. Its objective is to determine whether the first sentence entails the second sentence or not. For example, *I was late for class on Monday* entails that *I had a class on Monday*. It could be viewed as a semantic matching problem of two sentences that requires a high-level understanding of sentence-level information. We provide more examples in Table 4.3 to help readers better understand the task. Same as other classification tasks, neural models can automatically learn the two-sentence representations, and a classifier is used for the detection of entailment. The RNN [7] is one of the baseline models for NLI tasks, which derives the representations for both sentences. Apart from using sentence representation directly, some also perform word-level matching to facilitate semantic learning [99]. Kim et al. [46] concatenate features from the attention mechanism with the original hidden states at each layer of RNNs and obtain better performance. Linguistic features like syntactic information [14] are also used to enhance LSTM representation. The recurrent entity network [35] is an entity-centered RNN, which contains several RNN cells, and each cell learns specific entity-related representations. It improves the memory capacity of the original RNN and achieves satisfactory results on NLI tasks.

Table 4.3 Some examples of natural language inference

Premise	Relation	Hypothesis
A cat jumped	`Entailment`	A cat moved
Some cats walked	`Contradiction`	No cats moved
Every cat jumped	`Neutral`	One cat ate
It is nice talking to you all righty	`Neutral`	I talk to you every day
Fun for adults and children	`Contradiction`	Fun only for children
Well it's been very interesting	`Entailment`	It has been very intriguing
You can access the database anytime you want	`Entailment`	The database is accessible to you
He smiled back at me	`Neutral`	He was so happy at that moment

4.5.2 *Information Retrieval*

In the Internet era, information retrieval becomes one of the most critical applications of sentence and document representations. Information retrieval aims to obtain relevant resources from a large-scale collection of information resources. As shown in Fig. 4.14, given the query "William Shakespeare" as input, the search engine (a typical information retrieval application) provides relevant webpages for users. Traditional information retrieval data consists of search queries and document collections D. And the ground truth is available through explicit human judgments or implicit user behavior data such as clickthrough rate.

For the given query q and document d, traditional information retrieval models estimate their relevance through lexical matches. Neural information retrieval models pay more attention to garnering the query and document relevance from semantic matches. Both lexical and semantic matches are essential for neural information retrieval. Thriving from neural network black magic, it helps information retrieval models catch more sophisticated matching features and have achieved the state of the art in the information retrieval task [22].

Neural ranking models typically fall into two groups: representation-based and interaction-based [34]. Studies in the early stage primarily focus on representation-based models. They learn informative representations and match them in the embedding space of queries and documents. On the other hand, interaction-based methods model the query-document matches from the interactions of their terms.

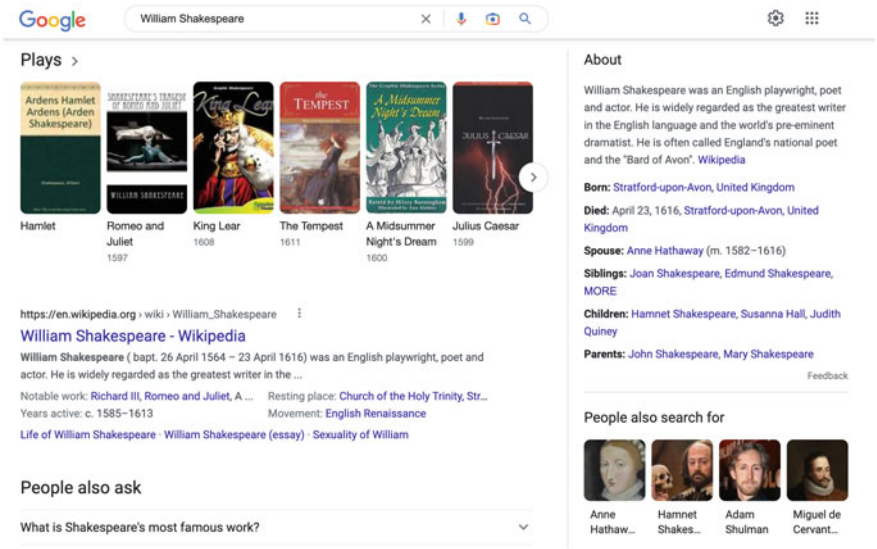


Fig. 4.14 An example of information retrieval. This is a screenshot of the Google search engine

4.5.3 Reading Comprehension

Reading comprehension is crucial to question-answering systems and therefore has been the focus of NLP research. The development of neural-based models has dramatically boosted the performance of reading comprehension. As shown in Fig. 4.15, machine reading comprehension aims to determine the answer given a question and a passage. The task could be viewed as a standard supervised learning task: with a set of training instances, our goal is to learn a mapping that takes the context (i.e., the passage) and related questions as inputs and outputs an answer. The input context can be either a single passage or multiple passages. Intuitively, the longer the provided context is, the more complex the task is. The evaluation metric is typically correlated with the answer type, which will be discussed in the following.

Generally, the current machine reading comprehension task could be divided into four groups according to the answer types [11], i.e., cloze style, multiple-choice, span prediction, and free-form answer.

Cloze Style The cloze style task such as CNN/DAILY MAIL [36] consists of fill-in-the-blank sentences where the question contains a placeholder to be filled in. The answer is either from a predefined candidate set or the vocabulary.

Multiple-Choice The multiple-choice task such as RACE [50] and MCTEST [83] aims to select the best answer from a set of answer choices. It is typical to use accuracy to measure the performance on these two tasks: the percentage of correctly

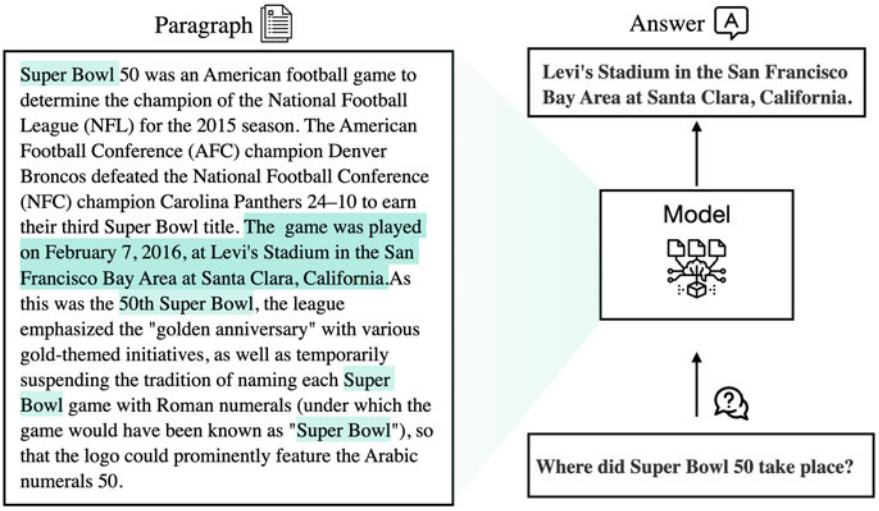


Fig. 4.15 An example of machine reading comprehension from SQuAD [80]

answered questions in the whole example set, since the question could be either correctly answered or not from the given hypothesized answer set.

Span Prediction The span prediction task such as SQuAD [80] is perhaps the most widely adopted task among all since it compromises flexibility and simplicity. It extracts a most likely text span from the passage as the answer to the question, which is usually modeled as predicting the start position and end position of the answer span. We typically use two evaluation metrics proposed by the SQuAD benchmark [80]. The exact match assigns a full score of 1.0 to the predicted answer span if it exactly equals the ground truth answer; otherwise, 0.0. F1-score measures the degree of overlap between prediction and truth by computing a harmonic mean of the precision and recall.

Free-Form Answer The free-form answer task such as MS MARCO [74] does not restrict the answer form or length and is also referred to as *generative question answering*. It is practical to model the task as a sequence generation problem, where the discrete token-level prediction was made. Currently, a consensus on the ideal evaluation metrics has not been achieved. It is common to adopt standard metrics in machine translation and summarization, including ROUGE [58] and BLEU [95].

Since the span prediction format is the most widely researched problem, the following part of this section will be mainly devoted to the mainstream methods in machine reading comprehension with span prediction. With neural networks, the machine reading comprehension system is commonly composed of three consecutive phases: the embedding phase, the reasoning phase, and the prediction phase. Like many other NLP tasks, the embedding phase often adopts pre-trained or contextual word embedding with RNNs, character embedding, or hybrid embeddings. The query and the context are separately encoded. The reasoning phase is responsible for joint learning based on the two representations and is the focus of most works. The prediction phase decides how the output is finally drawn. For extractive mode like span prediction, where a piece of text is extracted from the context, a standard operation is to predict the start position and the end position of the extracted part.

We will mainly introduce the different approaches in the reasoning phase. As shown in Fig. 4.16, while encoding the passage, the model retains the length of the sequence and encodes the question into a fixed-length hidden representation $\mathbf{q}$. The question's hidden vector is then used as a pointer to scan over the passage representation $\{\mathbf{p}_i\}_{i=1}^{n}$ and compute scores on every position in the passage. While maintaining this similar architecture, most machine reading comprehension models vary in the interaction methods between the passage and the question. In the following, we will introduce several classic reading comprehension architectures that follow this paradigm. Most of them merge the two lines of information from the query and the context with the attention mechanism. And they mainly differ in two aspects: the direction of attention and the dimension of attention. Direction refers to whether using only query-to-context attention (as shown in Fig. 4.16) or both directions. Dimension refers to whether attention is only calculated at the

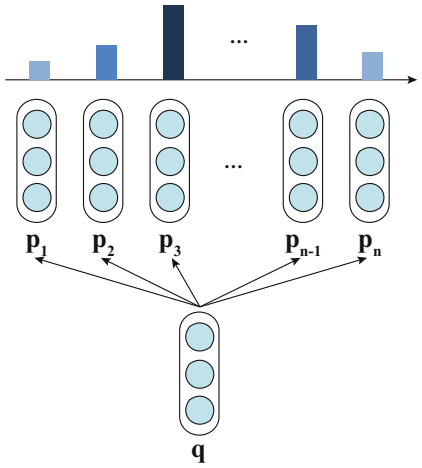


Fig. 4.16 The architecture of classic machine reading comprehension models

sentence representation level, which outputs a single-dimension vector, or at the word embedding level, where output is an embedding matrix.

Single Direction and Single Dimension The first attempt [36] to apply neural networks on machine reading comprehension constructs bidirectional LSTM reader models along with attention mechanisms. The work introduces two reader models, i.e., the attentive reader and the impatient reader. After encoding the passage and the query into hidden states using LSTMs, the attentive reader computes a scalar distribution over the passage tokens and uses it to calculate the weighted sum of the passage's hidden states. The impatient reader extends this idea further by repeatedly updating the weighted sum of passage hidden states after seeing each query token. Following Hermann et al. [36], Chen et al. [12] modify the method to compute attention and simplify the prediction layer in the attentive reader with a simple bilinear term.

Bidirectional Attention and Single Dimension The attention-over-attention reader [21] also computes both query-to-context and context-to-query attention but handles them differently. Instead of simply averaging the token-level query-to-context attention to obtain a final vector for prediction, attention-over-attention computes a weighted vector with a query word importance vector. The word importance vector is computed by averaging the context-to-query attention. This operation is considered to learn the contributions of individual question words explicitly.

Bidirectional Attention Flow and Multi-Dimension Instead of unifying the document and query representation to a single vector with query-to-context attention only, the BiDAF network [85] computes the attentive token representation of both query-to-context and context-to-query at each bidirectional long short-term memory (BiLSTM) layer to allow fine-grained information flow. It consists of the token embedding layer, the contextual embedding layer, the bidirectional attention flow

layer, the LSTM modeling layer, and the Softmax output layer. At each layer, the input is the concatenation of the previous layer's hidden states, the query-to-context representation, and the context-to-query representation. The representation of multiple granularities and a bidirectional attention flow can fully capture the interaction between document and query for start and end position prediction.

The gated-attention reader [25] adopts the gated-attention module, where each token representation of the passage is scaled by the attended query vector after each BiGRU layer. This gated-attention mechanism allows the query to interact directly with the token embeddings of the passage at the semantic level. And such layer-wise interaction enables the model to learn conditional token representation given the question at different representation levels.

4.5.4 Open-Domain Question Answering

Open-domain QA (OpenQA) [33] aims to answer open-domain questions utilizing external resources such as collections of documents [98], webpages [15, 49], structured knowledge graphs [2, 6], or automatically extracted relational triples [28]. Recently, with the development of machine reading comprehension techniques [12, 25, 86, 102], researchers attempt to answer open-domain questions via performing reading comprehension on plain texts with neural-based models [13]. As illustrated in Fig. 4.17, a neural-based OpenQA system usually retrieves relevant articles or paragraphs of the question from a large-scale corpus (e.g., Wikipedia). It then generates answers from these texts by a reading comprehension model introduced in the last section. Open-domain question answering essentially combines two critical applications: information retrieval and reading comprehension.

The system [13], namely, DrQA, is composed of two modules: (1) one document retriever module to retrieve relevant articles or paragraphs and (2) one document reader to produce the final answers from the extracted articles.

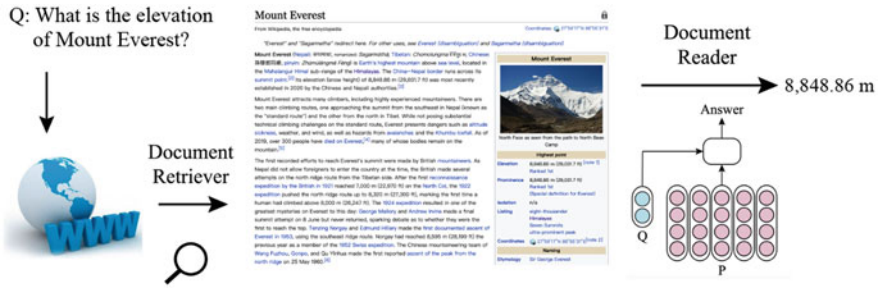


Fig. 4.17 An example of open-domain question answering. This figure is re-drawn according to Fig. 1 in the DrQA paper [13]

The document retriever is used as a first quick skim to narrow the search space and focus on potentially relevant documents. The retriever builds TF-IDF weighted bag-of-words vectors for the documents and the questions and computes similarity scores for ranking. The retriever uses bigram counts with hash to further utilize local word order information while ensuring speed and memory efficiency. The document reader model takes in the top five Wikipedia articles yielded by the document retriever and extracts the final answer to the question. The document reader predicts an answer span with a confidence score for each article. The final prediction is made by maximizing the unnormalized exponential prediction scores across the documents.

Given each document, the document reader first builds a feature representation for each word in the document, which is often the concatenation of the following components: (1) Word embeddings: The pre-trained word embeddings like GloVe embeddings pre-trained on Wikipedia. (2) Manual features: The manual features combined with part-of-speech (POS) and named entity recognition tags and normalized term frequencies (TF). (3) Exact match: This feature indicates whether the word in the document can be precisely matched to one question word. (4) Aligned question embeddings: This feature aims to encode a soft alignment between words in the document and the question in the word embedding space.

Then the feature representation of the document is fed into a multilayer bidirectional LSTM (BiLSTM) to encode the contextual representation. For the question, the contextual representation is simply obtained by encoding the word embeddings using a multilayer BiLSTM. After that, the contextual representation is aggregated into a fixed-length vector using self-attention. In the answer prediction phase, the start and end probability distributions are calculated following the paradigm mentioned in the strategy in Sect. 4.5.3.

Despite its success, the DrQA system is prone to noise in retrieved texts which may hurt the performance of the system. Hence, several approaches [18, 100] are proposed to attempt to tackle the noise problem in DrQA by using two separate procedures for question answering: paragraph selection and answer extraction. However, they both only select the most relevant paragraph among all retrieved paragraphs to extract answers and may lose valuable information distributed in other paragraphs.

Wang et al. [101] adopt strength-based and coverage-based methods for re-ranking, aggregating the answers that existing methods retrieved from all the paragraphs. Nevertheless, the challenge of noisy data is still unsolved. To address this issue, a coarse-to-fine denoising OpenQA model [60] is developed to the first screen out relevant paragraphs and then retrieve correct answers.

4.5.5 Sequence Labeling

Sequence labeling is a classic application in natural language processing. In this paradigm, given an input sequence $\{w_1, \ldots, w_n\}$, we need to assign a label y_i

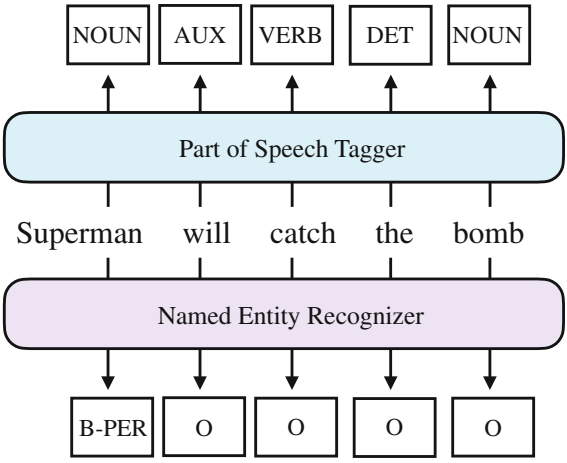


Fig. 4.18 An example of sequence labeling

to each token w_i. Part-of-speech (POS) tagging and named entity recognition (NER) are the two most representative sequence labeling tasks. Sequence labeling requires the model to capture the correlations of words in the sequence accurately. Hence, classic approaches use probabilistic graphical models (PGM) to represent the dependency structure of different words. Modern methods use powerful deep neural networks to produce richer representations and adopt conditional random field (CRF) or direct token-level classification to conduct sequence labeling [38]. In addition to these two tasks, word segmentation of languages without delimiters (e.g., Chinese) is typically treated as a sequence labeling task [26, 66, 107] (Fig. 4.18).

Part-of-Speech (POS) Tagging POS tagging aims to assign part-of-speech tags to each word in a given piece of text, including nouns, verbs, adjectives, etc. Some tags might be evident and static (e.g., proper nouns), while most words are polysemy, and their part-of-speech attributes are context-dependent. For example, the word "record" can be either a noun or a verb. Early on, Brill et al. [8] propose rule-based methods that highly rely on expert knowledge and extraction of rich linguistic features in syntax, morphology, and lexicon. Classical statistical models like the hidden Markov model (HMM) [41] model the probability of tags given words in a context-aware manner. Modern neural networks are based on contextual representations of words and parameterize the predicted probability with a conditional random field (CRF) layer and a simple MLP classifier head. CNNs and RNNs are common backbones used for feature extraction [67, 77].

Named Entity Recognition (NER) In NER, we need to identify if a word in an input sequence is a *named entity*, a term that could specifically indicate a real-world object. Typical named entity types include *Person, Organization, Location*, etc. A named entity could be one word or a phrase with multiple words. Hence, in this task, a BIO label schema is universally adopted, where a word could be classified at the beginning of an entity (B), inside an entity (I), and outside an

entity (O). Final entity prediction is extracted based on the word assigned tags, and evaluation is conducted at the entity level [27, 103]. Feature-based methods extract word-level and character-level features and adopt classic classification models for prediction. Bike et al. [4] and Mcnamee et al. [68] propose an HMM-based and support vector machine (SVM)-based NER system, respectively. Deep learning methods allow for richer feature representation. Apart from using pre-trained word embeddings like skip-gram, a series of works [16, 52, 56, 82] also learn character-level features and incorporate them with word representations for better performance. The bidirectional LSTM-CNN [16] encodes character-level features with a CNN and word-level features with a BiLSTM. The bidirectional LSTM-CRF model [38] also adds other features, including spelling features, context features, and gazetteer features, to enhance final representations in a BiLSTM-CRF model.

4.5.6 Sequence-to-Sequence Generation

Sequence-to-sequence generation refers to a group of tasks that require sequence generation based on an input sequence, including machine translation, text summarization, question generation, etc. A famous model structure for sequence-to-sequence problems is an encoder-decoder structure, where the model is composed of an encoder and a decoder. The encoder encodes the input source language $S = \{s_1, s_2, \ldots, s_n\}$ and passes the encoded representation to the decoder. The decoder decodes and outputs tokens in target language $T = \{t_1, t_2, \ldots, t_m\}$ based on encoder output. More specifically, output tokens are typically generated in an autoregressive manner, i.e., each t_i is generated depending on the previously generated tokens $\{t_1, t_2, \ldots, t_{i-1}\}$. Both structures are trained in an end-to-end fashion with parallel training data. Below is a formalized training objective for a sequence-to-sequence problem:

$$\arg\max \prod_{i=1}^{m} P(t_i | t_{j<i}, S) \tag{4.61}$$

Metrics First, it is essential to learn the commonly used metrics to evaluate a sequence-to-sequence system.

BLEU [75] is an adjusted precision calculation based on the count of n-grams. First, it extracts all n-grams in the output sequence. Then, it calculates the sum of occurrences of these n-grams in the reference sequence (i.e., the correct translation) against the total number of n-grams in the output sequence. For example, if the output is *the cat cat* and the reference is *the cat jumps*, all 2 grams in the output is *"the cat," cat cat* and the total number of their occurrence in the reference is 1 (*the cat'*). So the score of 2-gram will be $p_2 = \frac{1}{2} = 0.5$. BLEU also takes a brevity penalty (BP) that penalizes the mismatch of output and reference length. Suppose we set a range for the number of grams involving the calculation as $[1, N]$, the

BLEU score is

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{i=1}^{N} w_i \log p_i\right), \tag{4.62}$$

$$\text{BP} = \begin{cases} 1 & c > r \\ e^{(1-r/c)} & c \leq r, \end{cases} \tag{4.63}$$

where w_i is a weight and can be set to $\frac{1}{N}$, c is the length of the output sequence, and r is the length of the reference sequence.

ROUGE [58] is a group of metrics often used in evaluating text summarization systems. ROUGE-N (most commonly ROUGE-1 and ROUGE-2) calculates the recall of n-grams. So take the example above; we can get ROUGE-1 $= 2/3$ and ROUGE-2 $= 1/2$. ROUGE-L concerns the ratio of the length of the longest common subsequence against the reference length. In the example above, ROUGE-L $= 2/3$.

Next, we introduce some representative models in machine translation and text summarization.

Machine Translation Machine translation aims to translate texts in one language into another language while retaining their semantic meanings. While traditional rule-based and statistical machine translation systems require abundant expert knowledge and often fail to capture meaning from context to handle polysemy, the development of deep neural networks has inspired neural machine translation systems and achieved competitive performance.

Kalchbrenner et al. [43] use a one-dimensional CNN as the encoder and a single-layer RNN as the decoder. Cho et al. [17] enhance the alignment scores calculation between phrases with an RNN encoder-decoder structure and improve on the traditional statistical machine translation system. Sutskever et al. [92] adopt a deep LSTM encoder-decoder.

GNMT [105] is the first NMT system put into production. It has an eight-layer LSTM encoder and 8-layer LSTM decoder, and the first layer of the encoder is bidirectional. The attention mechanism is also applied to the output of the encoder. In terms of decoding, it also adds coverage penalty and length normalization to encourage the generation of longer and high-quality sentences. And the Transformer [96], an encoder-decoder neural network, is proposed initially as a sequence-to-sequence model and used on the machine translation task. The model then achieves the new state-of-art performance on benchmark datasets compared to models based on LSTM.

Text Summarization Text summarization takes a long passage as its input and generates a relatively short one that summarizes the key points in the original passage. It is worth noting that typically sequence-to-sequence models can be

simultaneously applied to machine translation and text summarization since the task is of the same form.

Pointer-generator network [84] is one of the most classical text summarization models that combine LSTM-attention-based encoder-decoder with pointer network [97]. The basic structure contains a single-layer bidirectional LSTM encoder with attention and a single-layer LSTM decoder. Apart from the standard encoder-decoder pipeline, it applies an extra pointer while decoding. The pointer depends on the encoder output, the current decoder hidden states, and decoder input and calculates a probability p_{gen} indicating how much we favor the decoder generated results. The final distribution from which the next token is drawn is a weighted sum of distribution given by the decoder and distribution given by attention weights of the encoder output, each weighted by p_{gen} and $1 - p_{\text{gen}}$. So the pointer serves as a mediator between generated tokens and copied tokens from the original input. It is especially beneficial for text summarization as copying original words from the input can help keep the semantics on the right track.

4.6 Summary and Further Readings

This chapter introduces basic concepts, methodologies, and applications of sentence and document representation learning, which encode sentences and documents into real-valued representation vectors. We first introduce the symbolic representation for sentences and probabilistic language models. Then we extensively introduce several neural language models, including adopting feed-forward neural networks, convolutional neural networks, recurrent neural networks, and Transformers for language models. We further introduce document representation learning methods, including memory-based and hierarchical approaches. Finally, we introduce several typical applications of sentence and document representation. Sentence and document representations provide an effective way of downstream tasks utilizing high-level semantic information and have significantly improved the performances of these tasks. For further understanding of sentence representation learning and its applications, there are also some recommended surveys and books that introduce neural network methods [30, 42], sentence representation methods [57], and Transformers [59].

More recently, pre-trained language models based on deep Transformers show state-of-the-art performance in this area. Meanwhile, it also spawns particular research issues of sentence and document representation learning. We will introduce and discuss this topic in the next chapter. In addition, the use of more efficient neural network architectures, the establishment of a more stable and universal representation of long text, and the development of a comprehensive evaluation approach are worthy research topics in this field.

Acknowledgments Zhiyuan Liu, Yankai Lin, and Maosong Sun designed the overall architecture of this chapter; Ning Ding and Yankai Lin drafted the chapter. Zhiyuan Liu and Yankai Lin proofread and revised this chapter.

We also thank Zhengyan Zhang, Cunchao Tu, Hongyin Luo, Zhenghao Liu, and Haozhe Ji for providing the initial materials for the first edition. And we thank Ganqu Cui, Yuan Yao, Shi Yu, Yulin Chen, Xingtai Lv, and Suyuan Zhao for proofreading this chapter.

This is the sentence and document representation learning chapter of the second edition of the book *Representation Learning for Natural Language Processing*, with its first edition published in 2020 [65]. As compared to the first edition of this chapter, the main changes include the following: (1) we merged the sentence representation and document representation to a new chapter by restructuring the organization, (2) we polished and rewrote the content of the methods and applications, and (3) we added new methods for document representation learning.

References

1. Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3, 2003.
2. Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of EMNLP*, 2013.
3. Parminder Bhatia, Yangfeng Ji, and Jacob Eisenstein. Better document-level sentiment analysis from rst discourse parsing. In *Proceedings of EMNLP*, 2015.
4. Daniel M. Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: a high-performance learning name-finder. In *Fifth Conference on Applied Natural Language Processing*, 1997.
5. David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
6. Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*, 2015.
7. Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of EMNLP*, 2015.
8. Eric Brill. A simple rule-based part of speech tagger. In *Proceedings of ANLP*, 1992.
9. Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.
10. Sarath Chandar, Sungjin Ahn, Hugo Larochelle, Pascal Vincent, Gerald Tesauro, and Yoshua Bengio. Hierarchical memory networks. *arXiv preprint arXiv:1605.07427*, 2016.
11. Danqi Chen. *Neural Reading Comprehension and Beyond.* PhD thesis, Stanford University, 2018.
12. Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. In *Proceedings of ACL*, 2016.
13. Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *Proceedings of ACL*, 2017.
14. Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced LSTM for natural language inference. In *Proceedings of ACL*, 2017.
15. Tongfei Chen and Benjamin Van Durme. Discriminative information retrieval for question answering sentence selection. In *Proceedings of EACL*, 2017.
16. Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *Transactions of the association for computational linguistics*, 4:357–370, 2016.

17. Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of EMNLP*, 2014.
18. Eunsol Choi, Daniel Hewlett, Jakob Uszkoreit, Illia Polosukhin, Alexandre Lacoste, and Jonathan Berant. Coarse-to-fine question answering for long documents. In *Proceedings of ACL*, 2017.
19. Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *Proceedings of ICML*, 2015.
20. Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. In *Proceedings of EACL*, 2017.
21. Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-over-attention neural networks for reading comprehension. In *Proceedings of ACL*, 2017.
22. Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proceedings of WSDM*, 2018.
23. Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of ICML*, 2017.
24. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.
25. Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. In *Proceedings of ACL*, 2017.
26. Ning Ding, Dingkun Long, Guangwei Xu, Muhua Zhu, Pengjun Xie, Xiaobin Wang, and Hai-Tao Zheng. Coupling distant annotation and adversarial training for cross-domain chinese word segmentation. In *Proceedings ACL*, 2020.
27. Ning Ding, Guangwei Xu, Yulin Chen, Xiaobin Wang, Xu Han, Pengjun Xie, Hai-Tao Zheng, and Zhiyuan Liu. Few-nerd: A few-shot named entity recognition dataset. In *Proceedings of ACL*, 2021.
28. Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. Open question answering over curated and extracted knowledge bases. In *Proceedings of KDD*, 2014.
29. Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of ICML*, 2011.
30. Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.
31. Irving J Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3–4):237–264, 1953.
32. Joshua Goodman. Classes for fast maximum entropy training. In *Proceedings of ASSP*, 2001.
33. Bert F Green Jr, Alice K Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *Proceedings of IRE-AIEE-ACM*, 1961.
34. Jiafeng Guo, Yixing Fan, Qingyao Ai, and W.Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of CIKM*, 2016.
35. Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. In *Proceedings of ICLR*, 2017.
36. Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Proceedings of NeurIPS*, 2015.
37. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.
38. Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.
39. Zhiheng Huang, Geoffrey Zweig, and Benoit Dumoulin. Cache based recurrent neural network language model inference for first pass speech recognition. In *Proceedings of ICASSP*, 2014.
40. Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. In *Proceedings of ACL-HLT*, 2015.

41. Nisheeth Joshi, Hemant Darbari, and Iti Mathur. Hmm based pos tagger for hindi. In *Proceeding of AISC*, 2013.
42. Dan Jurafsky and James H Martin. Speech and language processing. 3rd, 2022.
43. Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of EMNLP*, 2013.
44. Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of ACL*, 2014.
45. Slava Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401, 1987.
46. Seonhoon Kim, Inho Kang, and Nojun Kwak. Semantic sentence matching with densely-connected recurrent and co-attentive information. In *Proceedings of AAAI*, 2019.
47. Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of EMNLP*, 2014.
48. Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of ICML*, 2016.
49. Cody Kwok, Oren Etzioni, and Daniel S Weld. Scaling question answering to the web. *TOIS*, pages 242–262, 2001.
50. Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.
51. Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Proceedings of AAAI*, 2015.
52. Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *Proceedings of NAACL*, 2016.
53. Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of ICML*, 2014.
54. Jiwei Li, Minh-Thang Luong, Dan Jurafsky, and Eduard Hovy. When are tree structures necessary for deep learning of representations? In *Proceedings of EMNLP*, 2015.
55. Jiwei Li, Thang Luong, and Dan Jurafsky. A hierarchical neural autoencoder for paragraphs and documents. In *Proceedings of ACL*, 2015.
56. Peng-Hsuan Li, Ruo-Ping Dong, Yu-Siang Wang, Ju-Chieh Chou, and Wei-Yun Ma. Leveraging linguistic structures for named entity recognition with bidirectional recursive neural networks. In *Proceedings of EMNLP*. Association for Computational Linguistics, 2017.
57. Ruiqi Li, Xiang Zhao, and Marie-Francine Moens. A brief overview of universal sentence representation methods: A linguistic view. *ACM Computing Surveys (CSUR)*, 55(3):1–42, 2022.
58. Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.
59. Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *arXiv preprint arXiv:2106.04554*, 2021.
60. Yankai Lin, Haozhe Ji, Zhiyuan Liu, and Maosong Sun. Denoising distantly supervised open-domain question answering. In *Proceedings of ACL*, 2018.
61. Fei Liu, Trevor Cohn, and Timothy Baldwin. Improving end-to-end memory networks with unified weight tying. In *Proceedings of ALTA*, 2017.
62. Fei Liu and Julien Perez. Gated end-to-end memory networks. In *Proceedings of EACL*, 2017.
63. Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. In *Proceedings of IJCAI*, 2016.
64. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
65. Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation Learning for Natural Language Processing*. Springer, 2020.

66. Ji Ma, Kuzman Ganchev, and David Weiss. State-of-the-art chinese word segmentation with bi-lstms. In *Proceedings of EMNLP*, 2018.
67. Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of ACL*, 2016.
68. Paul McNamee and James Mayfield. Entity extraction without language-specific resources. In *Proceedings of COLING*, 2002.
69. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of ICLR*, 2013.
70. Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proceedings of InterSpeech*, 2010.
71. Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. In *Proceedings of EMNLP*, 2016.
72. Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of ICML*, 2012.
73. Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of AISTATS*, 2005.
74. Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.
75. Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of ACL*, 2002.
76. Michael J Pazzani and Daniel Billsus. *Content-based recommendation systems*. Springer, 2007.
77. Barbara Plank, Anders Søgaard, and Yoav Goldberg. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of ACL*, 2016.
78. Matt Post and Shane Bergsma. Explicit and implicit syntactic features for text classification. In *Proceedings of ACL*, 2013.
79. Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.
80. Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*, 2016.
81. Juan Ramos. Using tf-idf to determine word relevance in document queries. In *Proceedings of ICML*, 2003.
82. Marek Rei, Gamal Crichton, and Sampo Pyysalo. Attending to characters in neural sequence labeling models. In *Proceedings of COLING*, 2016.
83. Matthew Richardson, Christopher JC Burges, and Erin Renshaw. MCTest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of EMNLP*, 2013.
84. Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of ACL*, 2017.
85. Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *Proceedings of ICLR*, 2017.
86. Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. ReasoNet: Learning to stop reading in machine comprehension. In *Proceedings of KDD*, 2017.
87. Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of EMNLP*, 2012.
88. Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of EMNLP*, 2011.
89. Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, 2013.

90. Daniel Soutner, Zdeněk Loose, Luděk Müller, and Aleš Pražák. Neural network language model with cache. In *Proceedings of ICTSD*, 2012.
91. Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Proceedings of NeurIPS*, volume 28, 2015.
92. Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Proceedings of NeurIPS*, 2014.
93. Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of ACL*, 2015.
94. Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of EMNLP*, 2015.
95. Ahmet Cüneyd Tantuğ, Kemal Oflazer, and Ilknur Durgar El-Kahlout. Bleu+: a tool for fine-grained bleu computation. In *Proceedings of LREC*, 2008.
96. Ashish Vaswani, Noam Shazeer, Niki Parmar, Llion Jones, Jakob Uszkoreit, Aidan N Gomez, and Lukasz Kaiser. Attention is all you need. In *Proceedings of NeurIPS*, 2017.
97. Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proceedings of NeurIPS*, 2015.
98. Ellen M Voorhees et al. The trec-8 question answering track report. In *Proceedings of TREC*, 1999.
99. Shuohang Wang and Jing Jiang. Learning natural language inference with LSTM. In *Proceedings of NAACL*, 2016.
100. Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerald Tesauro, Bowen Zhou, and Jing Jiang. R3: Reinforced ranker-reader for open-domain question answering. In *Proceedings of AAAI*, 2018.
101. Shuohang Wang, Mo Yu, Jing Jiang, Wei Zhang, Xiaoxiao Guo, Shiyu Chang, Zhiguo Wang, Tim Klinger, Gerald Tesauro, and Murray Campbell. Evidence aggregation for answer re-ranking in open-domain question answering. In *Proceedings of ICLR*, 2018.
102. Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of ACL*, 2017.
103. Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, et al. Ontonotes release 5.0 ldc2013t19. *Linguistic Data Consortium, Philadelphia, PA*, 23, 2013.
104. Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.
105. Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
106. Caiming Xiong, Stephen Merity, and Richard Socher. Dynamic memory networks for visual and textual question answering. In *Proceedings of ICML*, 2016.
107. Nianwen Xue. Chinese word segmentation as character tagging. In *Proceddings of IJCL*, 2003.
108. Tianyu Yang and Antoni B Chan. Learning dynamic memory networks for object tracking. In *Proceedings of ECCV*, 2018.
109. Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of NAACL-HLT*, 2016.
110. Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Proceedings of NeurIPS*, 2015.
111. Ye Zhang, Stephen Roller, and Byron C Wallace. MGNC-CNN: A simple approach to exploiting multiple word embeddings for sentence classification. In *Proceedings of NAACL-HLT*, 2016.
112. Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and F. Lau. A c-lstm neural network for text classification. *ArXiv*, abs/1511.08630, 2015.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Chapter 5
Pre-trained Models for Representation Learning

Yankai Lin, Ning Ding, Zhiyuan Liu, and Maosong Sun

Abstract Pre-training-fine-tuning has recently become a new paradigm in natural language processing, learning better representations of words, sentences, and documents in a self-supervised manner. Pre-trained models not only unify semantic representations of multiple tasks, multiple languages, and multiple modalities but also emerge high-level capabilities approaching human beings. In this chapter, we introduce pre-trained models for representation learning, from pre-training tasks to adaptation approaches for specific tasks. After that, we discuss several advanced topics toward better pre-trained representations, including better model architecture, multilingual, multi-task, efficient representations, and chain-of-thought reasoning.

5.1 Introduction

Representation learning is the critical component of machine learning systems, which aims to learn informative representations of objects from large-scale data. With the learned representations, machine learning systems thus can handle multiple tasks, languages, and modalities more flexibly and desirable. Representation learning for natural language processing (NLP) can be divided into three stages according to the learning paradigm: statistical learning, deep learning, and pre-trained models, with the paradigm shift of representation from symbolic representation to distributed representation.

Statistical learning started early in the 1940s [39, 93]. It requires domain experts to design task-specific rules according to their knowledge to transfer raw data into

Y. Lin
Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China
e-mail: yankailin@ruc.edu.cn

N. Ding · Z. Liu (✉) · M. Sun
Department of Computer Science and Technology, Tsinghua University, Beijing, China
e-mail: dingn18@mails.tsinghua.edu.cn; liuzy@tsinghua.edu.cn; sms@tsinghua.edu.cn

© The Author(s) 2023

Z. Liu et al. (eds.), *Representation Learning for Natural Language Processing*,
https://doi.org/10.1007/978-981-99-1600-9_5

task-related representation task-by-task. This makes representation learning based on statistical learning fragmented in multiple granularities of text and multiple tasks. Later, distributed representation learning with deep learning techniques [45] was developed with larger datasets, more computing power, and advanced neural architectures. It utilizes deep neural networks such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to extract task-related representations automatically. It makes an initial step toward unified representation learning: although deep learning still results in various model parameters for multiple tasks, the learned representations can be transferred to multiple NLP tasks, and the same neural network architecture can be applied to model the data of various NLP tasks.

Recently, pre-trained models (PTMs) for representation learning [7, 20], also known as foundation models [6], have become a new trend in NLP. As shown in Figs. 5.1 and 5.2, compared with conventional representation learning techniques, the pre-training-fine-tuning paradigm of PTMs enables them to learn unified representations for multiple tasks, languages, and modalities. Moreover, big PTMs have shown high-level capabilities like human beings. In the following, we introduce the new characteristics of PTMs in detail.

(1) **Pre-training-Fine-Tuning Paradigm.** Transfer learning [103] enables the knowledge (usually stored in model parameters) learned from one task/domain to be transferred to help the learning of other tasks/domains in the same model architecture. Inspired by the idea of transfer learning, PTMs learn general task-agnostic representations via self-supervised learning from large-scale unlabeled data and then adapt their model parameters to downstream tasks by task-specific fine-tuning. Different from conventional deep learning techniques that only learn from task-specific supervised data, the self-supervised pre-training objective enables PTMs to learn from larger unlabeled web-scale data without labeled task signals automatically. With the pre-training-fine-tuning pipeline,

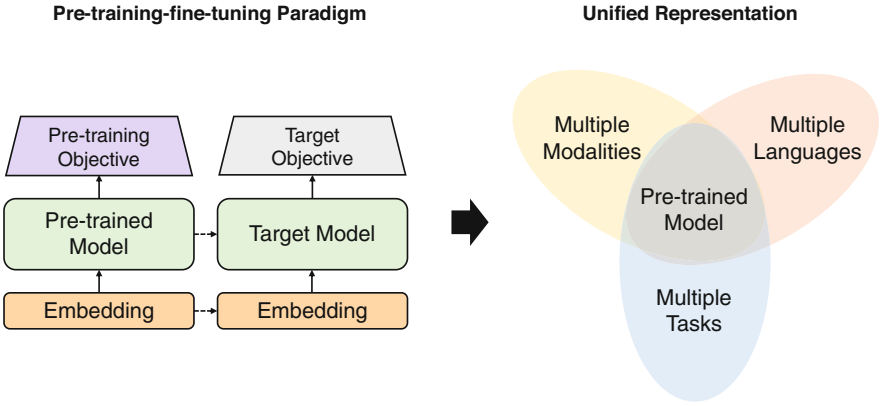


Fig. 5.1 Pre-training-fine-tuning paradigm and unified representation of pre-trained models

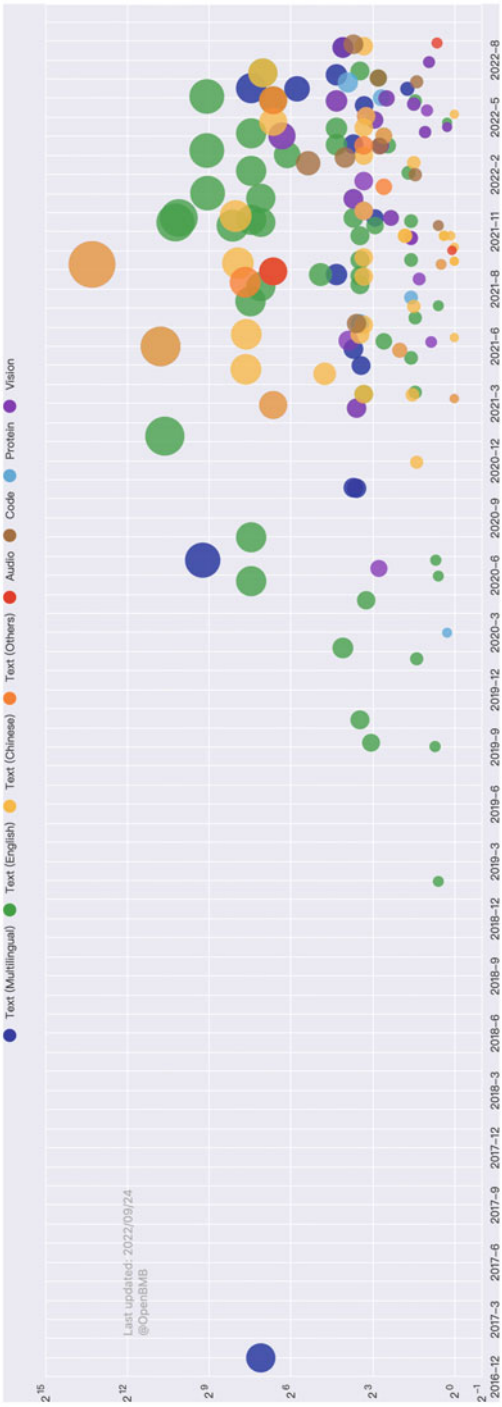


Fig. 5.2 The development trends of big PTMs. The size of the circles indicates the model scale. The figure is obtained from the official website of OpenBMB (https://openbmb.github.io/BMList/)

PTMs learn general knowledge in the self-supervised pre-training and then stimulate task-related knowledge to complete the downstream tasks through model adaptation.

(2) **Unified Representation.** The development of the Transformer architecture [104] unifies the encoders of multiple entries such as text, image, and video. Based on the Transformer architecture and the pre-training-fine-tuning paradigm, PTMs unify the paradigm for representation learning from three perspectives: task, language, and modality. A unified representation learned by pre-trained models can be adapted and utilized for multiple downstream tasks, multiple languages, and even multiple modalities.

(3) **Larger Models with Novel Capabilities.** With more training data and computation power available, constructing larger PTMs has become a new trend in representation learning research. We demonstrate the development trends of big PTMs in Fig. 5.2. As model sizes go larger, PTMs emerge with many fantastic abilities approaching human beings. For example, big PTMs can perform in-context learning [7], which learns the downstream tasks with a task instruction and some optional examples as the additional input text. Big PTMs can also perform chain-of-thought reasoning [112], which mimics the intuitive thought process from human-written reasoning chains as the additional input text, and behavior learning [72] which learns the human behavior such as operating search engine. It indicates that PTMs are quickly evolving into more intelligent agents than we have ever imagined.

In this section, we mainly introduce text-based PTMs since the fantastic Transformer-based PTMs for representation learning begin from NLP, leaving the introduction of the PTMs for graph in Chap. 6, multi-modality in Chap. 7, and knowledge in Chaps. 9, 10, 11, and 12. In the rest of this chapter, we first introduce pre-training tasks in Sect. 5.2, including word-level and sentence-level pre-training tasks. After that, we present how to adapt PTMs to downstream tasks, including full-parameter fine-tuning, delta tuning, and prompt learning in Sect. 5.3. Note that we only discuss the PTMs with pre-training-fine-tuning paradigm in this section, while the feature-based PTMs have been discussed in Chapter refchap:word. Finally, we overview four advanced topics, including better model architecture, multilingual learning, multi-task learning, efficient representations, and chain-of-thought reasoning in Sect. 5.4.

5.2 Pre-training Tasks

As introduced in Sect. 5.1, PTMs for representation learning typically consist of two phases: pre-training and adaptation (fine-tuning). During pre-training, PTMs learn the task-agnostic representations, which aim to capture the text's lexical, syntactic,

Table 5.1 A list of typical pre-trained models in NLP

Model	Architecture	Size
BERT [20]	Encoder	340 M
RoBERTa [68]	Encoder	340 M
SpanBERT [49]	Encoder	340 M
UniLM [23]	Encoder	340 M
ELECTRA [15]	Encoder	340 M
XLM [17]	Encoder	340 M
KnowBERT [76]	Encoder	340 M
K-BERT [66]	Encoder	340 M
ERNIE (Tsinghua) [132]	Encoder	110 M
ERNIE (Baidu) [100]	Encoder	340M
ELMo [75]	Decoder	–
GPT [86]	Decoder	340 M
XLNET [117]	Decoder	340 M
GPT-2 [87]	Decoder	1.5 B
CPM-1 [133]	Decoder	2.6 B
GPT-3 [7]	Decoder	175 B
GLM-130B [127]	Decoder	130 B
BART [59]	Encoder-decoder	340 M
T5 [88]	Encoder-decoder	11 B
CPM-2 [131]	Encoder-decoder	11 B
mT5 [115]	Encoder-decoder	13 B
OPT [129]	Encoder-decoder	175 B
Switch-Transformer [26]	Encoder-decoder	1.6 T

semantic, and discourse knowledge as well as the world and commonsense knowledge hiding in the text. The typical form of PTMs can be divided into three types: encoder-based PTMs, decoder-based PTMs, and encoder-decoder-based PTMs. We list typical PTMs in Table 5.1. Based on existing pre-training tasks designed for these PTMs, we conclude two major categories of existing pre-training tasks from them, including word-level and sentence-level pre-training tasks.

5.2.1 Word-Level Pre-training

Word-level pre-training tasks aim to learn contextualized word representations for PTMs from the large-scale unlabeled corpus. As discussed in Chap. 2, contextualized word representations can generate different representations according to different contexts, aiming to capture the lexical meaning of a word as well as the syntactic and semantic relations with its context words. Next, we present multiple widely used word-level pre-training objectives, including casual language

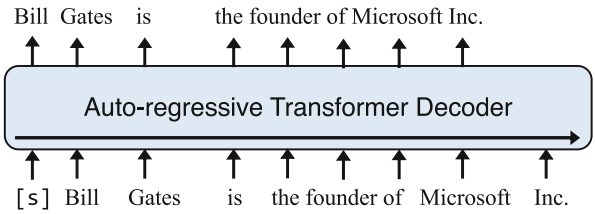


Fig. 5.3 The casual language model objective

modeling, masked language modeling (MLM), replaced language modeling (RLM), and denoising language modeling (DLM).

Casual Language Modeling (CLM) CLM is the most typical form of language modeling task. It is widely adopted as the pre-training objective of decoder-based PTMs such as GPT [86], which utilizes an auto-regressive Transformer decoder to model the probability of the input text. As shown in Fig. 5.3, CLM feeds the whole input text sequence into the Transformer decoder word-by-word auto-regressively and then asks the model to predict the next word at each position. Formally, given the input text $s = (w_1, w_2, \ldots, w_N)$ with N words, the pre-training objective of CLM is formulated as:

$$\mathscr{L}_{CLM} = -\sum_{i=1}^{N} \log P(w_i | w_0, w_1, \ldots, w_{i-1}), \tag{5.1}$$

where w_0 is the start token `[s]` of the sentence and $P(w_i|w_0, w_1, \ldots, w_{i-1})$ is the probability of w_i modeled conditioned on the historical context generated by an auto-regressive Transformer decoder. In fact, CLM is widely used as the pre-training task when training big PTMs due to its effectiveness and efficiency.

Although CLM can learn the contextualized word representations simply and effectively, it can only encode the historical information in one direction in language understanding tasks. Hence, the downstream language understanding applications usually concatenate the word representations of left-to-right and right-to-left Transformer decoders learned by CLM, which can naturally combine the contextual information from both directions.

Masked Language Modeling (MLM) MLM is another widely used word-level pre-training objective for PTMs. MLM believes that the contextual information of a word is not a simple combination of the historical information captured by a left-to-right model and the future information captured by a right-to-left model. When generating a word, a deep bidirectional Transformer model should be utilized to consider both historical and future information. However, casual language modeling cannot be directly applied in pre-training the deep bidirectional Transformer model since it suffers from information leakage brought by the self-attention operation. Therefore, as shown in Fig. 5.4, MLM first masks out part of the words with `[MASK]` token in the input text and then asks the model to predict the masked words

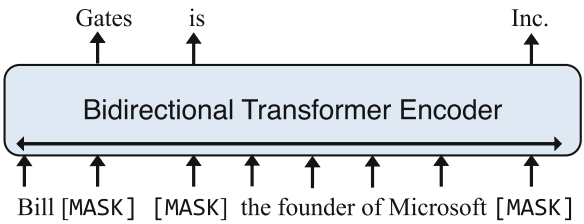


Fig. 5.4 The masked language model objective

according to the remaining unmasked context. Formally, we denote the masked words as $s_{\text{mask}} = (w_{m_1}, \ldots, w_{m_M})$ where m_i is the index of the masked word and M is the number of the masked words and the masked sequence as $\bar{s}$. The pre-training objective of MLM is formulated as:

$$\mathscr{L}_{\text{MLM}} = -\sum_{w \in s_{\text{mask}}} \log P(w|\bar{s}). \tag{5.2}$$

MLM was first adopted by BERT [20] and later used by many other PTMs. To address the gap between pre-training and adaptation phases caused by the introduction of the `[MASK]` token, BERT further utilizes a masking strategy: for a randomly selected word to be masked, BERT replaces it with (1) the `[MASK]` token with an 80% probability, (2) the original word with a 10% probability, and (3) a random word with a 10% probability. A major limitation of word-level masking is that it may not sufficiently capture the linguistic and knowledge information at the span level, such as phrases and named entities. Span-level semantics are important for many downstream NLP tasks, such as named entity recognition and entity linking. Hence, SpanBERT [49] and ERNIE (Baidu) [100] further introduce a novel masking strategy for MLM: span-based masking. Span-based masking proposes to mask contiguous random spans instead of individual random words. The PTMs can better encode the span-level semantics in their learned representations by predicting the entire masked spans.

Although MLM can take advantage of the superior power of a bidirectional Transformer encoder, its pre-training objective can only cover part of the input text, e.g., BERT only masks 15% input words. The reason is that it must ensure that the contextual information in the remaining unmasked words is sufficient to recover the masked words to some extent. Hence, the training efficiency of MLM is lower than that of CLM, which predicts every input word. MLM requires more training steps for convergence.

Replaced Language Modeling (RLM) RLM is then proposed to improve the training efficiency of MLM, which is first adopted by ELECTRA [15]. RLM proposes to replace part of the words in random positions of the input text and then asks the model to predict which positions are replaced words. As shown in Fig. 5.5, RLM trains the PTMs in an adversarial manner. It uses a smaller bidirectional encoder as the generator, which generates replaced words that are

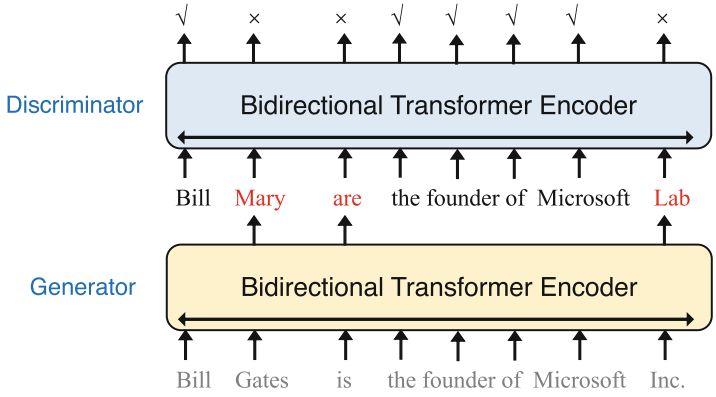


Fig. 5.5 The replaced language modeling objective

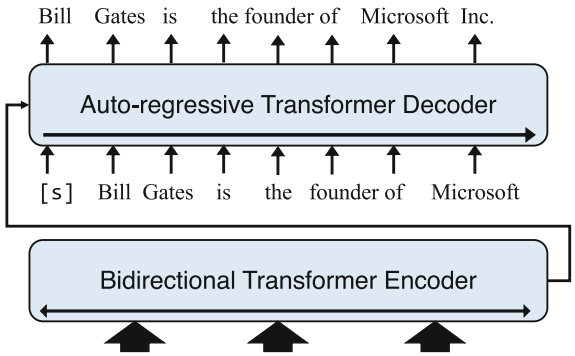
harder to be discriminated against. And then, it regards the PTMs as a discriminator to distinguish the replaced words from other unreplaced words. Hence, RLM can cover its pre-training objective in all input words. Let $\bar{s}$ denote the corrupted input text after random word replacement, and we can define the pre-training objective of RLM as:

$$\mathscr{L}_{\text{RLM}} = -\sum_{i=1}^{N} \log P(y_i|\bar{s}), \tag{5.3}$$

where y_i is the predicted label indicating whether the i-th word is replaced or not.

Denoising Language Modeling (DLM) DLM can cover nearly all the pre-training forms introduced above. It is widely used in encoder-decoder-based PTMs, which contain a bidirectional Transformer encoder and an auto-regressive Transformer decoder. With the encoder-decoder architecture, DLM allows more modifications to the input text sequence, which can help PTMs capture more lexical, syntactic, and semantic knowledge from the text. As shown in Fig. 5.6, DLM randomly modifies the input text in several strategies [59, 88, 97]:

- *Word Masking* is to mask parts of the words in the input text. This strategy is the corresponding form of the MLM pre-training task in DLM, ensuring that DLM can make the PTMs capture the information learned by MLM.
- *Text Infilling* is to mask a contiguous span of the input text with a single `[MASK]` token. It can be viewed as a harder version of word masking, where PTMs require learning to predict how many words the masked span originally has instead of only distinguishing whether the word is masked. It is similar to the span-level masking strategy of MLM.
- *Word Deletion* is to delete parts of the words from the input text randomly. It requires PTMs to decide which words have been deleted.



Word Masking: Bill [MASK] [MASK] the founder of Microsoft [MASK]
Text Filling: Bill [MASK] the founder of Microsoft [MASK]
Word Deletion: Bill the founder of Microsoft
Word Permutation: Gates Bill is Microsoft the founder of Inc.

Fig. 5.6 The denoising language model objective

- *Word Permutation* is to shuffle all words from the input text in random order. It requires PTMs to understand the syntactic and semantic relations between words as well as the whole meaning of the sentence to recover the sentence order.

Besides the above word-level modifications, DLM also allows sentence-level modifications, such as *sentence permutation* and *document rotation*. The sentence- and document-level modifications help the word representation learned by PTMs capture high-level semantics, such as the discourse relations between different sentences. We will discuss the sentence-level pre-training tasks in the next subsection.

Let $\bar{s}$ denote the corrupted input text by applying several above input text modification strategies on s. The pre-training objective of DLM is then formulated as:

$$\mathscr{L}_{\text{DLM}} = -\sum_{i=1}^{N} \log P(w_i|\bar{s}, w_0, w_1, \ldots, w_{i-1}), \tag{5.4}$$

where $P(w_i|\bar{s}, w_0, w_1, \ldots, w_{i-1})$ is the conditional probability of w_i, which is modeled with an encoder-decoder Transformer model.

5.2.2 Sentence-Level Pre-training

As discussed in Chap. 4, sentence representations are essential for many downstream NLP tasks such as information retrieval, question answering, machine translation, etc. Sentence-level pre-training aims to learn sentence representation

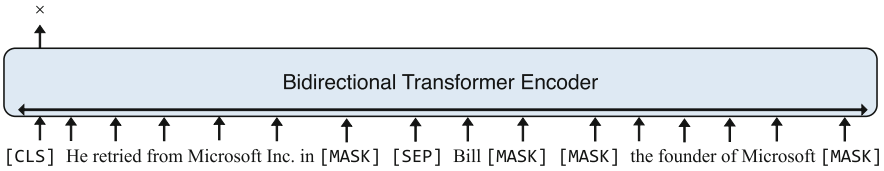


Fig. 5.7 The next sentence prediction objective

which can capture the global meanings of sentences as well as the relationships between sentences for PTMs. In this subsection, we introduce three typical sentence-level pre-training tasks for PTMs, including the next sentence prediction (NSP), sentence order prediction (SOP), and sentence contrastive learning (SCL) tasks.

Next Sentence Prediction (NSP) NSP is the first self-supervised pre-training objective to learn sentence representation for PTMs. As shown in Fig. 5.7, for the sentence s, NSP adds a `[CLS]` token in the front of the sentence and utilizes the representation of `[CLS]` as the sentence representation. After that, NSP adds another sentence s' at the end of s with a token `[SEP]` to indicate the sentence boundary. s' can be the next sentence of s in the document or a randomly selected sentence from the pre-training corpus. NSP aims to determine whether s' and s appear consecutively in the original text, which may help PTMs understand the relationship between sentences. Formally, NSP's pre-training objective can be formulated as:

$$\mathscr{L}_{\text{NSP}} = -\log P(y|s, s'), \tag{5.5}$$

where y is the predicted label indicating whether s' is the next sentence of s or not. In practice, BERT utilizes a uniform sampling strategy, i.e., choosing s' with (1) the original next sentence of s with a half chance and (2) the randomly selected sentence with a half chance.

NSP is adopted by BERT, which claims that NSP can help PTMs capture sentence-level semantics. However, RoBERTa [68] reimplements BERT and surprisingly finds that the performance of PTMs on most downstream NLP tasks is even better by removing the NSP objective and only pre-training on the MLM objective. ALBERT [55] further points out that the lack of task difficulty of NSP may be the key reason for its ineffectiveness. In fact, due to the big difference in topic distribution between the original next sentence and the randomly selected sentence, NSP usually suffers from a shortcut, i.e., it just requires PTMs to perform topic prediction. It is easy and already partly covered by the MLM objective.

Sentence Order Prediction (SOP) SOP is proposed to avoid the problem of NSP in modeling inter-sentence relationships. SOP also adds a `[CLS]` token in front of the sentence to obtain the sentence representation. After that, SOP randomly swaps the two consecutive sentences and asks the PTMs to predict the proper orders. In this way, the instances of SOP with correct or wrong sentence orders do not differ explicitly in topic distribution. Hence, SOP forces PTMs to distinguish discourse-level coherence relations between two input sentences rather than their topics. Formally, the objective of SOP can be formulated similarly to the NSP objective:

$$\mathscr{L}_{\text{SOP}} = -\log P(y|s, s'), \tag{5.6}$$

where y is the predicted label indicating whether s' and s are in order or not. The experimental results on several downstream tasks of ALBERT show that SOP can somewhat solve the problem of NSP, which may come from analyzing misaligned coherence cues.

Sentence Contrastive Learning (SCL) SimCSE [29] introduces sentence contrastive learning to pre-train the PTMs. Unlike NSP and SOP, which learn the sentence-level semantics by distinguishing the relations between different raw sentences, SimCSE simply predicts whether two input sentences are the same. The basic idea of SimCSE is that the representations of a sentence with different dropout masks should be closer than representation of other sentences. Formally, as shown in Fig. 5.8, let and $\mathbf{s}^{z'}$ denote the sentence representations of sentence s with dropout mask z and z', respectively. We can define the pre-training objective of SimSCE as:

$$\mathscr{L}_{\text{SimCSE}} = -\log \frac{\exp(\cos(\mathbf{s}^{z}, \mathbf{s}^{z'}))}{\sum_{i=1}^{M} \exp(\cos(\mathbf{s}^{z}, \mathbf{s}_i))}, \tag{5.7}$$

where $\mathbf{s}_i$ is the representation of the i-th negative sentence in the training batch, $\cos(\cdot)$ indicates the cosine similarity, and M is the batch size. In practice, the negative sentences are usually sampled from the same mini-batch for convenience. Although SimCSE is strikingly simple, it outperforms the NSP and SOP pre-training objectives by a large margin in a series of downstream NLP tasks [29]. Other concurrent works also adopt the idea of sentence contrastive learning for sentence-level pre-training, such as self-guidance contrastive learning [52], contrastive tension [46], and TSDAE [107].

Besides word-level and sentence-level pre-training tasks, knowledge-level pre-training tasks have also been widely explored to help PTMs better capture the world knowledge hiding behind the text. We will introduce how to pre-train PTMs at the knowledge level in Chap. 9.

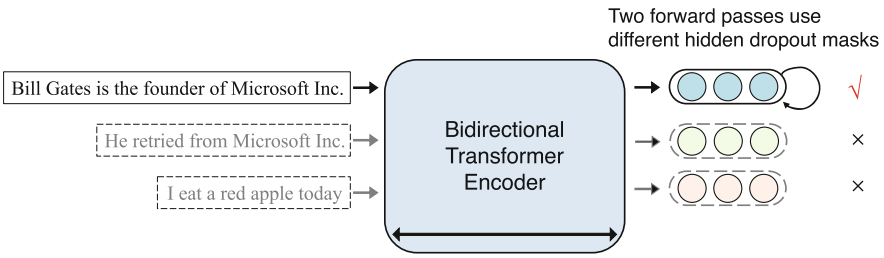


Fig. 5.8 The SimCSE objective. The figure is redrawn according to Fig. 1 from SimCSE paper [29]

5.3 Model Adaptation

Through self-supervised pre-training on the large-scale unlabeled corpus, PTMs have learned a strong ability to understand language and thus can generate task-agnostic informative representations. Then for downstream NLP tasks, it is natural to introduce task-specific objectives to adapt the PTMs, aiming to directionally stimulate the specific functionality of PTMs and obtain task-specific text representation. Now, the remaining question is how to adapt big PTMs to target downstream tasks effectively and efficiently. We introduce the model adaptation methods from full-parameter fine-tuning to optimization-efficient delta tuning and data-efficient prompt learning. In fact, between pre-training and model adaptation, some works also explore to pre-adapt the PTMs with multi-task learning or domain-specific learning. We remain the introduction of model preadaptation in Sect. 5.4.

5.3.1 Full-Parameter Fine-Tuning

Full-parameter fine-tuning is the most straightforward solution for adapting PTMs to downstream tasks. Full-parameter fine-tuning tunes all parameters of PTMs with the guidance of task-specific data, aiming to stimulate the task-specific abilities of PTMs. Given a PTM model $\Theta = \{\theta_1, \theta_2, \ldots, \theta_{|\Theta|}\}$ and the training data D of the downstream task, the goal of fine-tuning phase can be formulated as finding the parameter updates $\Delta\Theta$:

$$\Delta\Theta = \nabla f_{\Theta}(D), \tag{5.8}$$

where $f_{\Theta}(D)$ is the adaptation objective of the downstream task. That is, we can simply feed the task-specific inputs into PTMs and fine-tune all the parameters so that the parameters of PTMs for the downstream task can be obtained by $\Theta' = \Theta - \Delta\Theta$.

Now, the remaining problem is how to define the adaptation objective $f_\Theta(D)$. It can be divided into three categories according to the downstream task types: classification, sequence labeling, and generation.

Classification Classification is one of the typical forms of NLP tasks, such as topic classification, sentiment classification, natural language inference, etc. Formally, given the input sentence s and the output label y, the classification task models the conditional probability $P(y|s)$. As shown in Figs. 5.9, 5.10, and 5.11, a common solution to fine-tune PTMs is to add a task-specific classifier on the top of the sentence/document representation generated by PTMs, i.e., $P(y|s) = P(y|\mathbf{s})$. As for the sentence/document representation $\mathbf{s}$, we usually use (1) the representation of the `[CLS]` token for encoder-based PTMs, (2) the representation of the last word in the sentence for decoder-based PTMs, and (3) the representation of the start word in the Transformer-based decoder for encoder-decoder-based PTMs. Besides adding an external classifier, decoder-based and encoder-decoder-based PTMs also model

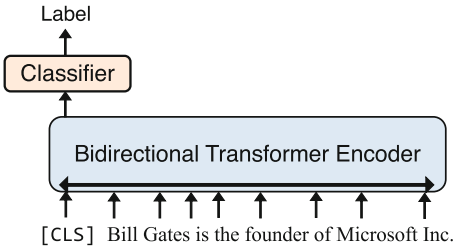


Fig. 5.9 The adaptation form of encoder-based PTMs for classification tasks

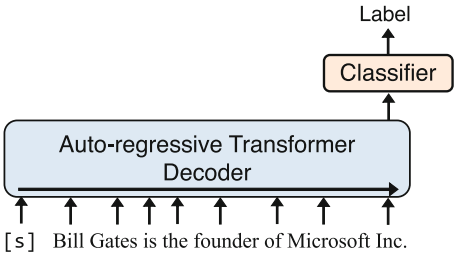


Fig. 5.10 The adaptation form of decoder-based PTMs for classification tasks

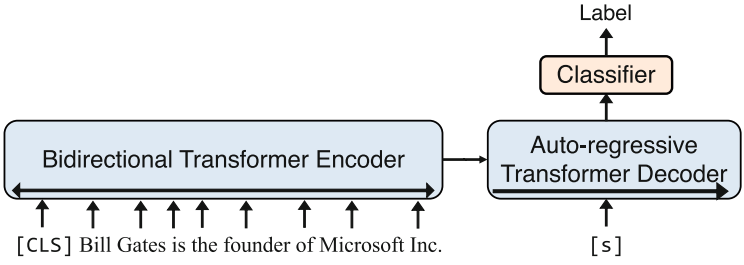


Fig. 5.11 The adaptation form of encoder-decoder-based PTMs for classification tasks

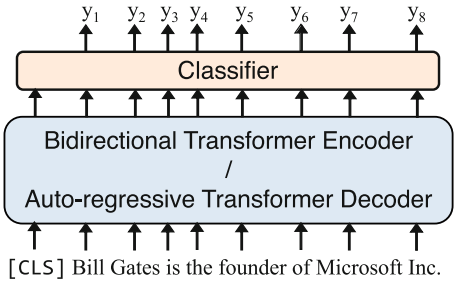


Fig. 5.12 The adaptation form of PTMs for sequence labeling tasks

the classification tasks as text generation, which directly generate the target labels in the decoder.

Sequence Labeling Sequence labeling is also a classical NLP task format, such as part-of-speech tagging, named entity recognition, etc. Formally, given the input sentence $s = (w_1, \ldots, w_N)$ and the corresponding output labels $y = (y_1, \ldots, y_N)$ for all words, the sequence labeling task models the conditional probabilities $P(y|s)$. It is usually modeled as the word-level classification form, in which the output labels of all words are conducted independently, i.e., $P(y|s) = \prod P(y_i|s)$. As shown in Fig. 5.12, we can add a task-specific classifier on top of the output representation $\mathbf{h}_i$ for the i-th word generated by either the bidirectional Transformer encoder (e.g., encoder-based PTMs and encoder-decoder-based PTMs) or the auto-regressive Transformer decoder (e.g., decoder-based PTMs), i.e., $P(y_i|s) = P(y_i|\mathbf{h}_i)$. Except for the basic word-level classification form, we can also regard the sequence labeling task as a generation task, i.e., directly generating the whole label sequence.

Generation As we have introduced in Chap. 4, many typical NLP tasks are in text generation form, such as machine translation, summarization, etc. Formally, given the source sentence s and the corresponding target sentence t, the generation task models the conditional probability $P(t|s)$ (for the language modeling task, we only model $P(t)$ without any condition). As shown in Fig. 5.13, for decoder-based PTMs, we can directly feed the input text into the auto-regressive Transformer decoder and ask it to generate the target sentence after the input text continually. As shown in Fig. 5.14, for encoder-decoder-based PTMs, we can feed the text into the bidirectional Transformer encoder and ask the auto-regressive Transformer decoder to generate the target sentence.

Fine-tuning the whole PTMs is simple and effective, showing superior performance in a wide range of downstream NLP tasks. However, performing full-parameter fine-tuning has two significant drawbacks. First, it is time- and resource-consuming, especially considering the growing model scale. Nowadays, researchers [7, 88] have revealed that the performance of PTMs can be continually improved as the PTMs get larger and the increasing scale has become an irreversible trend for developing PTMs. Full-parameter fine-tuning requires the PTMs to update all the model parameters during adaptation and storing the whole model for each

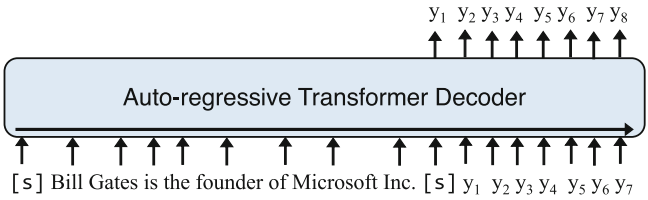


Fig. 5.13 The adaptation form of decoder-based PTMs for generation tasks

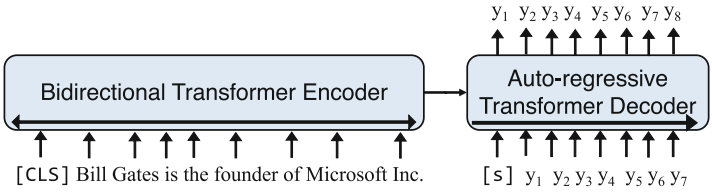


Fig. 5.14 The adaptation form of encoder-decoder-based PTMs for generation tasks

task. Second, it is hard to generalize from a few examples and thus still requires considerable training examples in the downstream tasks for model adaptation. In fact, when taking a closer look at model adaptations, we can find the gap between pre-training and full-parameter fine-tuning. Hence, this raises a new question: **how can we adapt PTMs more effectively?** Therefore, delta tuning and prompt learning target these two problems from **model optimization perspective** and **data utilization perspective**, respectively. We will introduce them as follows.

5.3.2 Delta Tuning

Delta tuning (a.k.a., parameter-efficient tuning) [22] proposes to only update part of the model parameters instead of full-parameter updating for adapting PTMs to downstream tasks, which improves the model adaptation from the optimization perspective. The basic assumption of delta tuning is that we can stimulate the necessary abilities for downstream tasks by only modifying a few model parameters. Formally, different from full-parameter fine-tuning that the number of updated parameters $|\Delta\Theta|$ is equal to the number of whole model parameters $|\Theta|(\Theta = \theta_1, \theta_2, \ldots, \theta_n)$, delta tuning only updates a small number of parameters while achieving the same adaptation objectives. From the perspective of representation learning, the general representations obtained by self-supervised pre-training can be adapted to task-specific representations with little cost.

We classify existing delta tuning methods into three main categories [22]: addition-based, specification-based, and reparameterization-based methods, as shown in Fig. 5.15. In this subsection, we detail these three types of delta tuning approaches.

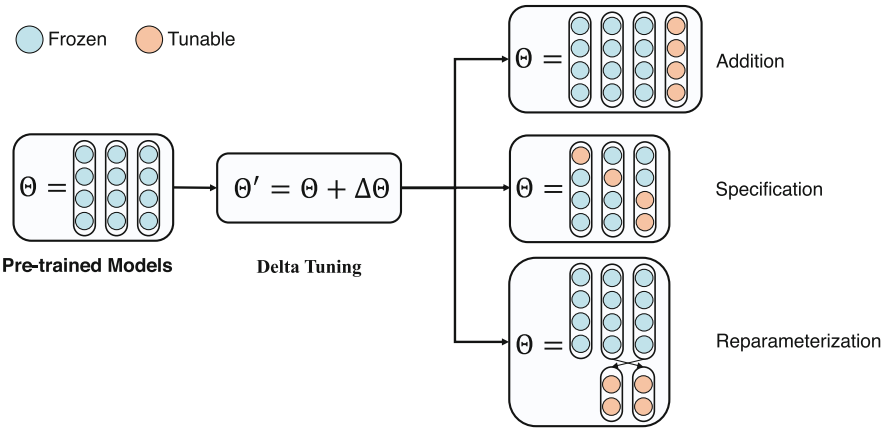


Fig. 5.15 The overall architecture of delta tuning. The figure is redrawn according to Fig. 4 from delta tuning paper [22]

Addition-Based Approach Addition-based approach keeps all the parameters in the original PTMs frozen and inserts new trainable neural modules or parameters (denoted as $\Delta\Theta = \Theta_{add} = \{\theta_{n+1}, \theta_{n+2}, \ldots, \theta_{n+m}\}$ for tuning the downstream tasks). In practice, we have $m \ll n$ in the addition-based methods. In the following, we introduce two typical addition-based methods: adapter-based and prefix tuning.

Adapter-Based Methods Adapter-based methods insert tiny neural adapter modules into the middle of Transformer layers. It only tunes the parameters of the inserted adapter while keeping the PTMs frozen to adapt the model for downstream tasks. Vanilla adapter [42] first utilizes a two-layer feed-forward network as adapters and achieves comparable performance compared with full-parameter fine-tuning in a lot of downstream NLP tasks. As shown in Fig. 5.16, for an output hidden representation $\mathbf{h} \in \mathbb{R}^d$ of a PTM module, vanilla adapter first feeds $\mathbf{h}$ into a down-projection network which projects it into r-dimensional semantic space with a transform matrix $\mathbf{W}_{down} \in \mathbb{R}^{d \times r} (r < d)$ and then feeds the output into an up-projection network which projects it back to d-dimensional space with a transform matrix $\mathbf{W}_{up} \in \mathbb{R}^{r \times d}$. The process of vanilla adapter can be formulated as:

$$\mathbf{h} \leftarrow f(\mathbf{h}\underline{\mathbf{W}_{down}})\underline{\mathbf{W}_{up}} + \mathbf{h}, \tag{5.9}$$

where $\Delta\Theta = [\mathbf{W}_{down}, \mathbf{W}_{up}]$ are the tunable parameters (we highlight them by red color and underline) and $f(\cdot)$ is a nonlinear activation function.

In practice, the adapter modules are inserted in the middle of two Transformer blocks in the PTMs, and it can reduce the number of tunable parameters of PTMs to about 0.5–8%. Moreover, AdapterDrop [90] further proposes to dynamically remove adapter modules from lower Transformer layers to further reduce the computational cost for model inference.

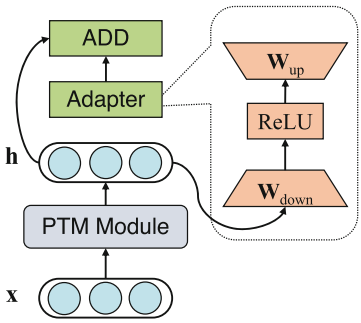


Fig. 5.16 The illustration of adapter-based tuning methods

After the vanilla adapter, recent works continue to explore better forms of adapter modules. For example, Compacter [50] further reduces the number of tunable parameters of the adapter module with parameterized hypercomplex multiplication layer. Formally, it replaces the original projection matrix with the sum of the Kronecker products of two low-rank matrices:

$$\mathbf{W}_{up} = \sum_{i=1}^{l} \mathbf{A}_i \otimes \mathbf{B}_i, \tag{5.10}$$

where $\mathbf{A}_i \in \mathbb{R}^{l \times l}$ and $\mathbf{B}_i \in \mathbb{R}^{(d/l) \times (r/l)}$ and $\otimes$ indicates the Kronecker product operation. The formulation of $\mathbf{W}_{down}$ is similar. The experimental results of Compacter [50] show that it can effectively reduce the number of tunable parameters in the adapter modules into $\frac{1}{l}$ without hurting the model performance in the downstream tasks.

Although existing adapter-based methods can achieve the performance of nearly full-parameter fine-tuning with fewer modified parameters, it still requires back-propagation through the whole PTM. To address this issue, Ladder side tuning [101] further proposes to move the adapter modules out of the Transformer architecture of PTMs, bridging a ladder outside the backbone model. Hence, it can effectively save computation of backpropagation of the original PTMs while updating adapter modules and also save memory by shrinking the hidden size of representations.

Prefix Tuning Methods Prefix tuning [62] adds trainable prefix vectors to the hidden states at each layer instead of inserting adapter modules in the middle of the Transformer layers. Formally, as shown in Fig. 5.17, prefix tuning can be viewed as concatenating two prefix matrices $\mathbf{P}_K, \mathbf{P}_V \in \mathbf{R}^{l \times d}$ (l is the number of the inserted prefix vectors in the prefix matrix) to the input key hidden matrix K and value hidden matrix V of the multi-head attention layer, which is formulated as:

$$\mathbf{h}_i = \text{ATT}(\mathbf{x}\mathbf{W}_Q^i, \text{concat}(\underline{\mathbf{P}_K^i}; \mathbf{x}\mathbf{W}_K^i), \text{concat}(\underline{\mathbf{P}_V^i}; \mathbf{x}\mathbf{W}_V^i)), \tag{5.11}$$

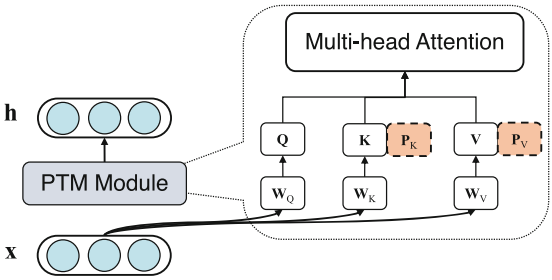


Fig. 5.17 The illustration of the prefix tuning method

where $\mathbf{P}_K^i$ and $\mathbf{P}_V^i$ are the i-th sub-vectors of $\mathbf{P}_K$ and $\mathbf{P}_V$ for i-th attention head's calculation, ATT$(\cdot)$ indicates the self-attention function, and $\mathbf{x}$ is the input feature of Transformer blocks. For prefix tuning, we have $\Delta\Theta = \mathbf{P}_K \cup \mathbf{P}_V$. Empirically, directly optimizing $\mathbf{P}_K$ and $\mathbf{P}_V$ may be unstable and hurt the performance slightly, and thus prefix tuning proposes to reparametrize them with feed-forward neural networks:

$$\begin{aligned} \mathbf{P}_K &= \text{MLP}(\mathbf{P}'_K), \\ \mathbf{P}_V &= \text{MLP}(\mathbf{P}'_V), \end{aligned} \tag{5.12}$$

and they only save $\mathbf{P}_K$ and $\mathbf{P}_V$ after training.

Prompt tuning [58] is a simplified form of prefix-tuning, which only adds prefix vectors (a.k.a., soft prompts) to the input layer instead of all layers. It shows that prompt tuning can achieve nearly the same performance as full-parameter fine-tuning when the model size increases. A significant limitation of prefix tuning approaches is that their extremely small parameter spaces make them challenging to optimize and thus require more training time to converge compared to full-parameter fine-tuning. This phenomenon is more severe in small-scale PTMs. Gu et al. [32] thus propose to pre-train the representations of soft prompt tokens in the pre-training stage. The experimental results demonstrate that pre-training soft prompts can effectively improve the performance of prompt learning in downstream tasks and even outperform full-parameter fine-tuning.

In summary, both prefix tuning and adapter-based methods insert new trainable parameters to learn the downstream tasks, and their major difference is the position of the inserted parameters.

Specification-Based Approach Specification-based approach proposes to specify part of the model parameters in the original PTMs to be tunable (denoted as $\Delta\Theta = \{\Delta\theta_{idx_1}, \Delta\theta_{idx_2}, \ldots, \Delta\theta_{idx_m}\}$ where $idx_i \in [1, n]$ is the index of tunable parameters) and also $m \ll n$.

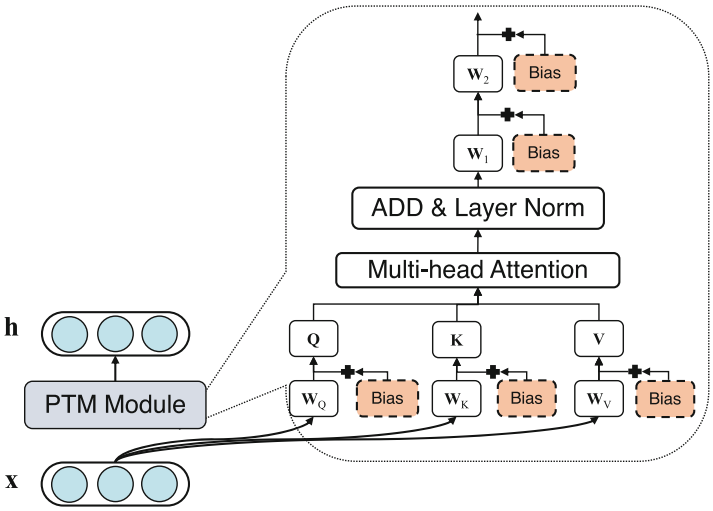


Fig. 5.18 The illustration of the BitFit method

BitFit [125] proposes to only optimize the bias terms inside the PTMs while freezing other parameters. Formally, as shown in Fig. 5.18, BitFit first specifies the multi-head attention layer in the Transformer block as:

$$\mathbf{h}_i = \text{ATT}(\mathbf{x}\mathbf{W}_Q^i + \mathbf{b}_Q^i, \mathbf{x}\mathbf{W}_K^i + \mathbf{b}_K^i, \mathbf{x}\mathbf{W}_V^i + \mathbf{b}_V^i), \tag{5.13}$$

and then specifies the next feed-forward layer as:

$$\mathbf{h}' = \text{GeLU}(\mathbf{h}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2, \tag{5.14}$$

where $\text{GeLU}(\cdot)$ indicates the Gaussian error linear unit [41]. We do not show the layer-norm layers for convenience, but their bias terms are also tunable in BitFit. Experimental results in BitFit show that it can achieve over 95% performance as full-parameter fine-tuning on several benchmarks. They also find that different functionalities may be controlled by different parts of specified bias terms during model adaptation.

Besides BitFit which directly specifies the bias term to be tuned, diff pruning [34] proposes to learn to select part of the model parameters for model adaptation. The basic idea of diff pruning is to encourage the delta parameter $\Delta\Theta$ to be as sparse as possible. To this end, Diff pruning first decomposes $\Delta\Theta$ into a binary mask vector $\mathbf{z} \in \{0, 1\}^{|\Theta|}$ multiplied with a dense vector $\mathbf{w} \in \mathscr{R}^{|\Theta|}$:

$$\Delta\Theta = \mathbf{z} \odot \mathbf{w}, \tag{5.15}$$

and then it optimizes an expectation with respect to $\mathbf{z}$ under a Bernoulli distribution parameter α:

$$\min_{\alpha,\mathbf{w}} \mathscr{E}_{\mathbf{z}\sim p(\mathbf{z};\alpha)}[\mathscr{L}(\Theta + \Delta\Theta) + \lambda\|\Delta\Theta\|_0], \tag{5.16}$$

where $\mathscr{L}(\cdot)$ indicates the learning objective of the downstream task and the L_0-norm penalty is added to achieve the goal of sparsity. The idea of learning a binary mask vector for delta tuning is also proposed by Zhao et al. [135].

Reparameterization-Based Approach Reparameterization-based approach proposes to reparameterize part of existing parameters in PTMs to a parameter-efficient form by transformation. Let $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\}$ represent the set of parameter subsets to be reparameterized and $\Delta\Theta = \Theta + (\cup_{i=1}^{m} R(\mathbf{p}_i))$ where $R(\mathbf{p}_i)$ is used to reparameterize the parameter subset $\mathbf{p}_i$.

LoRA [43] decomposes the change of the original weight matrices in the multi-head attention modules into low-rank matrices. Its basic idea is inspired by Aghajanyan et al. [2] that the full-parameter fine-tuning phase of PTMs has a low intrinsic dimension. As shown in Fig. 5.19, LoRA utilizes four low-rank matrices to decomposite the changes of the transform matrices for key and value spaces, which can be formulated as:

$$\mathbf{h}_i = \mathrm{ATT}(\mathbf{x}\mathbf{W}_Q^i, \mathbf{x}(\mathbf{W}_K^i + \underline{\mathbf{A}_K\mathbf{B}_K}), \mathbf{x}(\mathbf{W}_V^i + \underline{\mathbf{A}_V\mathbf{B}_V})), \tag{5.17}$$

where $\mathbf{A}_K, \mathbf{A}_V \in \mathbb{R}^{d\times r}$ and $\mathbf{B}_K, \mathbf{B}_V \in \mathbb{R}^{r\times d}$. In the experiment on the GLUE benchmark, LoRA can nearly achieve comparable performance with full-parameter fine-tuning for the PTMs of various scales and architectures.

Understanding Delta Tuning from Ability Space Qin et al. [84] point out that for a particular delta tuning method, the adaptations of PTM for multiple downstream tasks can be reparameterized as optimizations in a unified low-dimension parameter space. Based on this work, Yi et al. [119] further find that the optimization of different delta tuning methods for adapting PTMs to downstream tasks can also be reparameterized into optimizations in a unified low-dimension parameter space.

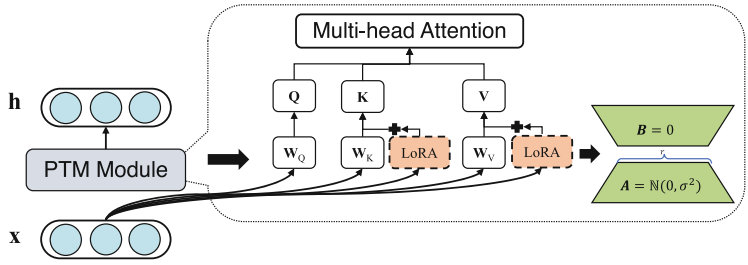


Fig. 5.19 The illustration of the LoRA method

This demonstrates the optimization space of PTMs' adaptation is intrinsically low-dimensional, which may explain why the adaptation of PTMs can be done with relatively small-scale downstream data. The intrinsic low-dimensional tuning parameter space may indicate parts of the parameters in the PTMs that are related to each other, which may be co-activated and controlled in a unified manner. This phenomenon is also observed by MoEfication [134].

5.3.3 *Prompt Learning*

Prompt learning [64] is proposed to overcome the limitation of full-parameter fine-tuning from the data utilization perspective. It reformulates the downstream tasks as the conditional language modeling form with a textual prompt as task instruction. This could effectively bridge the gap between model pre-training and adaptation for PTMs. Moreover, it incorporates the prior knowledge of domain experts into the model adaptation phase by elaborately designing the textual prompt, which can be viewed as feature engineering toward PTMs. Therefore, prompt learning can significantly reduce the requirements of extensive training data in the model adaptation phase while maintaining good performance.

Prompt learning is inspired by the in-context learning ability in GPT-3 [7]. In-context learning regards PTMs as a black box and utilizes the input to describe the downstream task with a task instruction and some optional examples to PTMs. It hopes PTMs to learn to proceed with the downstream task from the given descriptive context without updating the model parameters. Taking the English-to-Chinese translation task as an example, as shown in Fig. 5.20, in-context learning can be divided into two levels: (1) task instruction learning, which adds a task instruction (*Translate English to Chinese*) in front of the translated text sequence and requires the PTMs to perform zero-shot learning, and (2) example learning, which also adds some task examples besides the task instruction and requires the PTMs to perform few-shot learning based on the task-related context.

In-context learning provides a flexible way to utilize PTMs, with which we can describe many possible tasks, from text classification, named entity recognition,

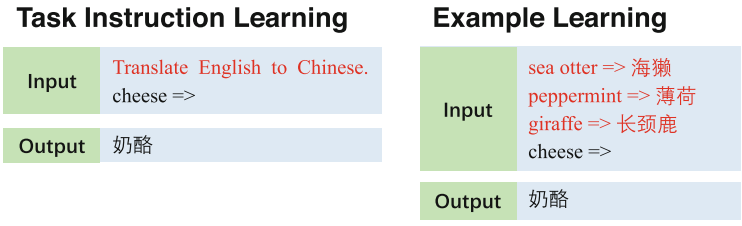


Fig. 5.20 The illustration of task instruction learning and example learning of in-context learning. The figure is redrawn according to Fig. 2.1 from OpenAI's GPT-3 paper [7]

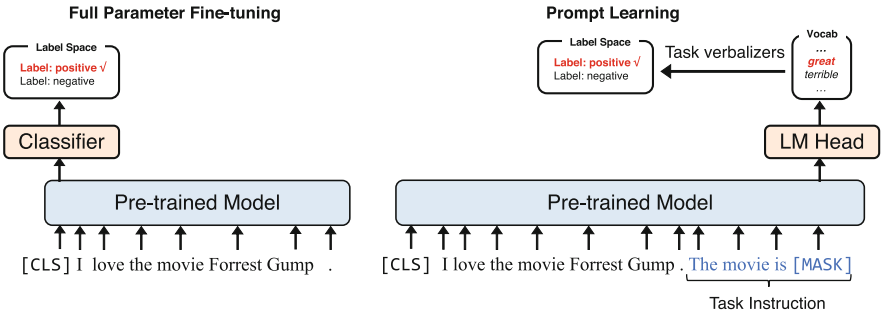


Fig. 5.21 An example of prompt learning

and question answering to machine translation. The experimental results in GPT-3 show that large PTM with in-context learning can even achieve better performance compared with the full-parameter fine-tuning in small PTMs.

From the fantastic results of in-context learning in GPT-3, researchers realize that we can stimulate PTMs' specific functionalities with textual prompts. After that, many researchers focus on exploring how to better stimulate PTMs with textual prompts, i.e., prompt learning. As shown in Fig. 5.21, prompt learning has two essential parts, including task instruction, which is a textual prompt (*The movie is* `[MASK]`) to stimulate the specific functionalities of PTMs for the downstream tasks, and task verbalizers, which maps the output words of the language modeling head to label space of the target task. Therefore, the research work on prompt learning focuses on how to design the optimal task instruction prompts and task verbalizers for downstream tasks.

Task Instruction Design Task instruction design aims to find the optimal task instruction prompts that can achieve the best performance in the downstream tasks. It can be divided into three categories, including manual, automatic, and knowledgeable methods.

Manual Design Early works [7, 19, 77, 120] usually design the task instruction prompts manually, based on the intuition of human experts. Although manual design methods are simple and effective, they still have two significant limitations: first, they require much time and expert experience. Second, the optimal task instruction prompts are highly related to specific PTMs and task datasets, and even experts may fail to find the optimal task instruction prompts.

Automatic Design Later, automatic design methods are proposed to learn or find the optimal task instruction prompts automatically. We categorize them into two typical types: (1) generate-then-rank. It first generates a candidate set of task instruction prompts by prompt mining [47], prompt paraphrasing [40, 122], or prompt generation [5, 28] and then ranks the best one according to the performance in the downstream tasks. (2) Gradient-based search. It searches over all words in the vocabulary to find short task instructions that can stimulate the specific

PTMs to generate the target output of the downstream tasks according to the gradients [95, 106].

Knowledgeable Design Knowledgeable design methods further incorporate external knowledge into the task instruction prompts. For example, Han et al. [38] propose prompt tuning with rules (PTR) to handle text classification tasks. It applies logic rules to guide the construction of task instruction prompts, encoding the prior knowledge of each class into prompt learning. Besides, Chen et al. [9] propose to insert the type markers in front of the head and tail entities to incorporate the entity type knowledge and insert a soft word with the average embeddings of the relation descriptions between the head and tail entities to incorporate the relation knowledge.

Task Verbalizer Design Task verbalizer design aims to find the optimal label word space of the verbalizer, i.e., the optimal words in the output vocabulary to map to the label words. Similar to task instruction design, it can also be divided into manual, automatic, and knowledgeable methods.

Manual Design Early manual task instruction designs usually accompany manual task verbalizer designs [19, 77, 120]. They ask the experienced experts to select the optimal words in the vocabulary as the task verbalizer, which is often based on specific downstream tasks such as sentiment classification, named entity recognition, etc. For example, as shown in Fig. 5.21, for sentiment analysis, it usually maps the probability of the word *great* into the probability of the *positive* sentiment and the probability of the word `terrible` to the `negative` sentiment.

Automatic Design After early manual designs, researchers have devoted much effort to automating the task verbalizer design. Its most typical form is to find a candidate word set by paraphrasing [47], searching [92], or generation [28, 121] that maps to task labels. After that, different from task instruction design, task verbalizer design usually selects the top-k candidate words/phrases as the verbalizer and sums up their probabilities as the label probabilities. The reason is that a task label may have multiple expressions in language. For example, we can describe that *The movie is great/interesting/fantastic/awesome*, and they are all mapped to `positive` sentiment.

Knowledgeable Approaches Knowledgeable task verbalizer design aims to utilize external knowledge information to help design or learn the label word space in the verbalizer. Hu et al. [44] first propose to utilize external knowledge bases to help to expand the verbalizer's label word space. Specially, for topic classification, they utilize the external topic-related vocabulary, and for sentiment classification, they use an external sentiment vocabulary to help expand the candidate words mapping to the label space of the verbalizer. Moreover, Cui et al. [18] extend the label word space from discrete words into soft embeddings and learn prototype vectors as verbalizers by self-supervised contrastive learning. Ding et al. [21] also learn to prototype vectors for entity typing tasks by self-supervised learning.

Connections Between Prompt Learning and Prompt Tuning Prompt learning directly utilizes textual prompts to stimulate the specific functionalities of PTMs

for downstream tasks. However, the optimal textual prompt corresponds to many factors, such as the selection of PTMs, task data distribution, etc. The restricted discrete space of words limits the manual [7, 19, 77, 120], automatic [5, 28, 47, 95, 106], or even knowledgeable prompt learning [9, 38] to find optimal textual prompts. Stimulating PTMs' abilities with textual prompts still has a performance gap with full-parameter fine-tuning in many scenarios. Hence, prompt learning [18, 21] proposes to extend the space of textual prompts to a soft form, i.e., utilizing several additional tunable tokens instead of hard prompt tokens. This can be viewed as a kind of prompt tuning introduced in Sect. 5.3.2. In summary, while prompt tuning is a more parameter-efficient way compared to prompt learning, prompt learning utilizes the prior knowledge of human beings by designing explainable textual prompts and is a natural interface of PTMs which is more explainable for users.

5.4 Advanced Topics

In the previous section, we have introduced the basics of PTMs, including the pre-training and adaptation of text representations. In this section, we present several advanced topics of PTMs, including better model architecture, multilingual representation, multi-task representation, efficient representation, and chain-of-thought reasoning.

5.4.1 Better Model Architecture

Although Transformer-based PTMs have achieved promising results in a wide range of downstream tasks, we still have a question: is Transformer the optimal architecture for PTMs? In this subsection, we introduce the explorations in better model architecture, which can be categorized into three types:

Improving Model Capacity Recently, researchers have found that the strong ability of PTMs comes from their large-scale parameters, i.e., the bigger model leads to better performance. Therefore, researchers explore improving the Transformer architecture to increase the number of model parameters while keeping the same theoretical computation complexity.

Sparsity, which indicates that the model only activates a part of the parameters for a specific task, has been widely explored. In this way, model capacity can be significantly increased without proportionally increasing theoretical computation complexity. Sparsely gated mixture of experts layer (MoE) [94] is thus proposed to allow models to only activate a part of the parameters for each input sample. As shown in Fig. 5.22, the architecture of sparse-gated MoE consists of two parts: experts and a routing network. Each expert is usually a feed-forward neural network. The routing network is to determine which experts are activated when processing

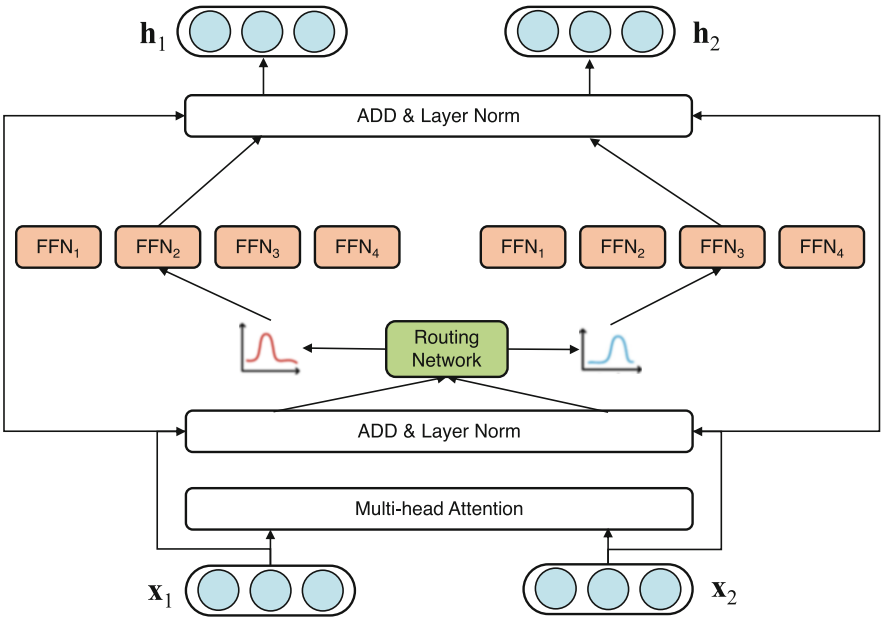


Fig. 5.22 The architecture of sparsely gated mixture of experts layer. The figure is redrawn according to Fig. 2 from the Switch Transformer paper [26].

each input sample. Compared with the vanilla Transformer, sparse-gated MoE only selects a part of experts for computation (the number of parameters is the same as the vanilla feed-forward layer), and thus does not increase the training and inference time.

However, the sparsely gated MoE still cannot be applied in real-world scenarios due to the training instability and the communication costs in GPU clusters. To address these issues, GShard [57], the first work to combine sparsely gated MoE with Transformer architecture, simplifies the routing strategy of sparsely gated MoE, which only assigns at most two experts for each instance and employs a capacity factor to balance the workload of each expert. Based on the improvement of GShard, Switch Transformer [26] extends sparsely gated MoE as the basic modeling block for PTMs, and only allows one expert for each input sample. GLaM [24] further improves the routing strategy by allowing PTMs to select two experts for each input sample, which provides more model capacity while restricting computation cost. The experimental results in Switch Transformer [26] and GLaM [24] show that PTMs with sparsely gated MoE can converge faster than that with vanilla Transformer architecture due to the significantly larger model capacity.

We believe sparse model architectures, which allow PTMs to stimulate a part of the neurons for each input sample, would be an essential feature of the next generation of PTMs' architecture. This corresponds to the phenomenon in

neuroscience that each neuron tends to have fewer average connections to other neurons with the increasing number of neurons in a primate brain. In fact, Zhang et al. [134] also point out that the vanilla Transformer can be transformed into a sparse-gated MoE form by their proposed MoEfication strategy, i.e., the vanilla Transformer is a special case of sparse-gated MoE. It may demonstrate that sparsity is the intrinsic emergent characteristic of the neural network after pre-training, even without any constraint or pre-design.

Modeling Long-Term Dependency Besides the model capacity, another critical problem of the vanilla Transformer is that its self-attention mechanism's computational and memory footprints are quadratic with the length of the input sequence. Hence, a question is can we implement a quadratic Transformer so that the scale of computational and memory requirements are linear with the input sequence length? To this end, a natural solution is to approximate the original multi-head attention with faster attention mechanisms. We introduce several typical fast attention mechanisms widely used in PTMs.

Structured Sparse Attention Clark et al. [14] point out that the attention heads of PTMs exhibit specific patterns. For example, some tokens may attend to the `[CLS]` token, and some tokens may attend to the other tokens' specific positional offsets, etc. Motivated by this phenomenon, later works propose to replace the original full-connected multi-head attentions with several types of pre-defined structured sparse attentions, including (1) the sparse global attention with which the token is visible for all other tokens and typically employed in the `[CLS]` token, (2) the structured local attention which reduces the visible field for most other tokens with stride window form [4, 124] or blockwise form [12, 85], etc.

Low-Rank Approximation Since the attention heads of PTMs exhibit specific patterns, the learned attention matrices are low-rank. Hence, several recent works [13, 51, 108] propose to approximate the multi-head attention matrices with low-rank decomposition, reducing the multi-head attention to an operation which is linear with the length of the sentence.

Cluster-Based Sparse Attention Its basic idea is that tokens can only attend to similar tokens according to the routing mechanism of the multi-head attention layer. Hence, it learns to cluster tokens in the input text sequence according to their similarities and restricts that only the tokens in the same clusters are visible to each other in the attention layer. For example, Reformer [53] employs a locality-sensitive hashing strategy to cluster tokens for attention calculation, and routing Transformer [89] employs a ks-means algorithm to cluster tokens.

Retrieving External Information Researchers argue that it is a very unreasonable way for traditional PTMs to store all knowledge in model parameters due to their limited capacity compared with the endless knowledge. Therefore, REALM [36] proposes to teach PLMs to retrieve and use external knowledge during inference. REALM augments the BERT model with a latent knowledge retriever, allowing PTMs to retrieve relevant text information (i.e., documents) from a large-scale

unlabeled corpus, such as Wikipedia. In the experiment, REALM demonstrates that it can achieve much better results compared to T5–11B, which has nearly 100 times parameters, verifying the effectiveness of retrieving external knowledge. Nevertheless, REALM is based on the BERT model, an encoder-based PTM, which is limited in classification tasks. To address this issue, RAG [60] further extends the idea of retrieval augmentation into the encoder-decoder-based PTM, allowing retrieval-based PTMs to handle text generation tasks.

5.4.2 Multilingual Representation

Big PTMs trained on the large-scale monolingual corpus, such as the English corpus, have shown superior performance in a wide range of NLP tasks. Nevertheless, there are thousands of languages in the world, and it is nearly impossible and unreasonable for us to train individual big PTMs for each language. The reason lies in two points: (1) there are many resource-scarce languages that we cannot easily collect a large amount of unlabeled text for pre-training; (2) there are many NLP tasks related to more than one language. In fact, semantics is independent of symbolic languages since people in the world can express the same meaning in different languages. Hence, training multilingual PTMs has recently attracted much attention from researchers. In this subsection, we introduce the explorations of learning multilingual PTMs in two main categories:

Nonparallel Pre-training Nonparallel pre-training is the initial attempt at learning multilingual PTMs, which directly pre-trains PTMs on nonparallel multilingual corpora with monolingual pre-training tasks. Its basic idea is that the lexical overlaps between languages can help to align the multilingual language representations of PTMs learned from corpora of multiple languages in the semantic space. It can be divided into three categories according to the model architecture: (1) encoder-based PTMs. Multilingual BERT (mBERT) [20] is the first multilingual PTM with nonparallel pre-training. It pre-trains with an MLM pre-training objective on multilingual Wikipedia corpora which have 104 languages but are nonparallel. (2) Decoder-based PTMs. Multilingual GPT (mGPT) [96] pre-trains with a CLM pre-training objective with Wikipedia and colossal clean crawled corpus, learning a multilingual PTM with 60 languages from 25 language families. (3) Encoder-decoder-based PTMs. mBART [67] and mT5 [115] extend the DLM pre-training objective to support multilingual pre-training. They simply add special language symbols to the end of the input text for the encoder and the start of the input text for the decoder of PTMs. Such special language symbols enable PTMs to realize the languages to be encoded and generated. With the development of multilingual PTMs with nonparallel pre-training, we still wonder how multilingual these PTMs can reach. Therefore, Pires et al. [79] take mBERT as an example for investigation and find that mBERT can achieve superior zero-shot performance in a wide range of cross-lingual NLP tasks, showing its ability in cross-lingual

knowledge generalization. This verifies the reasonability of learning multilingual capabilities from the nonparallel multilingual corpora with the Transformer-based PTMs.

A major challenge of multilingual pre-training is how to alleviate the data unbalance problem between high-resource and low-resource languages. To address this issue, mBERT perform exponentially smoothed weighting of the data distribution of different languages during pre-training data construction. Furthermore, XLM-R [16] constructs a new nonparallel multilingual corpus named CC-100, which has 100 languages. Compared to the Wikipedia corpora used by mBERT, CC-100 has a larger scale, especially for those low-resource languages.

Although the monolingual pre-training objective can simply extend to train multilingual PTMs in nonparallel corpora, it cannot well utilize the language-alignment signals from parallel corpora. In fact, such language-alignment signals are essential for multilingual NLP tasks such as cross-lingual information retrieval and machine translation.

Parallel Pre-training Parallel pre-training is another typical approach for learning multilingual PTMs, which mainly focuses on designing multilingual pre-training tasks to better utilize the language-alignment signals from parallel corpora. This line of research work can be divided into three types according to the pre-training tasks: (1) cross-lingual masked language modeling. XLM [17] thus proposes the cross-lingual masked language modeling (CMLM) pre-training objective to better utilize the language-alignment signals from bilingual sentence pairs. Extending the MLM objective, CMLM concatenates two semantically matched sentences in two languages and asks PTMs to recover randomly masked tokens in the connected sentence. Compared to MLM, CMLM allows PTMs to recover the masked tokens not only from the monolingual context information but also from its aligned tokens in another language. (2) Cross-lingual denoising language modeling. XNLG [10] proposes cross-lingual denoising language modeling (CDLM). Different from DLM, CDLM assigns the inputs of the encoder and decoder of PTMs with text in different languages, similar to CMLM. (3) Cross-lingual contrastive learning. InfoXLM [11] further analyzes MLM and CMLM from the perspective of information theory and proposes a contrastive pre-training objective for learning multilingual PTMs based on the analysis. Based on InfoXLM, HICTL [113] further extends the idea of cross-lingual contrastive learning to help PTMs to learn with multilingual representations at the word level and sentence level.

Compared to nonparallel pre-training, performing parallel pre-training can learn semantic-aligned multilingual representations more effectively and thus achieves promising results in a series of cross-lingual NLP tasks. However, most existing parallel pre-training objective relies on a large number of parallel data at the sentence level and even word level, which is quite rare for many languages. To address this issue, ERNIE-M [74] proposes to expand the scale of parallel multilingual corpora using the back-translation technique as well as a back-translation masked language modeling (BTMLM) pre-training objective. Except for utilizing machine translation technique, ALM [116] proposes a code-switched pre-training objective,

which directly replaces the tokens/spans in one language with the token/spans from either its semantic-aligned sentence in another language or bilingual lexicons and then performs CMLM on it.

5.4.3 Multi-Task Representation

Multi-task learning [8] has been widely explored in the representation learning of NLP. With the development of PTMs, pre-trained representations have become much more expressive, unifying text representations across a wide range of NLP tasks. Nevertheless, it still has no clear answer whether multi-task learning in downstream tasks can make the pre-trained representations more expressive. Therefore, researchers have devoted many efforts to exploring how multi-task learning of downstream tasks can promote the PTMs and stimulate the potential of pre-trained representations. We roughly divide the explorations into the following three directions:

Multi-Task Pre-training Multi-task pre-training unifies the learning paradigm of various kinds of NLP tasks during the pre-training stage for PTMs. The basic idea of multi-task pre-training is to introduce the learning signals of different NLP tasks into the pre-training phase. For example, T5 [88] unifies nearly all NLP tasks as text-to-text generation problems so that it can pre-train the encoder-decoder-based PTMs with all NLP task data besides self-supervised learning with unlabeled corpus. After that, Liu et al. [64] propose to unify the learning objective of all NLP tasks as prompt learning by inserting human-designed /automatically generated task prompts into the input text. This combines the idea of multi-task learning and prompt learning, which can further mitigate the gap between multi-task pre-training and task-specific model adaptation. Besides directly enhancing the PTMs by multi-task learning, some works also explore understanding the principle of task unification in the pre-training stage. Qin et al. [84] reveal that PTMs actually learn the capabilities to handle multiple tasks in the pre-training phase. Moreover, they find that there exists a unified low-dimensional task subspace to the task capabilities, and the task-specific model adaptation of big PTMs can be all reparameterized into optimizing the task vector in this space.

Multi-Task Preadaptation Multi-task preadaptation additionally adapts the big PTMs by adding intermediate auxiliary tasks between pre-training and model adaptation. The research of multi-task preadaptation can be roughly divided into three categories: (1) exploring the effectiveness of preadaptation. First, big PTMs could further learn more task capabilities that are not reflected in the self-supervised learning signals by incorporating the intermediate knowledge transfer from auxiliary tasks, such as text classification [78], named entity recognition [128], relation extraction [82], and question answering [30]. Second, preadaptation on domain-specific unlabeled data for downstream tasks could provide rich domain-specific knowledge for PTMs [33, 35, 56, 83]. (2) Understanding the working mechanism of

preadaptation. Although simple and effective, the success of preadaptation is very sensitive to the selection of intermediate auxiliary tasks. Hence, recent works have focused on exploring the reason for this phenomenon. One on hand, Aghajanyan et al. [1] find that scaling the number of tasks as well as adopting task-heterogeneous batches and task-rebalancing loss scaling is important for multi-task preadaptation. On the other hand, Pruksachatkun et al. [81] explore the task capabilities big PTMs learn during the model preadaptation stage and find that the preadaptation tasks requiring high-level reasoning abilities lead to better downstream task performance. (3) Selecting intermediate auxiliary tasks for preadaptation. Researchers also explore how to efficiently select the optimal intermediate auxiliary tasks according to the knowledge transferability among different tasks, such as embedding-based methods [80], manually defined feature-based methods [63], task gradient-based methods [27], etc.

Multi-Task Model Adaptation Multi-task model adaptation aims to fine-tune PTMs so that their generated text representations can jointly solve multiple tasks. Researchers argue that PTMs have learned versatile knowledge during self-supervised pre-training in the large-scale unlabeled corpus, which may help a wide range of NLP tasks. Hence, big PTMs can be stimulated with multiple task signals to build a unified downstream task model that can handle a variety of downstream NLP tasks. However, in real-world applications, we usually suffer from the data imbalance problem, i.e., the data volume of different tasks varies a lot. Hence, simply performing typical multi-task learning for model adaptation will lead to underfitting in resource-rich tasks and over-fitting on resource-scarce tasks [3]. To address this problem, the basic idea is to learn task-specific model modules for the PTMs, which can be divided into three types: (1) task-specific layers which are added on top of the shared universal text representations of PTMs [65]; (2) task-specific controlling modules which generate weights of the existing layers such as the FFN layers of the Transformer [102]; and (3) delta tuning modules which reduce the number of newly added model parameters for multiple downstream tasks [70, 98].

5.4.4 Efficient Representation

Although the text representations learned by big PTMs have shown fantastic abilities in language understanding, it requires a large amount of inference time, making them impractical in real-world applications. Therefore, many recent works have explored how to generate efficient pre-trained representations, which can be mainly divided into three types, including model pruning, knowledge distillation, and parameter quantization.

Model Pruning Model pruning reduces the size of PTMs by omitting redundant model parameters of big PTMs. Model pruning has two main categories: (1) unstructured pruning, which directly prunes the model parameter at the neuron

level. CompressingBERT [31] conducts a comprehensive analysis of the multi-head attention layers and feed-forward layers of Transformer blocks in PTMs and then prunes 30–40% of the weights in PTMs without loss of performance during the pre-training stage. This is because these weights do not encode any useful inductive bias for language understanding in the downstream tasks. Although unstructured pruning effectively makes PTMs more sparse, it cannot speed up the inference since the computation hardware cannot well deal with the pruned unstructured PTMs. (2) Structured pruning, which prunes the model parameter at the attention level or layer level. For layer-level pruning, Fan et al. [25] propose to randomly drop several layers so that they can dynamically pick up parts of the model layers during inference. Besides, DeeBERT [114] and CascadeBERT [61] learn to exit the inference in the shallow layer of PTMs in the downstream tasks. While these works all focus on the layer-wise early exiting for the classification tasks, TR-BERT [118] further extends the idea of early exiting into the inference of the sequence labeling tasks for PTMs. For attention-level pruning, researchers observe that there exist redundancy phenomena in attention heads, i.e., the same syntactic or semantic relations may be modeled by more than one attention head [71, 105], and thus they propose to remove the redundant attention heads. Compared to unstructured pruning, the PTMs after structured pruning is still structured and can be easily accelerated in typical computation hardware such as GPUs.

Knowledge Distillation Knowledge distillation learns a smaller student PTM to transfer the knowledge from a bigger teacher PTM, which aims to reduce both the inference time and memory cost while maintaining the performance of big PTMs. The main challenge of knowledge distillation is how to construct effective supervisions from the teacher PTMs, which can be divided into three types: from (1) the original output probabilities [91] of the self-supervised learning tasks or downstream tasks, (2) the hidden states in different layers [48, 99], and (3) the attention matrices [109]. Compared to directly training a smaller PTM, knowledge distillation can transfer the learned knowledge in larger PTMs, enhancing the representations generated by student PTMs.

Parameter Quantization Parameter quantization converts the precision of model parameters from a higher float point to the lower one. The original precision of PTMs is usually 32 bits, 16 bits, or mixed 32–16-bits. Q8BERT [123] first proposes to quantize the model parameters' coding of PTMs into 8-bit to speed up its inference speed. However, it is harder to reduce the parameter coding into extremely low-bit further (e.g., 1 or 2 bits) since the low fixed points have huge precision gaps with float points which may affect the output representations of PTMs. To address this issue, Q-BERT [110] further proposes to apply different levels of precisions for different kinds of modules in the PTMs according to their different precision requirements. Besides, TernaryBERT [130] proposes quantization-aware training for PTMs, which directly trains the quantized PTMs during the pre-training stage. However, extreme low-bit quantization is still limited in real-world applications since it relies on specially designed hardware implementation.

5.4.5 *Chain-of-Thought Reasoning*

Recent studies have revealed that even the extremely large-scale PTMs can still struggle with complex multi-step reasoning tasks, such as numerical reasoning and commonsense reasoning. Therefore, we have a question: do PTMs learn complex reasoning abilities in the pre-training stage? If yes, how can we stimulate the complex reasoning ability of PTMs?

To this end, chain-of-thought (COT) reasoning [112] is proposed to stimulate the complex reasoning ability of PTMs. The basic idea of COT reasoning is that a model-generated chain of thought can enable PTMs to mimic an intuitive thought process to perform reasoning. As shown in Fig. 5.23, COT reasoning adds a human-labeled explanation that describes the explicit intermediate reasoning path as the textual prompt for obtaining the final answer. COT reasoning hopes the PTMs can learn to decompose the complex reasoning task into multiple intermediate steps that are solved individually, and then PTMs can obtain the correct answer by reasoning over the generated path. In this way, PTMs can generate more interpretable solutions and improve the model performance in the samples requiring complex reasoning. Experimental results in the original paper [112] show that the complex reasoning ability emerges from PTMs when the model parameter grows up to about 100B with COT reasoning, and such big PTMs can achieve promising results on numerical reasoning and commonsense reasoning tasks. Later, Wang et al. [111] further propose an answer ensembling strategy to improve the reasoning accuracy for COT reasoning. They first sample a diverse set of reasoning paths with beam search and then perform reasoning over them. After that, they select the most consistent final answer from the generated answer set following these reasoning paths. Their experiments show that such a simple strategy can effectively improve the model performance without additional training for various PTMs with

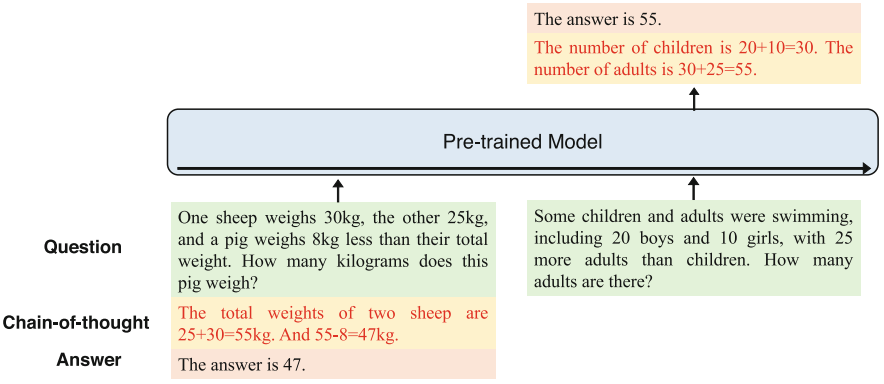


Fig. 5.23 An example of chain-of-thought reasoning. The text colored red is the added explanation. The figure is redrawn according to Fig. 1 from the chain-of-thought reasoning paper [112]

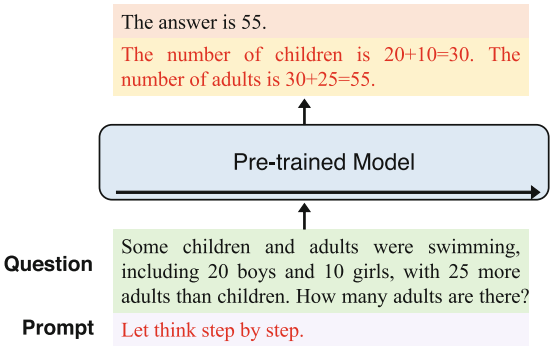


Fig. 5.24 An example of zero-shot chain-of-thought reasoning

different scales. Although simple and effective, a major drawback of COT reasoning is that it requires expensive manually annotating explanations for different tasks and datasets. To address this problem, STaR [126] further proposes a bootstrapping approach to generate high-quality explanations for each example from a tiny seed training set and verifies its effectiveness in arithmetic, math word problems, and commonsense reasoning, especially in the few-shot settings.

Now, the remaining question is where does the complex reasoning ability of PTMs come from? One possibility is the intrinsic ability of PTMs that are learned in the self-supervised pre-training phase, while another possibility is learning from the provided reasoning explanations due to the few-shot learning ability of big PTMs. Recently, Kojima et al. [54] reveal that big PTMs can perform complex multi-step reasoning without any human-labeled explanation prompt. As shown in Fig. 5.24, they find that big PTMs can perform zero-shot COT reasoning by simply adding "Let's think step by step" before each answer as the textual prompt. They show that with such a simple prompt, the zero-shot performance of big PTMs achieves consistent improvements in diverse NLP tasks, including arithmetic, symbolic, and other reasoning tasks. This preliminarily demonstrates that the complex reasoning ability may be learned by PTMs during pre-training on the large-scale corpus.

Nevertheless, it is still unclear to us how PTMs learn such ability: do such reasoning text patterns exist in the training corpus and PTMs just learn a shortcut? Or have PTMs evolved into more intelligent agents we never imagined, i.e., the pre-trained representations of big PTMs may also exist other untapped and understudied fundamental "magic" abilities? This is still an open question. But we confirmedly believe that big PTMs are the foundation and future direction toward high-level cognitive intelligence.

For this question, an interesting recent finding is that big PTMs have the ability to behavior learning. For example, WebGPT [72] can learn how to operate an online search engine like Bing API[1] to answer open-domain questions. InstructGPT [73] can perform various types of tasks according to the corresponding task instructions

[1] https://www.microsoft.com/en-us/bing/apis/bing-web-search-api.

by learning from human feedback with reinforcement learning. Inspired by the idea of reinforcement learning from human feedback of InstructGPT, more recently, ChatGPT[2] have also demonstrated the fantastic dialogue ability of big PTMs, which learns from tens of thousands of human conversation behaviors. All these works make an initial exploration to more intelligent utilization of big PTMs: by learning from human behavior with reinforcement learning, we may mine the unexplored high-level cognitive intelligence hiding in big PTMs.

5.5 Summary and Further Readings

In this section, we review the current progress and the remaining challenges of pre-trained models for representation learning in NLP. First, we introduce the pre-training tasks, including word-level pre-training and sentence-level pre-training. After that, we turn to the model adaptation, from full-parameter fine-tuning to optimization-perspective delta tuning and data-perspective prompt learning. Finally, we discuss several advanced topics, such as better model architecture, multilingual learning, multi-task learning, efficient representations, and chain-of-thought reasoning.

For further understanding of pre-trained models for representation learning, you can find more related papers in our paper lists about pre-trained models,[3] delta tuning[4] and prompt learning.[5] On the survey of pre-trained models, Han et al. [37] give a comprehensive review of the history and recent breakthroughs of PTMs and also discuss its remaining open challenges. Ding et al. [22] give a detailed review of existing delta tuning methods. Bommasani et al. [6] systematically review the PTMs' developments from the capability, technical principle, application, and societal impact perspectives.

Acknowledgments Zhiyuan Liu, Yankai Lin, and Maosong Sun designed the overall architecture of this chapter; Yankai Lin and Ning Ding drafted the chapter. Zhiyuan Liu proofread and revised this chapter.

We also thank Xu Han, Zheni Zeng, Shengding Hu, Zhengyan Zhang, Yingfa Chen, and Zhiyuan Zeng for proofreading the chapter.

This is the newly complemented chapter about pre-trained models of the second edition of the book *Representation Learning for Natural Language Processing*. The first edition of the book was published in 2020 [69].

[2] https://openai.com/blog/chatgpt/.

[3] https://github.com/thunlp/PLMpapers.

[4] https://github.com/thunlp/DeltaPapers.

[5] https://github.com/thunlp/PromptPapers.

References

1. Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. Muppet: Massive multi-task representations with pre-finetuning. In *Proceedings of EMNLP*, 2021.
2. Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of ACL-IJCNLP*, 2021.
3. Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George Foster, Colin Cherry, et al. Massively multilingual neural machine translation in the wild: Findings and challenges. *arXiv preprint arXiv:1907.05019*, 2019.
4. Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
5. Eyal Ben-David, Nadav Oved, and Roi Reichart. Pada: A prompt-based autoregressive approach for adaptation to unseen domains. *arXiv preprint arXiv:2102.12206*, 2021.
6. Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.
7. Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Proceedings of NeurIPS*, 2020.
8. Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.
9. Xiang Chen, Ningyu Zhang, Xin Xie, Shumin Deng, Yunzhi Yao, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. Knowprompt: Knowledge-aware prompt-tuning with synergistic optimization for relation extraction. In *Proceedings of WebConf*, 2022.
10. Zewen Chi, Li Dong, Furu Wei, Wenhui Wang, Xian-Ling Mao, and Heyan Huang. Cross-lingual natural language generation via pre-training. In *Proceedings of AAAI*, 2020.
11. Zewen Chi, Li Dong, Furu Wei, Nan Yang, Saksham Singhal, Wenhui Wang, Xia Song, Xian-Ling Mao, He-Yan Huang, and Ming Zhou. Infoxlm: An information-theoretic framework for cross-lingual language model pre-training. In *Proceedings of NAACL-HLT*, 2021.
12. Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.
13. Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, David Belanger, Lucy Colwell, et al. Masked language modeling for proteins via linearly scalable long-context transformers. *arXiv preprint arXiv:2006.03555*, 2020.
14. Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does BERT look at? an analysis of bert's attention. In *Proceedings of ACL Workshop BlackboxNLP*, 2019.
15. Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *Proceedings of ICLR*, 2019.
16. Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Édouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of ACL*, pages 8440–8451, 2020.
17. Alexis Conneau and Guillaume Lample. Cross-lingual language model pretraining. In *Proceedings of NeurIPS*, 2019.
18. Ganqu Cui, Shengding Hu, Ning Ding, Longtao Huang, and Zhiyuan Liu. Prototypical verbalizer for prompt-based few-shot tuning. In *Proceedings of ACL*, 2022.
19. Leyang Cui, Yu Wu, Jian Liu, Sen Yang, and Yue Zhang. Template-based named entity recognition using bart. In *Findings of ACL*, 2021.
20. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.

21. Ning Ding, Yulin Chen, Xu Han, Guangwei Xu, Pengjun Xie, Hai-Tao Zheng, Zhiyuan Liu, Juanzi Li, and Hong-Gee Kim. Prompt-learning for fine-grained entity typing. *arXiv preprint arXiv:2108.10604*, 2021.
22. Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*, 2022.
23. Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. Unified language model pre-training for natural language understanding and generation. In *Proceedings of NeurIPS*, 2019.
24. Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *Proceedings of ICML*, 2022.
25. Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *Proceedings of ICLR*, 2019.
26. William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.
27. Chris Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. Efficiently identifying task groupings for multi-task learning. In *Proceedings of NeurIPS*, 2021.
28. Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. In *Proceedings of ACL*, 2021.
29. Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. In *Proceedings of EMNLP*, 2021.
30. Michael Glass, Alfio Gliozzo, Rishav Chakravarti, Anthony Ferritto, Lin Pan, GP Shrivatsa Bhargav, Dinesh Garg, and Avirup Sil. Span selection pre-training for question answering. In *Proceedings of ACL*, 2020.
31. Mitchell Gordon, Kevin Duh, and Nicholas Andrews. Compressing bert: Studying the effects of weight pruning on transfer learning. In *Proceedings of RL4NLP*, 2020.
32. Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. Ppt: Pre-trained prompt tuning for few-shot learning. In *Proceedings of ACL*, 2022.
33. Yuxian Gu, Zhengyan Zhang, Xiaozhi Wang, Zhiyuan Liu, and Maosong Sun. Train no evil: Selective masking for task-guided pre-training. In *Proceedings of EMNLP*, 2020.
34. Demi Guo, Alexander M Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. In *Proceedings of ACL-IJCNLP*, 2021.
35. Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of ACL*, 2020.
36. Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *Proceedings of ICML*, 2020.
37. Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2021.
38. Xu Han, Weilin Zhao, Ning Ding, Zhiyuan Liu, and Maosong Sun. PTR: Prompt tuning with rules for text classification. *arXiv preprint arXiv:2105.11259*, 2021.
39. Zellig S Harris. Distributional structure. *Word*, 10(2–3):146–162, 1954.
40. Adi Haviv, Jonathan Berant, and Amir Globerson. Bertese: Learning to speak to bert. In *Proceedings of EACL*, 2021.
41. Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.
42. Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *Proceedings of ICML*, 2019.

43. Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. LoRA: Low-rank adaptation of large language models. In *Proceedings of ICLR*, 2021.
44. Shengding Hu, Ning Ding, Huadong Wang, Zhiyuan Liu, Jingang Wang, Juanzi Li, Wei Wu, and Maosong Sun. Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification. In *Proceedings of ACL*, 2022.
45. Goodfellow Ian, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.
46. Sverker Janson, Evangelina Gogoulou, Erik Ylipää, Amaru Cuba Gyllensten, and Magnus Sahlgren. Semantic re-tuning with contrastive tension. In *Proceedings of ICLR*, 2021.
47. Zhengbao Jiang, Frank F Xu, Jun Araki, and Graham Neubig. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438, 2020.
48. Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. In *Findings of EMNLP*, 2020.
49. Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020.
50. Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. In *Proceedings of NeurIPS*, 2021.
51. Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of ICML*, 2020.
52. Taeuk Kim, Kang Min Yoo, and Sang-goo Lee. Self-guided contrastive learning for bert sentence representations. In *Proceedings of ACL-IJCNLP*, 2021.
53. Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *Proceedings of ICLR*, 2019.
54. Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.
55. Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *Proceedings of ICLR*, 2019.
56. Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.
57. Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *Proceedings of ICLR*, 2020.
58. Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of EMNLP*, 2021.
59. Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of ACL*, 2020.
60. Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of NeurIPS*, 2020.
61. Lei Li, Yankai Lin, Deli Chen, Shuhuai Ren, Peng Li, Jie Zhou, and Xu Sun. Cascadebert: Accelerating inference of pre-trained language models via calibrated complete models cascade. In *Findings of EMNLP*, 2021.
62. Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of ACL-IJCNLP*, 2021.

63. Yu-Hsiang Lin, Chian-Yu Chen, Jean Lee, Zirui Li, Yuyan Zhang, Mengzhou Xia, Shruti Rijhwani, Junxian He, Zhisong Zhang, Xuezhe Ma, et al. Choosing transfer languages for cross-lingual learning. In *Proceedings of ACL*, 2019.
64. Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*, 2021.
65. Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *Proceedings of ACL*, 2019.
66. Ye Liu, Yao Wan, Lifang He, Hao Peng, and S Yu Philip. Kg-bart: Knowledge graph-augmented bart for generative commonsense reasoning. In *Proceedings of AAAI*, 2021.
67. Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742, 2020.
68. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
69. Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation Learning for Natural Language Processing*. Springer, 2020.
70. Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of ACL-IJCNLP*, 2021.
71. Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *Proceedings of NeurIPS*, 2019.
72. Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.
73. Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.
74. Xuan Ouyang, Shuohuan Wang, Chao Pang, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. Ernie-m: Enhanced multilingual representation by aligning cross-lingual semantics with monolingual corpora. In *Proceedings of EMNLP*, 2021.
75. Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, 2018.
76. Matthew E Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A Smith. Knowledge enhanced contextual word representations. In *Proceedings of EMNLP-IJCNLP*, 2019.
77. Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *Proceedings of EMNLP-IJCNLP*, 2019.
78. Jason Phang, Thibault Févry, and Samuel R Bowman. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*, 2018.
79. Telmo Pires, Eva Schlinger, and Dan Garrette. How multilingual is multilingual bert? In *Proceedings of ACL*, 2019.
80. Clifton Poth, Jonas Pfeiffer, Andreas Rücklé, and Iryna Gurevych. What to pre-train on? efficient intermediate task selection. In *Proceedings of EMNLP*, 2021.
81. Yada Pruksachatkun, Jason Phang, Haokun Liu, Phu Mon Htut, Xiaoyi Zhang, Richard Yuanzhe Pang, Clara Vania, Katharina Kann, and Samuel Bowman. Intermediate-task transfer learning with pretrained language models: When and why does it work? In *Proceedings of ACL*, 2020.

82. Yujia Qin, Yankai Lin, Ryuichi Takanobu, Zhiyuan Liu, Peng Li, Heng Ji, Minlie Huang, Maosong Sun, and Jie Zhou. Erica: Improving entity and relation understanding for pre-trained language models via contrastive learning. In *Proceedings of ACL-IJCNLP*, 2021.
83. Yujia Qin, Yankai Lin, Jing Yi, Jiajie Zhang, Xu Han, Zhengyan Zhang, Yusheng Su, Zhiyuan Liu, Peng Li, Maosong Sun, et al. Knowledge inheritance for pre-trained language models. *arXiv preprint arXiv:2105.13880*, 2021.
84. Yujia Qin, Xiaozhi Wang, Yusheng Su, Yankai Lin, Ning Ding, Zhiyuan Liu, Juanzi Li, Lei Hou, Peng Li, Maosong Sun, et al. Exploring low-dimensional intrinsic task subspace via prompt tuning. *arXiv preprint arXiv:2110.07867*, 2021.
85. Jiezhong Qiu, Hao Ma, Omer Levy, Wen-tau Yih, Sinong Wang, and Jie Tang. Blockwise self-attention for long document understanding. In *Findings of EMNLP*, 2020.
86. Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. 2018.
87. Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.
88. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67, 2020.
89. Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.
90. Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. Adapterdrop: On the efficiency of adapters in transformers. In *Proceedings of EMNLP*, 2021.
91. Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.
92. Timo Schick, Helmut Schmid, and Hinrich Schütze. Automatically identifying words that can serve as labels for few-shot text classification. In *Proceedings of COLING*, 2020.
93. Claude E Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.
94. Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.
95. Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of EMNLP*, 2020.
96. Oleh Shliazhko, Alena Fenogenova, Maria Tikhonova, Vladislav Mikhailov, Anastasia Kozlova, and Tatiana Shavrina. mgpt: Few-shot learners go multilingual. *arXiv preprint arXiv:2204.07580*, 2022.
97. Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. MASS: Masked sequence to sequence pre-training for language generation. In *Proceedings of ICML*, 2019.
98. Asa Cooper Stickland and Iain Murray. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *Proceedings of ICML*, pages 5986–5995. PMLR, 2019.
99. Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for BERT model compression. In *Proceedings of EMNLP-IJCNLP*, 2019.
100. Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. Ernie: Enhanced representation through knowledge integration. *arXiv preprint arXiv:1904.09223*, 2019.
101. Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *arXiv preprint arXiv:2206.06522*, 2022.
102. Yi Tay, Zhe Zhao, Dara Bahri, Donald Metzler, and Da-Cheng Juan. Hypergrid transformers: Towards a single model for multiple tasks. In *International Conference on Learning Representations*, 2020.

103. Sebastian Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, 1998.
104. Ashish Vaswani, Noam Shazeer, Niki Parmar, Llion Jones, Jakob Uszkoreit, Aidan N Gomez, and Lukasz Kaiser. Attention is all you need. In *Proceedings of NeurIPS*, 2017.
105. Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of ACL*, 2019.
106. Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. Universal adversarial triggers for attacking and analyzing nlp. In *Proceedings of EMNLP-IJCNLP*, 2019.
107. Kexin Wang, Nils Reimers, and Iryna Gurevych. Tsdae: Using transformer-based sequential denoising auto-encoderfor unsupervised sentence embedding learning. In *Findings of EMNLP*, 2021.
108. Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.
109. Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Proceedings of NeurIPS*, 2020.
110. Xiaosen Wang, Zeliang Zhang, Kangheng Tong, Dihong Gong, Kun He, Zhifeng Li, and Wei Liu. Triangle attack: A query-efficient decision-based adversarial attack. *arXiv preprint arXiv:2112.06569*, 2021.
111. Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
112. Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.
113. Xiangpeng Wei, Rongxiang Weng, Yue Hu, Luxi Xing, Heng Yu, and Weihua Luo. On learning universal representations across languages. In *Proceedings of ICLR*, 2020.
114. Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating bert inference. In *Proceedings of ACL*, 2020.
115. Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mt5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of NAACL-HLT*, 2021.
116. Jian Yang, Shuming Ma, Dongdong Zhang, Shuangzhi Wu, Zhoujun Li, and Ming Zhou. Alternating language modeling for cross-lingual pre-training. In *Proceedings of AAAI*, 2020.
117. Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. XLNet: Generalized autoregressive pretraining for language understanding. In *Proceedings of NeurIPS*, 2019.
118. Deming Ye, Yankai Lin, Yufei Huang, and Maosong Sun. Tr-bert: Dynamic token reduction for accelerating bert inference. In *Proceedings of NAACL-HLT*, 2021.
119. Jing Yi, Weize Chen, Yujia Qin, Yankai Lin, Ning Ding, Xu Han, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Different tunes played with equal skill: Exploring a unified optimization subspace for parameter-efficient tuning. *arXiv preprint arXiv:2210.13311*, 2022.
120. Wenpeng Yin, Jamaal Hay, and Dan Roth. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. In *Proceedings of EMNLP-IJCNLP*, 2019.
121. Zichun Yu, Tianyu Gao, Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Automatic label sequence generation for prompting sequence-to-sequence models. *arXiv preprint arXiv:2209.09401*, 2022.
122. Weizhe Yuan, Graham Neubig, and Pengfei Liu. Bartscore: Evaluating generated text as text generation. In *Proceedings of NeurIPS*, 2021.
123. Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. In *Proceedings of EMC2-NIPS*, 2019.

124. Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: transformers for longer sequences. In *Proceedings of NeuIPS*, 2020.
125. Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of ACL*, 2022.
126. Eric Zelikman, Yuhuai Wu, and Noah D Goodman. StAR: Bootstrapping reasoning with reasoning. *arXiv preprint arXiv:2203.14465*, 2022.
127. Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. GLM-130B: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*, 2022.
128. Qingkai Zeng, Wenhao Yu, Mengxia Yu, Tianwen Jiang, Tim Weninger, and Meng Jiang. Tri-train: Automatic pre-fine tuning between pre-training and fine-tuning for sciner. In *Findings of EMNLP*, 2020.
129. Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.
130. Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. Ternarybert: Distillation-aware ultra-low bit bert. In *Proceedings of EMNLP*, 2020.
131. Zhengyan Zhang, Yuxian Gu, Xu Han, Shengqi Chen, Chaojun Xiao, Zhenbo Sun, Yuan Yao, Fanchao Qi, Jian Guan, Pei Ke, et al. Cpm-2: Large-scale cost-effective pre-trained language models. *AI Open*, 2:216–224, 2021.
132. Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: Enhanced language representation with informative entities. In *Proceedings of ACL*, 2019.
133. Zhengyan Zhang, Xu Han, Hao Zhou, Pei Ke, Yuxian Gu, Deming Ye, Yujia Qin, Yusheng Su, Haozhe Ji, Jian Guan, et al. CPM: A large-scale generative chinese pre-trained language model. *AI Open*, 2:93–99, 2021.
134. Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Moefication: Conditional computation of transformer models for efficient inference. *arXiv preprint arXiv:2110.01786*, 2021.
135. Mengjie Zhao, Tao Lin, Fei Mi, Martin Jaggi, and Hinrich Schütze. Masking as an efficient alternative to finetuning for pretrained language models. In *Proceedings of EMNLLP*, 2020.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Chapter 6
Graph Representation Learning

Cheng Yang, Yankai Lin, Zhiyuan Liu, and Maosong Sun

Abstract Graph structure, which can represent objects and their relationships, is ubiquitous in big data including natural languages. Besides original text as a sequence of word tokens, massive additional information in NLP is in the graph structure, such as syntactic relations between words in a sentence, hyperlink relations between documents, and semantic relations between entities. Hence, it is critical for NLP to encode these graph data with graph representation learning. Graph representation learning, also known as network embedding, has been extensively studied in AI and data mining. In this chapter, we introduce a variety of graph representation learning methods that embed graph data into vectors with shallow or deep neural models. After that, we introduce how graph representation learning helps NLP tasks.

6.1 Introduction

Graph is a natural way to represent objects and their relationships. As a typical non-Euclidean data structure, it provides a flexible way to model the interactions between individual units in our daily lives. For example, social media networks, citation graphs, biological networks, and recommendation systems can all be modeled as graph structures.

C. Yang
School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China
e-mail: yangcheng@bupt.edu.cn

Y. Lin
Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China
e-mail: yankailin@ruc.edu.cn

Z. Liu (✉) · M. Sun
Department of Computer Science and Technology, Tsinghua University, Beijing, China
e-mail: liuzy@tsinghua.edu.cn; sms@tsinghua.edu.cn

© The Author(s) 2023

Z. Liu et al. (eds.), *Representation Learning for Natural Language Processing*,
https://doi.org/10.1007/978-981-99-1600-9_6

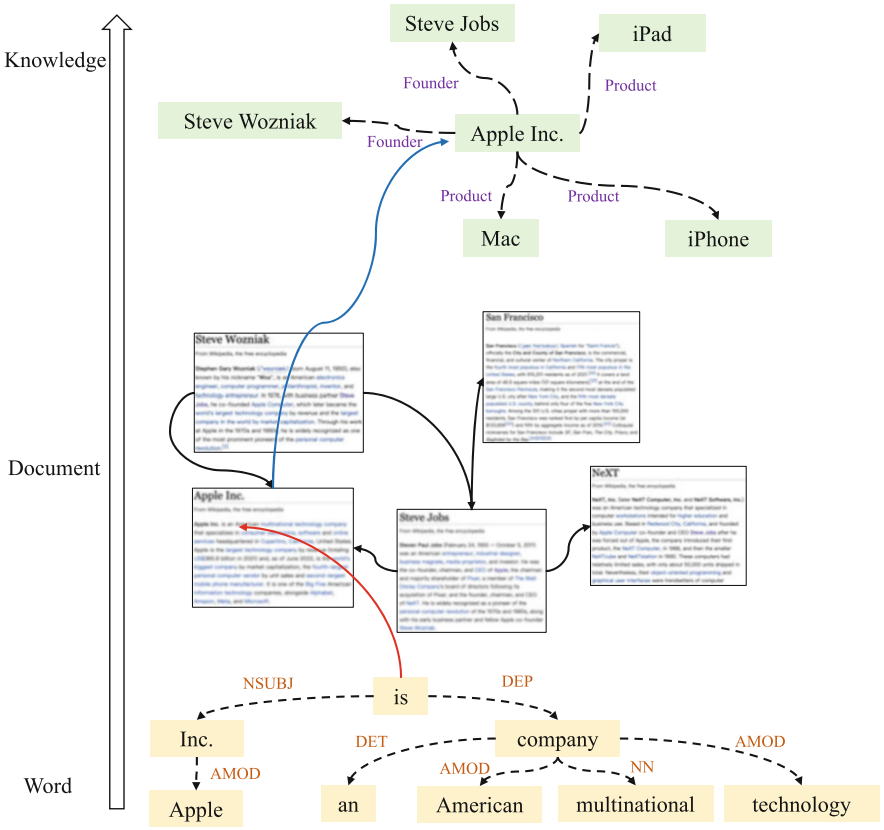


Fig. 6.1 An illustrative example of text-based graphs in different levels. The documents used in the figure are obtained from Wikipedia's official website

Graph structure plays an equally important role in the NLP area as the sequence form introduced in the previous chapters. As shown in Fig. 6.1, multiple granularities of text can be organized in graph forms:

1. **Word Level**. There are a variety of syntactic/semantic relations between the words within a sentence or a document. For example, we can obtain the dependency relations between words by dependency parsing or the coreference relations between words by coreference resolution. These syntactic/semantic relationships can effectively help us understand the compositional semantics of different constituents in the sentence and document. We can naturally regard words as nodes and their syntactic/semantic relations as edges, representing a sentence or a document as a relational graph.
2. **Document Level**. In the real world, documents usually connect with each other. For example, in the articles of Wikipedia, the entity mentions within an article may exist human-annotated hyperlinks to the articles describing these entities,

or a scientific paper may refer to other scientific papers as its related works. The connected documents may provide important background knowledge to comprehend the meaning of the document. We can represent the interactions between documents as a document graph by regarding documents as nodes and hyperlinks as edges.

3. **Knowledge Level**. Human knowledge, such as world, linguistic, commonsense, and domain knowledge is essential for understanding languages. In practice, most of this knowledge can be organized in a graph form. For example, the world knowledge graph in Chap. 9 regards entities as nodes and their relationships as edges. Representing human knowledge as graphs enables further complex reasoning over the document and knowledge.

It is critical to model the structural information in graph data, which could help models better understand, categorize, and reason text in NLP tasks. Graph representation learning aims to learn the low-dimensional representations of nodes or the entire graph. The geometric relationship in the low-dimensional semantic space should effectively reflect the structural information of the original graph, such as the global topological structure or the local graph connection.

In this chapter, we discuss how to properly represent graph data and their characteristics by starting from symbolic representations (Sect. 6.2). Then, we move to distributed representations including the local representations of nodes (Sects. 6.3 and 6.4) as well as the global representation of the whole graph (Sect. 6.5). We also introduce the recent advances of self-supervised learning on graphs (Sect. 6.6). Finally, we present how text is processed in graph form in downstream NLP tasks (Sect. 6.7), mainly for word, sentence, and document levels. We leave a detailed introduction of knowledge-level applications in Chap. 9.

6.2 Symbolic Graph Representation

A common practice is to denote the graph with its node set $\mathcal{V}$ and the edge set $\mathcal{E}$. We can naturally represent a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $e = (v_i, v_j) \in \mathcal{E}$ is a directed edge from node v_i to v_j or an undirected one between v_i and v_j. When processing graph data in a computer, we usually represent the connections in a graph as an adjacency matrix.

For an undirected graph, as shown in Fig. 6.2, we construct an adjacency matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$. If there is any edge between node v and node u, i.e., $(v, u) \in \mathcal{E}$, we have the corresponding element $\mathbf{A}_{vu} = \mathbf{A}_{uv} = 1$; otherwise $\mathbf{A}_{vu} = \mathbf{A}_{uv} = 0$.

For a directed graph, as shown in Fig. 6.3, $\mathbf{A}_{vu} = 1$ indicates there is an edge from node v to node u.

Moreover, for a weighted graph, we can store the weights of the edges instead of binary values in the adjacency matrix $\mathbf{A}$, as shown in Fig. 6.4.

In the era of statistical learning, such symbolic representations are widely used in graph-based NLP such as TextRank [67], where word and sentence graphs

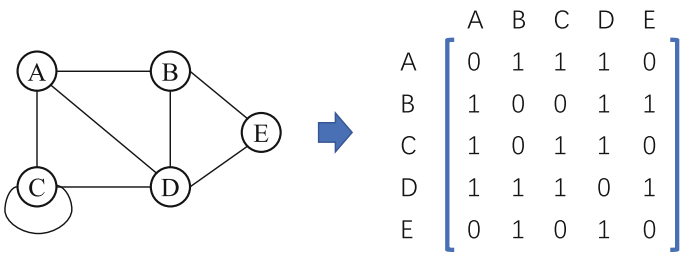


Fig. 6.2 The adjacency matrix representation for an undirected graph

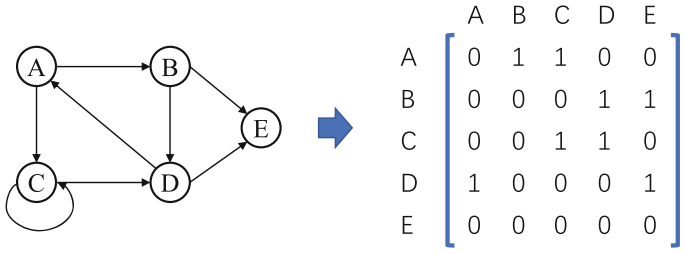


Fig. 6.3 The adjacency matrix representation for a directed graph

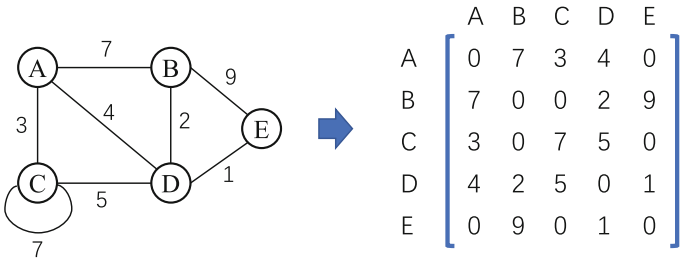


Fig. 6.4 The adjacency matrix representation for a weighted graph

can be respectively built for keyword extraction [6] and extractive document summarization [66]. Although convenient and straightforward, the representation of the adjacency matrix suffers from the scalability problem. Firstly, adjacency matrix $\mathbf{A}$ takes $|\mathcal{V}| \times |\mathcal{V}|$ storage space, which is usually unacceptable when $|\mathcal{V}|$ grows large. Secondly, the adjacency matrix is usually sparse, which means most of its entries are zeros. The data sparsity makes discrete algorithms applicable, but it is still hard to develop efficient algorithms for statistical learning [81].

6.3 Shallow Node Representation Learning

To address the above issues, shallow node representation learning methods propose to map nodes to low-dimensional vectors. Formally, the goal is to learn a d-dimensional vector representation $\mathbf{v} \in \mathbb{R}^d$ for every node $v \in \mathcal{V}$ in the graph. Learned node representations are supposed to capture the graph's structure information and then can be used as input features for downstream graph-related tasks. In this section, we will introduce several kinds of shallow node representation learning methods, including spectral clustering, shallow neural networks, and matrix factorization.

6.3.1 Spectral Clustering

Early methods of shallow node representation are usually based on spectral clustering, which typically computes the first k eigenvectors (or singular vectors) of an affinity matrix, such as adjacency or Laplacian matrix of the graph. Now we present several algorithms based on spectral clustering, including locally linear embedding, Laplacian Eigenmap, directed graph embedding, and latent social dimensions.

Locally Linear Embedding (LLE) LLE [88] assumes that the representations of a node and its neighbors lie in a locally linear patch of the manifold. In other words, a node's representation can be approximated by a linear combination of the representation of its neighbors. LLE uses the linear combination of neighbors to reconstruct the center node. Formally, the reconstruction error of all nodes can be expressed as

$$\mathcal{L}(\mathbf{W}, \mathbf{V}) = \sum_{v \in \mathcal{V}} \|\mathbf{v} - \sum_{u \in \mathcal{V}} \mathbf{W}_{vu}\mathbf{u}\|^2, \tag{6.1}$$

where $\mathbf{V} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the representation matrix containing all node representations $\mathbf{v}$ and $\mathbf{W}_{vu}$ is the learnable contribution coefficient of node u to v. LLE enforces $\mathbf{W}_{vu} = 0$ if v and u are not connected, i.e., $(v, u) \notin \mathcal{E}$. Further, the summation of a row of matrix $\mathbf{W}$ is set to 1, i.e., $\sum_{u \in \mathcal{V}} \mathbf{W}_{vu} = 1$.

Equation (6.1) is solved by alternatively optimizing weight matrix $\mathbf{W}$ and representation $\mathbf{V}$. The optimization over $\mathbf{W}$ can be solved as a least-squares problem. The optimization over $\mathbf{V}$ leads to the following optimization problem:

$$\mathcal{L}(\mathbf{W}, \mathbf{V}) = \sum_{v \in \mathcal{V}} \|\mathbf{v} - \sum_{u \in \mathcal{V}} \mathbf{W}_{vu}\mathbf{u}\|^2, \tag{6.2}$$

$$s.t. \sum_{v \in \mathcal{V}} \mathbf{v} = \mathbf{0}, \qquad |\mathcal{V}|^{-1} \sum_{v \in \mathcal{V}} \mathbf{v}^\top \mathbf{v} = \mathbf{I}_d, \tag{6.3}$$

where $\mathbf{I}_d$ denotes $d \times d$ identity matrix. The conditions in Eq. (6.3) ensure the uniqueness of the solution. The first condition enforces the center of all node representations to zero point, and the second condition guarantees different coordinates have the same scale, i.e., equal contribution to the reconstruction error.

The optimization problem can be formulated as the computation of eigenvectors of matrix $(\mathbf{I}_{|\mathcal{V}|} - \mathbf{W}^\top)(\mathbf{I}_{|\mathcal{V}|} - \mathbf{W})$, which is an easily solvable eigenvalue problem. More details can be found in the note [20].

Laplacian Eigenmap Laplacian Eigenmap [7] simply follows the idea that the representations of two connected nodes should be close. Specifically, the "closeness" is measured by the square of Euclidean distance. We use $\mathbf{D} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ to denote diagonal degree matrix where $\mathbf{D}_{vv}$ is the degree of node v. By defining the Laplacian matrix $\mathbf{L}$ as the difference between $\mathbf{D}$ and adjacency matrix A, we have $\mathbf{L} = \mathbf{D} - \mathbf{A}$. Laplacian Eigenmap algorithm wants to minimize the following objective:

$$\mathcal{L}(\mathbf{V}) = \sum_{\{v,u|(v,u)\in\mathcal{E}\}} \|\mathbf{v} - \mathbf{u}\|^2, \tag{6.4}$$

$$s.t.\ \mathbf{V}^\top \mathbf{D}\mathbf{V} = \mathbf{I}_d. \tag{6.5}$$

The cost function is the summation of the square loss of all connected node pairs, and the condition prevents the trivial all-zero solution caused by arbitrary scale. Equation (6.4) can be reformulated in matrix form as

$$\mathbf{V}^* = \underset{\mathbf{V}^\top \mathbf{D}\mathbf{V} = \mathbf{I}_d}{\arg\min} \operatorname{tr}(\mathbf{V}^\top \mathbf{L}\mathbf{V}), \tag{6.6}$$

where $\operatorname{tr}(\cdot)$ is the matrix trace function. The optimal solution $\mathbf{V}^*$ of Eq. (6.6) is the eigenvectors corresponding to d smallest nonzero eigenvalues of $\mathbf{L}$. Note that the Laplacian Eigenmap algorithm can be easily generalized to the weighted graph.

A significant limitation of both LLE and Laplacian Eigenmap is that they have a symmetric cost function, leading to both algorithms not being applied to directed graphs.

Directed Graph Embedding (DGE) DGE [17] generalizes Laplacian Eigenmap for both directed and undirected graphs based on a predefined transition matrix. For example, we can define a transition probability matrix $\mathbf{P} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ where $\mathbf{P}_{vu}$ denotes the probability that node v walks to u. The transition matrix defines a Markov random walk through the graph. We denote the stationary value of node v as π_v where $\sum_v \pi_v = 1$. The stationary distribution of random walks is commonly used in many ranking algorithms such as PageRank [74]. DGE designs a new cost function that emphasizes those important nodes with higher stationary values:

$$\mathcal{L}(\mathbf{V}) = \sum_{v\in\mathcal{V}} \pi_v \sum_{u\in\mathcal{V}} \mathbf{P}_{vu} \|\mathbf{v} - \mathbf{u}\|^2. \tag{6.7}$$

By denoting $\mathbf{M} = \text{diag}(\pi_1, \pi_2, \ldots, \pi_{|\mathcal{V}|})$, the cost function Eq. (6.7) can be reformulated as

$$\mathcal{L}(\mathbf{V}) = 2\text{tr}(\mathbf{V}^\top \mathbf{B}\mathbf{V}), \tag{6.8}$$

$$s.t.\ \mathbf{V}^\top \mathbf{M}\mathbf{V} = \mathbf{I}_d, \tag{6.9}$$

where

$$\mathbf{B} = \mathbf{M} - \frac{\mathbf{M}\mathbf{P} - \mathbf{P}^\top \mathbf{M}}{2}. \tag{6.10}$$

The condition Eq. (6.9) is added to remove an arbitrary scaling factor of solutions. Similar to Laplacian Eigenmap, the optimization problem can also be solved as a generalized eigenvector problem.

Latent Social Dimensions Latent social dimensions [103] introduce modularity [71] into the cost function instead of minimizing the distance between node representations in previous works. Modularity is a measurement that characterizes how far the graph is away from a uniform random graph. Given $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we assume that nodes $\mathcal{V}$ are divided into n nonoverlapping communities. By "uniform random graph," we mean nodes connect to each other based on a uniform distribution given their degrees. Then, the expected number of edges between v and u is $\deg(v)\frac{\deg(u)}{2|\mathcal{E}|}$. Then, the modularity Q of a graph is defined as

$$Q = \frac{1}{2|\mathcal{E}|}\sum_{v,u}\left(\mathbf{A}_{vu} - \frac{\deg(v)\deg(u)}{2|\mathcal{E}|}\right)\delta(v,u), \tag{6.11}$$

where $\delta(v,u) = 1$ if v and u belong to the same community and $\delta(v,u) = 0$ otherwise. Larger modularity indicates that the subgraphs inside communities are denser, which follows the intuition that a community is a dense well-connected cluster. Then, the problem is to find a partition that maximizes the modularity Q.

However, a hard clustering on modularity maximization is proved to be NP-hard. Therefore, they relax the problem to a soft case. Let $\mathbf{d} \in \mathbb{Z}_+^{|\mathcal{V}|}$ denotes the degrees of all nodes and $\mathbf{S} \in \{0,1\}^{|\mathcal{V}|\times n}$ denotes the community indicator matrix where $\mathbf{S}_{vc} = 1$ indicates node v belongs to community c and $\mathbf{S}_{vc} = 0$ otherwise. Then, we define modularity matrix $\mathbf{B}$ as

$$\mathbf{B} = \mathbf{A} - \frac{\mathbf{d}\mathbf{d}^\top}{2|\mathcal{E}|}, \tag{6.12}$$

and modularity Q can be reformulated as

$$Q = \frac{1}{2|\mathcal{E}|}\text{tr}(\mathbf{S}^\top \mathbf{B}\mathbf{S}). \tag{6.13}$$

By relaxing $\mathbf{S}$ to a continuous matrix, it has been proved that the optimal solution of $\mathbf{S}$ is the top-n eigenvectors of modularity matrix $\mathbf{B}$ [70]. Then, the eigenvectors can be used as node representations.

To conclude, spectral clustering-based methods often define a cost function that is linear or quadratic to the node representations. Then, the problems can be reformulated as a matrix form, and then solved by calculating the eigenvectors of the matrix. However, the computation of eigenvectors for large-scale matrices is both time- and space-consuming, limiting these methods from being applied in real-world scenarios.

6.3.2 Shallow Neural Networks

With the success of word2vec [68], many works resort to shallow neural networks for node representation learning. Typically, each node is assigned a vector of trainable parameters as its representation, and the parameters are trained by optimizing a certain objective via gradient descent.

DeepWalk DeepWalk [81] proposes a novel approach that introduces neural network techniques into graph representation learning for the first time. Compared with the aforementioned methods based on eigenvector computation, DeepWalk provides a faster way to learn low-dimensional node representations. The basic idea of DeepWalk is to adapt the well-known word representation learning algorithm word2vec [68] by regarding nodes as words and random walks as sentences.

Formally, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, DeepWalk uses node v to predict its neighboring nodes in short random walk sequences $u_{-w}, \ldots, u_{-1}, u_1, \ldots, u_w$ where w is the window size of skip-gram [68], which can be formulated as

$$\min \sum_{j=-w, j\neq 0}^{w} -\log P(u_j|v), \tag{6.14}$$

where it assumes the prediction probabilities of each node are independent and the overall loss integrates the losses of all nodes in every random walk.

DeepWalk assigns each node v with two representations: node representation $\mathbf{v} \in \mathbb{R}^d$ and context representation $\mathbf{v}^c \in \mathbb{R}^d$. Then, each probability $P(u|v)$ is formalized as a Softmax function over all nodes:

$$P(u|v) = \frac{\exp(\mathbf{v}^\top \mathbf{u}^c)}{\sum_{u' \in \mathcal{V}} \exp(\mathbf{v}^\top \mathbf{u}'^c)}. \tag{6.15}$$

LINE LINE [102] proposes a graph representation model which can handle large-scale graphs with arbitrary types: (un)directed or weighted. To characterize the interaction between nodes, LINE models the first-order proximity and second-order proximity of the graph.

Before we introduce the details of the algorithm, we can move one step back and see how the idea works. The modeling of first-order proximity, i.e., observed links, is the modeling of the adjacency matrix. As the adjacency matrix is usually too sparse, the modeling of second-order proximity, i.e., nodes with shared neighbors, can serve as complementary information to enrich the adjacency matrix and make it denser.

Formally, first-order proximity between node v and u is defined as the edge weight $\mathbf{A}_{vu}$ in the adjacency matrix. If nodes v and u are not connected, then the first-order proximity between them is 0.

Second-order proximity between node v and u is defined as the similarity between their neighbors. Let the row of node v in the adjacency matrix $\mathbf{A}(v,:)$ denote the first-order proximity between node v and other nodes. Then, the second-order proximity between v and u is defined as the similarity between $\mathbf{A}(v,:)$ and $\mathbf{A}(u,:)$. If they have no shared neighbors, the second-order proximity is 0.

To approximate first-order proximity $\hat{P}_1(v,u) = \frac{\mathbf{A}_{vu}}{\sum_{(v',u')\in\mathscr{E}}\mathbf{A}_{v'u'}}$, the joint probability between v and u is defined as

$$P_1(v,u) = \frac{1}{1+\exp(-\mathbf{v}^\top\mathbf{u})}. \tag{6.16}$$

To approximate second-order proximity $\hat{P}_2(u|v) = \frac{\mathbf{A}_{vu}}{\sum_{u'}\mathbf{A}_{vu'}}$, the probability that node u appears in v's context $P_2(u|v)$ is defined in the same form as Eq. (6.15). In this way, the second-order relationship between node representations is bridged by the context representations of their shared neighbors.

For parameter learning, the distances between $\hat{P}_1(v,u)$ and $P_1(v,u)$ as well as $\hat{P}_2(u|v)$ and $P_2(u|v)$ are minimized. In specific, LINE learns node representations for first-order and second-order proximities individually, and then concatenates them as output embeddings. Although LINE can effectively capture both first-order and second-order local topological information, it cannot be easily extended to capture higher-order global topological information.

Connection with Matrix Factorization We prove that DeepWalk algorithm with the skip-gram model is actually factoring a matrix $\mathbf{M}$ where each entry $\mathbf{M}_{ij}$ is the logarithm of the average probability that node v_i randomly walks to node v_j in fixed steps [127]. Intuitively, the matrix $\mathbf{M}$ is much denser than the adjacency or Laplacian matrices, and therefore can help representations encode more structural information. In practice, the entry value $\mathbf{M}_{ij}$ is estimated by random walk sampling. A more detailed proof can be found at our arxiv note [125]. Moreover, LINE is also proved to be equivalent to matrix factorization [83, 128], where the matrix is defined by the first- and second-order proximities.

To summarize, DeepWalk and LINE introduce shallow neural networks into graph representation learning. Thanks to the modeling ability of neural networks and the efficiency of shallow representations, these methods can outperform conventional graph representation learning methods such as spectral clustering-based algorithms, and are also efficient for large-scale graphs.

6.3.3 Matrix Factorization

Inspired by the connection between DeepWalk and matrix factorization, many research works turn to explore how to learn better node representations by regarding its learning phase as a matrix factorization problem. Note that the eigenvector decomposition in spectral clustering methods can also be seen as a special case of matrix factorization. Here we present two typical matrix factorization-based graph representation learning algorithms, GraRep [12] and TADW [127] in this subsection.

GraRep GraRep [12] directly follows the proof of matrix factorization form of DeepWalk. According to our proof [125], DeepWalk is actually factorizing a matrix $\mathbf{M}$ where $\mathbf{M} = \log \frac{\mathbf{A}+\mathbf{A}^2+\cdots+\mathbf{A}^K}{K}$. From the matrix factorization form, DeepWalk has considered the high-order topological information of the graph jointly. In contrast, GraRep proposes to regard different k-step information separately in graph representation learning, and can be divided into three steps:

- Calculate k-step transition probability matrix $\mathbf{A}^k$ for each $k = 1, 2, \ldots, K$.
- Obtain each k-step node representations.
- Concatenate all k-step node representations as the final node representations.

For the second step to obtain the k-step node representations, GraRep directly uses a typical matrix decomposition technique, i.e., SVD on $\mathbf{A}^k$. However, this algorithm is not very efficient, especially when k becomes large.

TADW As the first attributed network embedding algorithm where node features are also available for learning node representations, text-associated DeepWalk (TADW) [127] further generalizes the matrix factorization framework to take advantage of text information. As shown in Fig. 6.5, the main idea of TADW is to factorize node affinity matrix $\mathbf{M} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ into the product of three matrices, $\mathbf{W} \in \mathbb{R}^{d\times|\mathcal{V}|}$, $\mathbf{H} \in \mathbb{R}^{d\times f_t}$, and text feature matrix $\mathbf{T} \in \mathbb{R}^{f_t\times|\mathcal{V}|}$, where d and f_t are the rank and feature dimensions, respectively. Then, TADW concatenates $\mathbf{W}$ and $\mathbf{HT}$ as $2d$-dimensional node representations $\mathbf{V} = \text{concat}(\mathbf{W}; \mathbf{HT})$.

Now the question is how to build node affinity matrix $\mathbf{M}$ and how to extract text feature matrix $\mathbf{T}$ from the text information. Following the proof of matrix factorization form of DeepWalk, TADW set node affinity matrix $\mathbf{M}$ to a trade-

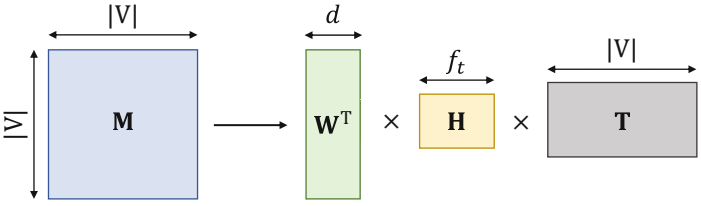


Fig. 6.5 Illustration of text-associated DeepWalk (TADW)

off between efficiency and effectiveness: factorizing the matrix $\mathbf{M} = (\mathbf{A} + \mathbf{A}^2)/2$ where $\mathbf{A}$ is the row-normalized adjacency matrix. In this way, $\mathbf{M}$ can encode both first-order and second-order proximities. For text feature matrix $\mathbf{T}$, TADW first constructs the TF-IDF matrix from the text of nodes, and then reduces the dimension of the TF-IDF matrix via SVD decomposition.

Formally, the model of TADW minimizes the following optimization function:

$$\min_{\mathbf{W},\mathbf{H}} ||\mathbf{M} - \mathbf{W}^\top \mathbf{H}\mathbf{T}||_F^2 + \frac{\lambda}{2}(||\mathbf{W}||_F^2 + ||\mathbf{H}||_F^2), \tag{6.17}$$

where λ is the regularization factor and $||\cdot||_F$ is the Frobenius norm. The optimization of parameters is processed by updating $\mathbf{W}$ and $\mathbf{H}$ iteratively via conjugate gradient descent.

To summarize, the shallow graph representation learning methods have shown outstanding ability in modeling various kinds of graphs, such as large-scale [153] or heterogeneous [110] graphs. They are also used in many application scenarios such as recommendation systems [91, 129]. However, they still have several drawbacks [38]. Firstly, the model capacity of shallow methods is usually limited, and leads to suboptimal performance in complex scenarios. Secondly, the representations of different nodes share no parameters, which makes the number of parameters grow linearly with the number of nodes. This leads to the problem of computational inefficiency. Thirdly, the shallow graph representation methods are typically transductive, and thus cannot generalize to new nodes in a dynamic graph without retraining.

6.4 Deep Node Representation Learning

To address the problems of shallow node representations, researchers propose to utilize deep neural networks to aggregate information from graph structure. In this section, we introduce several typical solutions for node representation learning, including autoencoder-based methods, graph convolutional networks, graph attention networks, graph recurrent networks, graph Transformers, and several extensions based on them. Most deep node representation learning methods assume node features are available, and stack multiple neural network layers for representation learning. Here we denote the initial feature vector of node v as $\mathbf{x}_v$ and the hidden representation of node v at the k-th layer as $\mathbf{h}_v^k$.

6.4.1 Autoencoder-Based Methods

Different from previous methods that use shallow neural networks to characterize the graph representations, structural deep network embedding (SDNE) [109]

employs a deep autoencoder to model more complex relationships between node representations. The main body of SDNE is a deep autoencoder whose input and output vectors are the initial node feature $\mathbf{x}_v$ and reconstructed feature $\hat{\mathbf{x}}_v$, respectively. The algorithm takes the representations from the intermediate layer as node embeddings and encourages the embeddings of connected nodes to be similar.

Formally, a deep autoencoder first compresses the input node feature into a low-dimensional intermediate vector and then tries to reconstruct the original input from the low-dimensional intermediate vector. Hence, the intermediate hidden representation can capture the information of the input node since we can recover the input features from them. Assume the input vector is $\mathbf{x}_v$, then the hidden representation of each layer k is defined as

$$\begin{aligned} \mathbf{h}_v^1 &= \text{Sigmoid}(\mathbf{W}^1 \mathbf{x}_v + \mathbf{b}^1), \\ \mathbf{h}_v^k &= \text{Sigmoid}(\mathbf{W}^k \mathbf{h}_v^{k-1} + \mathbf{b}^k), k = 2, 3 \ldots, K \ldots, \end{aligned} \tag{6.18}$$

where $\mathbf{W}^k$ and $\mathbf{b}^k$ are weighted matrix and bias vector of the k-th layer. We assume that the hidden representation of the K-th layer has the minimum dimension, and the intermediate layer $\mathbf{h}_v^K$ can be seen as the low-dimensional representation of node v. Afterward, we can compute the output $\hat{\mathbf{x}}_v$ by applying the reversed calculation process on $\mathbf{h}_v^K$. The optimization objective of the autoencoder is to minimize the difference between input vector $\mathbf{x}_v$ and output vector $\hat{\mathbf{x}}_v$:

$$\mathscr{L}_1 = \sum_{v \in \mathscr{V}} \| \hat{\mathbf{x}}_v - \mathbf{x}_v \|^2. \tag{6.19}$$

To encode the structure information, SDNE simply requires that the representations of connected nodes should be close to each other. Thus, the loss function is

$$\mathscr{L}_2 = \sum_{(v,u) \in \mathscr{E}} \| \mathbf{h}_v^K - \mathbf{h}_u^K \|^2. \tag{6.20}$$

Finally, the overall loss function is $\mathscr{L} = \mathscr{L}_1 + \alpha \mathscr{L}_2$, where α is a harmonic hyper-parameter. After the training process, $\mathbf{h}_v^K$ is taken as the representation of node v and used for downstream tasks.

Experimental results show that SDNE can effectively reconstruct the input graph and achieve better results in several downstream tasks. However, the deep neural network part of SDNE, i.e., the deep autoencoder, is isolated with graph structure during the feed-forward computation, which neglects high-order information interaction among nodes.

6.4.2 Graph Convolutional Networks

Graph convolutional networks (GCNs) aim to generalize convolutional operation from CNNs [55] to the graph domain. The success of CNNs comes from its local connection and multilayer [54] architectures, which may benefit graph modeling as well: (1) graphs are also locally connected; (2) multilayer architectures can help capture the hierarchical patterns in the graph. However, CNNs can only operate on regular Euclidean data like text (1D sequence) and images (2D grid), and cannot be directly transferred to the graph structure. In this subsection, we introduce how GCNs extend the convolutional operation to deal with the non-Euclidean graph data.

Mainstream GCNs usually adopt semi-supervised settings for training, while previous graph embedding methods are mostly unsupervised or self-supervised. Here we only introduce the encoder architectures of GCNs, and omit their loss functions which depend on downstream tasks. In specific, typical GCNs can be divided into spectral and spatial (nonspectral) approaches.

Spectral Methods From the signal processing perspective, the convolutional operation first transforms a signal to the spectral domain, then modifies the signal with a filter, and finally projects the signal back to the original domain [63]. Spectral GCNs follow the same process and define the convolution operator in the spectral domain of graph signals.

Formally, d-dimensional input representations $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ of a graph can be seen as d graph signals. Then, spectral GCNs are formulated as

$$\mathbf{H} = \mathscr{F}^{-1}(\mathscr{F}(\mathbf{g}) \odot \mathscr{F}(\mathbf{X})), \tag{6.21}$$

where $\mathbf{H}$ is the representations of all nodes, $\mathbf{g}$ is the filter in the spatial domain, $\mathscr{F}(\cdot)$ and $\mathscr{F}^{-1}(\cdot)$ indicate the graph Fourier transform (GFT) [9] and inverse GFT respectively, which can be defined as

$$\mathscr{F}(\mathbf{X}) = \mathbf{U}^{\top}\mathbf{X}, \mathscr{F}^{-1}(\mathbf{X}) = \mathbf{U}\mathbf{X}, \tag{6.22}$$

where $\mathbf{U}$ is the eigenvector matrix of the normalized graph Laplacian $\mathbf{L} = \mathbf{I}_{|\mathcal{V}|} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$. $\mathbf{A}$ is adjacency matrix and $\mathbf{D}$ is degree matrix.

In practice, we can use a learnable diagonal matrix $\mathbf{g}_{\theta}$ to approximate the spectral graph filter $\mathscr{F}(\mathbf{g})$, and the graph convolutional operation can be reformulated as

$$\mathbf{H} = \mathbf{U}\mathbf{g}_{\theta}\mathbf{U}^{\top}\mathbf{X}. \tag{6.23}$$

Intuitively, initial graph signals $\mathbf{X}$ are transformed into spectral domain by multiplying $\mathbf{U}^{\top}$. Then filter $\mathbf{g}_{\theta}$ performs the convolution, and $\mathbf{U}$ projects graph signals back to their original space. The above form of graph convolution is used in spectral network [10], the first spectral GCN method.

However, the original form in Eq. (6.23) has several limitations: (1) the filter $\mathbf{g}_\theta$ is not directly related to the graph structure; (2) the kernel size of the graph filter grows with the number of nodes in the graph, which may cause inefficiency and overfitting issues; (3) the calculation of $\mathbf{U}$ relies on computationally inefficient matrix factorization. Now we introduce two typical spectral GCNs improving the original formats, including ChebNet [23] and GCN [52].

ChebNet ChebNet [23] proposes to approximate $\mathbf{U}\mathbf{g}_\theta\mathbf{U}^\top$ by Chebyshev polynomials $\mathbf{T}_k(\cdot)$ up to K^{th}-order, which involve information within K-hop neighborhood. In this way, ChebNet does not need to compute matrix $\mathbf{U}$, and the number of learnable parameters is related to K instead of $|\mathcal{V}|$. Formally, the operation of ChebNet is reformulated as

$$\mathbf{H} = \sum_{k=0}^{K} \theta_k \mathbf{T}_k(\tilde{\mathbf{L}})\mathbf{X}, \tag{6.24}$$

where $\tilde{\mathbf{L}} = 2/\lambda_{max}\mathbf{L} - \mathbf{I}_{|\mathcal{V}|}$ is the normalized Laplacian matrix, λ_{max} is the largest eigenvalue of $\mathbf{L}$, and $\theta \in \mathbb{R}^K$ is a weight vector indicating Chebyshev coefficients. The Chebyshev polynomials are defined as

$$\begin{aligned} \mathbf{T}_0(\tilde{\mathbf{L}}) &= \mathbf{I}_{|\mathcal{V}|}, \\ \mathbf{T}_1(\tilde{\mathbf{L}}) &= \tilde{\mathbf{L}}, \\ \mathbf{T}_k(\tilde{\mathbf{L}}) &= 2\tilde{\mathbf{L}}\mathbf{T}_{k-1}(\tilde{\mathbf{L}}) - \mathbf{T}_{k-2}(\tilde{\mathbf{L}}). \end{aligned} \tag{6.25}$$

The K^{th}-order polynomial can be efficiently computed in a recursive manner, and the parameter number of the graph filter is reduced to K.

GCN GCN [52] is a first-order approximation of ChebNet. GCN reveals that ChebNet may suffer from the overfitting problem when handling graphs with very wide node degree distributions. Hence, GCN limits the maximum order of Chebyshev polynomials to $K = 1$, and the equation is simplified to the following form with two trainable scalars θ'_0 and θ'_1:

$$\begin{aligned} \mathbf{H} &= \theta'_0\mathbf{X} + \theta'_1\left(\mathbf{L} - \mathbf{I}_{|\mathcal{V}|}\right)\mathbf{X} \\ &= \theta'_0\mathbf{X} - \theta'_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}. \end{aligned} \tag{6.26}$$

GCN further reduces the number of parameters to address overfitting by setting $\theta'_0 = -\theta'_1 = \theta$. And the equation is reformulated as

$$\mathbf{H} = \theta\left(\mathbf{I}_{|\mathcal{V}|} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)\mathbf{X}. \tag{6.27}$$

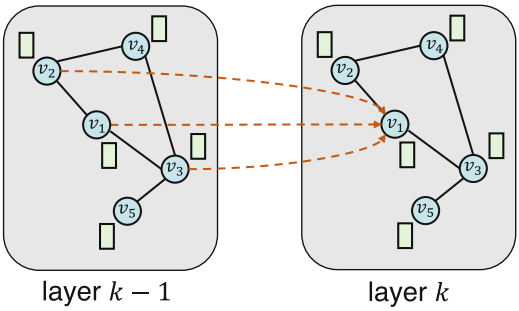


Fig. 6.6 Illustration of spatial GCNs. In the feed-forward computation, each node aggregates information from the representations of its neighbors and itself

To summarize, spectral GCNs can effectively capture complex global patterns in graphs with spectral graph filters. However, the learned filters of the spectral methods usually depend on the Laplacian eigenbasis of the graph structure. This leads to the problem that a spectral-based GCN model learned on a specific graph cannot be directly transferred to another graph.

Spatial Methods Different from spectral GCNs, spatial GCNs define graph convolutional operation by directly aggregating information on spatially close neighbors, which is also known as the message-passing process. As shown in Fig. 6.6, the representation $\mathbf{h}_v^k$ of node v at k-th layer can be seen as a function aggregating the representations of v and its neighbors at $(k-1)$-th layer:

$$\mathbf{h}_v^k = f(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1} | \forall u \in \mathcal{N}_v\}), \tag{6.28}$$

where $\mathcal{N}_v$ is the neighbor set of node v.

The key challenge of spatial GCNs is how to define the convolutional operation to satisfy the nodes with different degrees while maintaining the local invariance of CNNs. In this subsection, we introduce two widely used spatial GCNs, including Neural FPs and GraphSAGE.

Neural FPs Neural FPs [27] propose to use different weight matrices for nodes with different-sized neighborhoods:

$$\begin{aligned} \mathbf{z}_v^k &= \mathbf{h}_v^{k-1} + \sum_{u \in \mathcal{N}_v} \mathbf{h}_u^{k-1}, \\ \mathbf{h}_v^k &= \sigma(\mathbf{W}_{|\mathcal{N}_v|}^k \mathbf{z}^k), \end{aligned} \tag{6.29}$$

where $\mathbf{W}_{|\mathcal{N}_v|}^k$ is the weight matrix for nodes with degree $|\mathcal{N}_v|$ at layer k and $\sigma(\cdot)$ is a nonlinear function such as Sigmoid. Neural FPs require learning weight matrices for all node degrees in the graph. Hence, when applied to large-scale graphs with diverse node degrees, it cannot capture the invariant information among different node degrees, and needs more parameters as node degrees get larger.

GraphSAGE GraphSAGE [37] transfers GCNs to handle the inductive setting, where the representations of new nodes should be computed without retraining. Instead of utilizing the full set of neighbors, GraphSAGE learns graph representations by uniformly sampling a fixed-size neighbor set from each node's local neighborhood. GraphSAGE can be formulated as

$$\begin{aligned} \mathbf{h}_{\mathcal{N}_v}^k &= \text{Aggregate}(\{\mathbf{h}_u^{k-1} | \forall u \in \mathcal{N}_v\}), \\ \mathbf{h}_v^k &= \sigma(\mathbf{W}^k \, \text{concat}(\mathbf{h}_v^{k-1}; \mathbf{h}_{\mathcal{N}_v}^k)), \end{aligned} \tag{6.30}$$

where $\mathcal{N}_v$ is the sampled neighbor set of node v and the aggregator functions Aggregate$(\cdot)$ usually utilize the following three types:

1. Mean aggregator. By utilizing a mean-pooling aggregator, GraphSAGE can be viewed as the inductive version of the original transductive GCN framework [52], which can be formulated as

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}^k \cdot \text{Mean}\left(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1} | \forall u \in \mathcal{N}_v\}\right)\right). \tag{6.31}$$

2. Max-pooling aggregator. Max-pooling aggregator first feeds each neighbor's hidden representation into a fully connected layer and then utilizes a max-pooling operation to the obtained representations of the node's neighbors. It can be formulated as

$$\mathbf{h}_{\mathcal{N}_v}^k = \text{Max}(\{\sigma(\mathbf{W}^k \mathbf{h}_u^{k-1} + \mathbf{b}^k) | \forall u \in \mathcal{N}_v\}). \tag{6.32}$$

3. LSTM aggregator. GraphSAGE also proposes to use an LSTM-based aggregator with a stronger expressive capability. Since LSTMs process inputs sequentially, GraphSAGE randomly permutes node v's neighbors to adapt LSTMs.

In summary, GCNs extend the idea of CNNs into the graph domain, enabling models to capture local connectivity of the graph, and have shown their superior abilities in a wide range of downstream tasks compared to the previous autoencoder-based methods. Even now, we can still get state-of-the-art performance by equipping vanilla GCNs with proper training strategies [48] or knowledge distillation methods [123, 124].

6.4.3 Graph Attention Networks

The attention mechanism has shown its strong ability to consider instance importance in learning representations in many NLP applications, such as machine translation [3, 33, 105] and machine reading [18]. Hence, many works [106, 146] focus on generalizing the attention mechanism to the graph domain and hope graph neural networks (GNNs) with attention-based operators can achieve better results

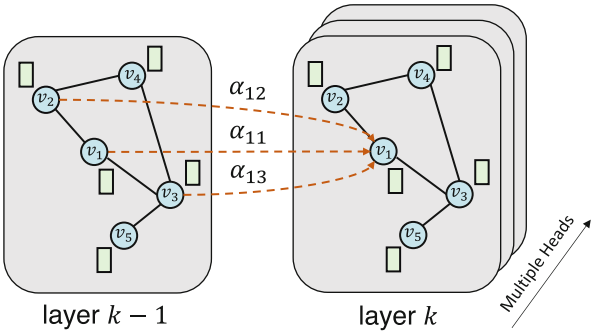


Fig. 6.7 Illustration of GATs. Compared with spatial GCNs, GATs assign different aggregation weights to different neighbors, and employ multiple parallel attention heads for computation

by considering the importance of neighbors. Figure 6.7 illustrates the architecture of graph attention networks (GATs).

GAT GAT [106] proposes to adopt the self-attention mechanism for the information aggregation of GNNs. Specifically, each node's representation is calculated by attending to its neighbors:

$$\mathbf{h}_v^k = \sigma\left(\sum_{u \in \mathcal{N}_v \cup \{v\}} \alpha_{vu}^k \mathbf{W}^k \mathbf{h}_u^{k-1}\right), \tag{6.33}$$

where $\sigma(\cdot)$ is a nonlinear function and α_{vu}^k is the attention coefficient of node pair (v, u) at the k-th layer, which is normalized over node v's neighbors:

$$\alpha_{vu}^k = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top \text{concat}(\mathbf{W}^k \mathbf{h}_v^{k-1}; \mathbf{W}^k \mathbf{h}_u^{k-1})\right)\right)}{\sum_{m \in \mathcal{N}_v \cup \{v\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top \text{concat}(\mathbf{W}^k \mathbf{h}_v^{k-1}; \mathbf{W}^k \mathbf{h}_m^{k-1})\right)\right)}, \tag{6.34}$$

where $\mathbf{W}^k$ is the weight matrix of a shared linear transformation applied to every node, $\mathbf{a}$ is a learnable weight vector and LeakyReLU$(\cdot)$ is a nonlinear function.

Moreover, GAT utilizes the multi-head attention mechanism similar to [105] to further aggregate different types of information. Specifically, GAT concatenates (or averages) the output representations of M independent attention heads:

$$\mathbf{h}_v^k = \|_{m=1}^M \sigma\left(\sum_{u \in \mathcal{N}_v \cup \{v\}} \alpha_{vu}^{k,m} \mathbf{W}_m^k \mathbf{h}_u^{k-1}\right), \tag{6.35}$$

where $\alpha_{vu}^{k,m}$ is the attention coefficient from the m-th attention head at the k-th layer, $\mathbf{W}_m^k$ is the transform matrix for the m-th attention head, and $\|$ is the concatenation operation.

To summarize, by incorporating the attention mechanism into the information aggregating phase of spatial GCNs, GATs can assign proper weights to different neighbors and offer better interpretability. The ensemble of multiple attention heads further increases the model capacity and brings performance gains over GCNs.

6.4.4 Graph Recurrent Networks

To capture the dependency between two distant nodes by a graph encoder, one has to stack many GNN layers so that the information can propagate from one to another. However, stacking too many GNN layers in the feed-forward computation will cause the over-smoothing issue, which makes node representations less discriminative and harms the performance. Inspired by the success of gated recurrent unit (GRU) [19] and LSTM [42] in modeling long-term dependency in NLP, graph recurrent networks (GRNs) propose to equip the information aggregation of GCNs with gate mechanisms. Similar to the usage in RNNs, the gate mechanisms allow the information to propagate farther without severe gradient vanishment or over-smoothing issues. In this way, GRNs can improve the model's ability in capturing the long-range dependency across the graph. In this subsection, we introduce several variants of GRNs, including GGNN, Tree-LSTM, and Graph LSTM.

Gated Graph Neural Network (GGNN) GGNN [57] introduces a GRU-like function to improve the information propagation of the vanilla GCN architecture. In each layer, GGNN updates the representations of nodes by combining the information of their neighbors and themselves with the update and reset gates. Specifically, the recurrence of each layer in GGNN is defined as

$$\begin{aligned}
\mathbf{a}_v^k &= \sum_{u \in \mathcal{N}_v} \mathbf{h}_u^{k-1} + \mathbf{b}, \\
\mathbf{z}_v^k &= \text{Sigmoid}\left(\mathbf{W}_z \mathbf{a}_v^k + \mathbf{U}_z \mathbf{h}_v^{k-1}\right), \\
\mathbf{r}_v^k &= \text{Sigmoid}\left(\mathbf{W}_r \mathbf{a}_v^k + \mathbf{U}_r \mathbf{h}_v^{k-1}\right), \\
\widetilde{\mathbf{h}}_v^k &= \tanh\left(\mathbf{W} \mathbf{a}_v^k + \mathbf{U}(\mathbf{r}_v^k \odot \mathbf{h}_v^{k-1})\right), \\
\mathbf{h}_v^k &= (1 - \mathbf{z}_v^k) \odot \mathbf{h}_v^{k-1} + \mathbf{z}_v^k \odot \widetilde{\mathbf{h}}_v^k,
\end{aligned} \tag{6.36}$$

where $\mathbf{a}_v^k$ represents node v's neighborhood information, $\mathbf{b}$ is the bias vector, $\widetilde{\mathbf{h}}_v^k$ is the candidate representation, $\mathbf{z}$ and $\mathbf{r}$ are the update and reset gates, and $\mathbf{W}$ and $\mathbf{U}$ are weight matrices.

Typical RNNs can also be seen as a special case of GGNN, where the graph is a chain structure. Hence, GRNs like GGNN have been widely used in language models. Note that tree structure is very popular in text data, such as the dependency parsing tree. Next, we introduce Tree-LSTM [101], which extends GRNs to model the tree structure.

Tree-LSTM Tree-LSTM [101] uses an LSTM-based unit with input/output gates $\mathbf{i}_v/\mathbf{o}_v$ and memory cell $\mathbf{c}_v$ to update representation $\mathbf{h}_v$ of each tree node v. There are two variants of Tree-LSTM: Child-sum Tree-LSTM and N-ary Tree-LSTM. Instead of using a unified forget gate like LSTM, Child-sum Tree-LSTM assigns a forget gate $\mathbf{f}_{vm}$ for each child m of node v, which allows tree node v to adaptively gather information from its children. N-ary Tree-LSTM requires each node to have at most N children, and assigns different learnable parameters for each child. Child-sum Tree-LSTM is suitable for trees whose children are unordered, and thus can be used for modeling dependency trees. N-ary Tree-LSTM can characterize the diverse relational information for each node's children, and thus is usually used to model constituency parsing tree structure. Here we only present the formulas of Child-sum Tree-LSTM:

$$\begin{aligned}
\widetilde{\mathbf{h}}_v^{k-1} &= \sum_{m \in \mathcal{N}_v^e} \mathbf{h}_m^{k-1}, \\
\mathbf{i}_v^k &= \text{Sigmoid}\left(\mathbf{W}_i \mathbf{x}_v + \mathbf{U}_i \widetilde{\mathbf{h}}_v^{k-1} + \mathbf{b}_i\right), \\
\mathbf{f}_{vm}^k &= \text{Sigmoid}\left(\mathbf{W}_f \mathbf{x}_v + \mathbf{U}_f \mathbf{h}_m^{k-1} + \mathbf{b}_f\right), \\
\mathbf{o}_v^k &= \text{Sigmoid}\left(\mathbf{W}_o \mathbf{x}_v + \mathbf{U}_o \widetilde{\mathbf{h}}_v^{k-1} + \mathbf{b}_o\right), \\
\mathbf{u}_v^k &= \tanh\left(\mathbf{W}_u \mathbf{x}_v + \mathbf{U}_u \widetilde{\mathbf{h}}_v^{k-1} + \mathbf{b}_u\right), \\
\mathbf{c}_v^k &= \mathbf{i}_v^k \odot \mathbf{u}_v^k + \sum_{m \in \mathcal{N}_v} \mathbf{f}_{vm}^k \odot \mathbf{c}_m^{k-1}, \\
\mathbf{h}_v^k &= \mathbf{o}_v^k \odot \tanh(\mathbf{c}_v^k),
\end{aligned} \tag{6.37}$$

where $\mathcal{N}_v^e$ is the children set of node v and $\mathbf{x}_v$ is the input representation for tree node v. Readers can refer to the original paper [101] for the details of N-ary Tree-LSTM variant.

Graph LSTM Graph LSTM [78, 144] proposes to adapt Tree-LSTM to model the graph structure, and utilizes different weight matrices to represent different labels on the edges. Formally, assume that the edge label between node v and its child m is l. Compared with Eq. (6.37) in Tree-LSTM, Graph LSTM uses label-specific weight matrix $\mathbf{U}_l$ to compute relevant gates and hidden states.

In summary, GRNs with gate mechanisms can effectively model the long-range dependencies between distant nodes, which is very important in text modeling. By

adapting to tree-structure data, GRNs can also handle the diverse syntactic and semantic relations in text.

6.4.5 Graph Transformers

Transformer [105] has set off a craze in both NLP and CV areas [26, 84]. Based on the powerful self-attention architecture, graph Transformer networks [53, 87, 138] have been proposed to improve the expressive ability of GNNs. The core idea is to leverage the Transformer architecture to capture long-range relationships between distant nodes. Compared with GRNs, graph Transformers can benefit the advantages of Transformers against LSTMs.

Connections with Transformers in Text Modeling In graph Transformers, nodes are the basic units instead of words, and self-attention mechanism is then performed between all node pairs. In this way, all nodes are directly connected regardless of the original graph structure. To utilize the original topology information, current graph Transformers focus on the modifications of input features and attention coefficients. Other operations including multi-head ensembling, feed-forward network, and layer normalization remain unchanged. Now we will introduce three typical graph Transformers, including Graphormer [138], GraphTrans [116], and SAT [15].

Graphormer Graphormer [138] designs three structural encoding modules to inject graph structure information into the Transformer architecture. In specific, *centrality encoding* adds node degrees to the input which can indicate the importance of different nodes:

$$\mathbf{h}_v^0 = \mathbf{x}_v + \mathbf{z}^-_{\deg^-(v)} + \mathbf{z}^+_{\deg^+(v)}, \tag{6.38}$$

where $\mathbf{x}_v$ is the feature vector of node v and $\mathbf{z}^-, \mathbf{z}^+$ are learnable embedding vectors indexed by node in-degree and out-degree.

Spatial encoding of node distances and edge encoding of edge features serve as bias terms for the attention coefficients in the self-attention layer:

$$\alpha^k_{vu} = (\mathbf{W}_Q \mathbf{h}_v^{k-1})^\top (\mathbf{W}_K \mathbf{h}_u^{k-1})/\sqrt{d} + b_{\phi(v,u)} + c_{vu}, \tag{6.39}$$

where α^k_{vu} is the attention coefficient between node v and u in the self-attention layer, $\mathbf{W}_Q, \mathbf{W}_K$ are weight matrices, d is the hidden dimension, $b_{\phi(v,u)}$ is the learnable scalar indexed by the distance $\phi(v, u)$ between v and u, and c_{vu} is the scalar derived from the edge features between v and u.

To take advantage of deep models in utilizing structure information, GraphTrans [116] and SAT [15] propose to integrate GNN and Transformer architectures for modeling.

GraphTrans GraphTrans [116] directly stacks a Transformer module on the top of a GNN module. In other words, the output node representations of the GNN are used as the input features of the Transformer. GraphTrans adopts a special `[CLS]` token which connects to all other nodes, and the representation of `[CLS]` token after the Transformer is taken as the graph representation. Hence, the Transformer can also be seen as a pooling operator for GNN. As a result, GraphTrans can capture both the local structured information and long-range relationships on graphs at the same time.

SAT SAT [15] proves that modifying the position encoding module in standard Transformers could not fully capture the structural similarity between nodes on a graph. To address this problem, SAT defines attention coefficients in the Transformer by the similarity of GNN-based representations. In this way, GNN serves as a structure extractor to integrate more information into self-attention layers.

In summary, graph Transformer architecture can model long-range relationships on a graph and go beyond the limitations of traditional deep GNNs, such as over-smoothing. Compared with GNNs limited by the Weisfeiler-Lehman test [119], the Transformer-based methods become more expressive. Besides, GNNs can also be integrated into graph Transformers to better utilize the graph topology information.

6.4.6 Extensions

In this subsection, we will talk about several typical extensions of GNNs, including skip connection, neighborhood sampling, and the modeling of diverse graph types, which can respectively improve the effectiveness, efficiency, and generalizability of GNNs.

GNNs with Skip Connection Theoretically, we can enhance the expressive ability by stacking more layers of GNNs. However, existing works find that deeper GNNs do not perform better in downstream tasks and even perform worse [52]. Chen et al. [14] further attribute this phenomenon to the low information-noise ratio received by the nodes in deep GNNs. The residual network [40], which has been verified in the computer vision community, is a straightforward solution to the problem. But researchers find that deep GNNs with residual connections still perform worse compared to the two-layer GNNs. Therefore, Rahimi et al. [85] and Xu et al. [120] further explore how to enhance the performance of GNNs with skip connections. Inspired by the idea from the highway network [160], Rahimi et al. [85] employ the layer-wise gate mechanism, and the performance can peak at four layers. Formally, the Highway GCN can be defined as

$$\begin{aligned} \mathbf{T}(\mathbf{h}_v^{k-1}) &= \text{Sigmoid}\left(\mathbf{W}^{k-1}\mathbf{h}_v^{k-1} + \mathbf{b}^{k-1}\right), \\ \mathbf{h}_v^k &= \mathbf{h}_v^k \odot \mathbf{T}(\mathbf{h}_v^{k-1}) + \mathbf{h}_v^{k-1} \odot (1 - \mathbf{T}(\mathbf{h}_v^{k-1})). \end{aligned} \tag{6.40}$$

Besides, Xu et al. [120] present the jump knowledge network, which selects representations from all intermediate layers as final node representations. The selecting mechanism enables the jump knowledge network to adaptively pick the reasonable neighborhood information for each node.

GNNs with Neighborhood Sampling The vanilla GNN [89] has several limitations: (1) the computation is based on the entire graph Laplacian matrix, and thus computationally expensive for large graphs; (2) it is trained and specialized for a given graph, and cannot be transferred to another graph. To address the problems, the aforementioned GraphSAGE [37] first samples neighborhood nodes of a target node, and then aggregates the representations of all sampled nodes. Thus, GraphSAGE can get rid of the graph Laplacian and can be applied to unseen nodes. Ying et al. [139] further propose the importance-based sampling method PinSage, which simulates random walks starting from target nodes and samples the neighborhood set according to the normalized visit counts. Instead of sampling neighbors for each node, Chen et al. [16] propose FastGCN, which directly samples the receptive field for each layer. In other words, only sampled nodes can participate in layer-wise information propagation. FastGCN sets the sampling importance of nodes according to their degrees, and tends to keep the nodes with larger degrees. Besides, Huang et al. [45] introduce a parameterized and trainable sampler, which performs layer-wise sampling based on the previous layer.

GNNs for Graphs of Diverse Types The vanilla GNN [89] is designed for undirected graphs, the simplest graph format. However, as shown in Fig. 6.8, we have graphs of diverse types in real-world scenarios. Now we will introduce several extensions of GNNs to deal with directed, heterogeneous, or dynamic graphs:

Directed Graphs Directed edges contain extra information compared to undirected ones. For example, a famous person in a social network may be followed by lots of other users. Usually, her influence is stronger than most of her followers. This suggests that GNNs should treat the information propagation process in two edge

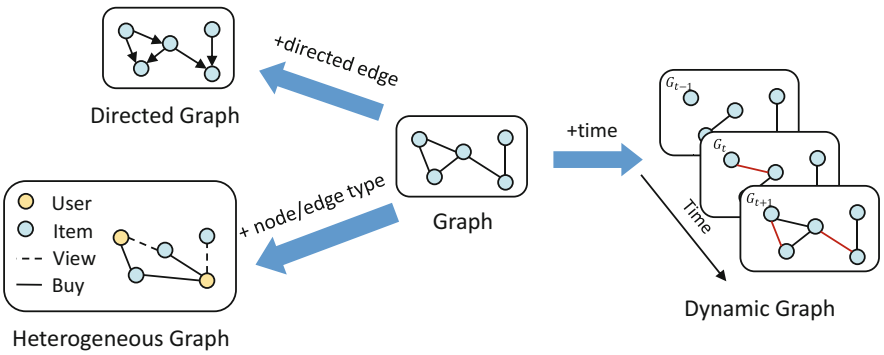


Fig. 6.8 Diverse types of graphs

directions differently. DGP [49] defines two graphs where nodes are respectively connected to all their ancestors or descendants. We correspondingly denote the normalized adjacency matrices as $\mathbf{D}_p^{-1}\mathbf{A}_p$ and $\mathbf{D}_c^{-1}\mathbf{A}_c$. The encoder of DGP can be formulated as

$$\mathbf{H}^k = \sigma(\mathbf{D}_p^{-1}\mathbf{A}_p\sigma(\mathbf{D}_c^{-1}\mathbf{A}_c\mathbf{H}^{k-1}\mathbf{W}_c)\mathbf{W}_p), \tag{6.41}$$

where $\mathbf{H}^k$ is the representation matrix of all nodes in the k-th layer, $\mathbf{W}_p$ and $\mathbf{W}_c$ are weight matrices, and $\sigma(\cdot)$ is a nonlinear function. In this way, the information propagations of two directions are processed separately.

Heterogeneous Graphs A heterogeneous graph [92] has several kinds of nodes. For example, for the graph in the shopping recommendation system, we have user nodes, item nodes, etc. The simplest way to process heterogeneous graphs is to consider the node type information in their input features, i.e., convert the node type information into a one-hot feature vector and concatenate it to the original features. Besides the simple feature-based approach, GraphInception [150] introduces the concept of meta-path into the information propagation on heterogeneous graphs. GraphInception utilizes GNNs to perform information propagation on homogeneous subgraphs, which are extracted based on human-designed meta-paths. At last, it concatenates the outputs from different homogeneous GNNs as final node representations. Wang et al. [111] further propose the heterogeneous graph attention network (HAN) with node-level and meta-path-level attention mechanisms. In addition, some works [112, 154] also consider the modeling of network schema [99], which is a meta template of a heterogeneous graph indicating node types and their relations.

Besides, there are many graphs containing edges with weights or types as additional information. A typical way to handle such graphs is to build a bipartite graph, where the original edges are converted into nodes linking to the original endpoint nodes, and the type information is thus converted to node type information. Another way is to assign different propagation weight matrices for different edge types. However, if a graph has lots of edge types, the parameter numbers will be large. To address the problem, R-GCN [90] introduces two regularization tricks that can decompose the transformation matrix W_r of type r to a set of base transformations shared among all edge types. Thus, R-GCN can reduce the number of parameters and capture the relationship between different edge types.

Dynamic Graphs Graphs are usually dynamic and vary over time. For example, a user in a social network may newly follow another user. To model the graph structure changing over time, DCRNN [56] and STGCN [143] first capture the static graph information at each time step by GNNs and then feed the output representation into a sequence model like RNNs. In addition, Structural-RNN [47] and ST-GCN [122] extend static graph structure with temporal connections and apply conventional GNNs on the extended graphs, which can capture structural and temporal information at the same time. MetaDyGNN [131] combines GNNs with meta-learning for few-shot link prediction in dynamic graphs.

6.5 From Node Representation to Graph Representation

In previous sections, we introduce how to represent nodes in a graph, from shallow node representation to deep node representation. In many scenarios, we also need to compute the representation of an entire graph or a specific subgraph. Inspired by the pooling operation in NLP and CV areas, graph pooling is then designed for obtaining the graph representation from node representations. Here we present two typical groups of graph pooling methods, namely, flat pooling and hierarchical pooling.

6.5.1 Flat Pooling

Flat pooling assumes a flat graph structure to generate graph representation, which includes max/mean/sum pooling as simple node pooling methods, and SortPooling [147] considering node ordering of structural roles.

Simple Node Pooling Similar to the pooling operation in NLP and CV, we can directly apply node-wise max/mean/sum operators on top of node representations for graph pooling. The graph representation can be formulated as

$$\begin{cases} \max\limits_{v \in \mathcal{V}} \mathbf{v}, & \text{(Graph max-pooling)} \\ \dfrac{1}{|\mathcal{V}|} \sum\limits_{v \in \mathcal{V}} \mathbf{v}, & \text{(Graph mean-pooling)} \\ \sum\limits_{v \in \mathcal{V}} \mathbf{v}. & \text{(Graph sum-pooling)} \end{cases} \tag{6.42}$$

The above pooling operators are general and parameter-free, but completely neglect the graph structure.

SortPooling SortPooling [147] first sorts node representations by their structural roles, which enables a consistent node ordering in different graphs and makes it possible to train typical neural networks on sorted node representations for pooling. In particular, SortPooling feeds the sorted representations into a 1-D CNN to get the graph representation, and makes the graph pooling operation to keep more information of global graph topology.

Though simple and effective, flat pooling ignores the hierarchical structure of nodes in a graph, e.g., nodes, subgraphs, and communities, thus leading to suboptimal graph representations.

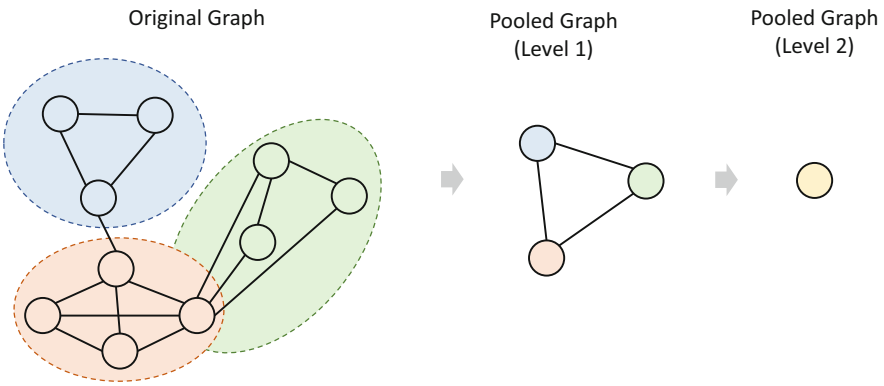


Fig. 6.9 The illustration of DiffPool. The figure is redrawn from Figure 1 of [140]

6.5.2 Hierarchical Pooling

The basic idea of hierarchical pooling is to group structure-related nodes into clusters to form a subgraph recursively, and obtain the graph representation layer by layer. Next, we will introduce two typical hierarchical pooling operations.

DiffPool DiffPool [140] proposes to learn hierarchical representations at the top of node representations and can be combined with various node representation learning methods in an end-to-end fashion. As shown in Fig. 6.9, DiffPool learns a differentiable soft cluster assignment for each node, and then maps nodes to a set of clusters layer by layer.

Formally, let $\mathbf{S}^k \in \mathbb{R}^{C_k \times C_{k+1}}$ denote the learned cluster assignment matrix at the k-th layer, where $\mathbf{S}^k_{vc}$ indicates whether node v belongs to cluster c at k-th layer, and C_k is the number of clusters in each layer. With the cluster assignment matrix $\mathbf{S}^k$, we can then calculate the adjacency matrix $\mathbf{A}^{k+1} \in \mathbb{R}^{C_{k+1} \times C_{k+1}}$ for the next layer by the connectivity strength between learned clusters in $\mathbf{S}^k$:

$$\mathbf{A}^{k+1} = {\mathbf{S}^k}^\top \mathbf{A}^k \mathbf{S}^k. \tag{6.43}$$

Then the output node representations $\mathbf{H}^{k+1}$ are computed by GNN encoder:

$$\mathbf{H}^{k+1} = \mathrm{GNN}(\mathbf{A}^{k+1}, \mathbf{X}^{k+1}), \tag{6.44}$$

where input node representations $\mathbf{X}^{k+1}$ are obtained by aggregating the output representations from the k-th layer:

$$\mathbf{X}^{k+1} = {\mathbf{S}^k}^\top \mathbf{H}^k. \tag{6.45}$$

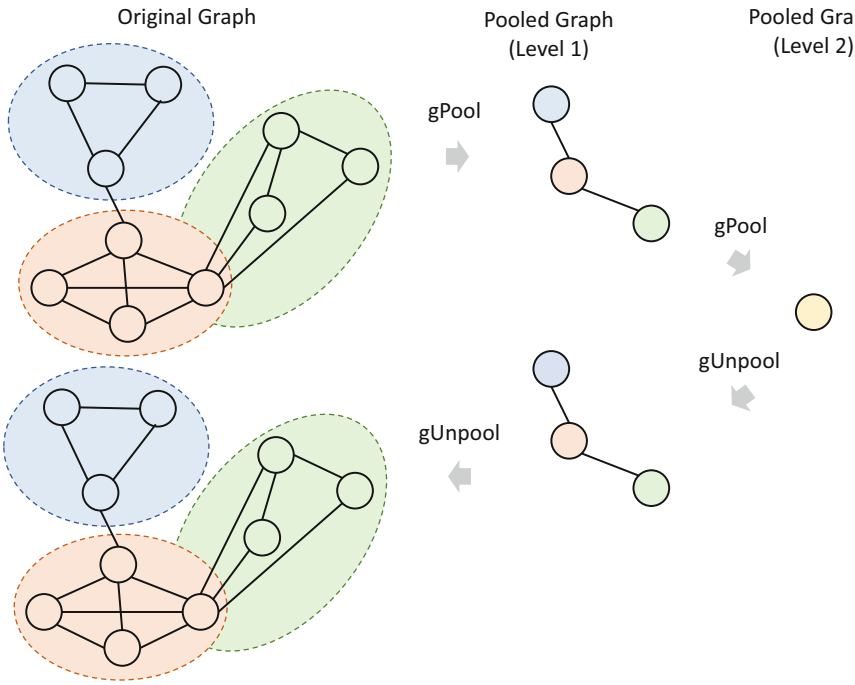


Fig. 6.10 Illustration of gPool and gUnpool. The figure is redrawn from Figure 2 of [32]

DiffPool predefines the number of clusters for each layer, and applies another GNN on the coarsened adjacency matrix $\mathbf{A}^k$ to generate the soft cluster assignment matrix $\mathbf{S}^k$. Finally, DiffPool feeds the top-layer graph representation into a downstream task classifier for the supervision of cluster assignment matrices.

gPool As shown in Fig. 6.10, gPool [32] presents both graph pooling (gPool) and unpooling (gUnpool) operations, based on which the graph data is modeled by an encoder-decoder architecture. The encoder/decoder includes the same number of encoder/decoder blocks, and each encoder/decoder block will contain a GCN layer and a gPool/gUnpool operator. The representations after the last decoder block are used as final representations for downstream tasks.

The gPool operation learns projection scores for each node with a learnable projection vector, and selects nodes with the highest scores as important ones to feed to the next layer:

$$\begin{aligned} \mathbf{s}^k &= \mathbf{X}^k \mathbf{p}^k / ||\mathbf{p}^k||, \\ idx^k &= \text{rank}(\mathbf{s}^k, n^k), \end{aligned} \tag{6.46}$$

where $\mathbf{s}^k$ is the importance score; $\mathbf{p}^k$ and $\mathbf{X}^k$ are the projection vector and input feature matrix in the k-th layer, respectively; and $\text{rank}(\mathbf{s}^k, n^k)$ returns the indices of the elements with top-n^k scores in $\mathbf{s}^k$.

Then in each layer, we define the adjacency matrix and input feature matrix based on the corresponding rows or columns indexed by idx^k:

$$\begin{aligned}\mathbf{A}^{k+1} &= \mathbf{A}^k(idx^k, idx^k),\\ \hat{\mathbf{s}}^k &= \text{Sigmoid}(\mathbf{s}^k(idx^k)),\\ \hat{\mathbf{X}}^k &= \mathbf{X}^k(idx^k, :),\\ \mathbf{X}^{k+1} &= \hat{\mathbf{X}}^k \odot (\hat{\mathbf{s}}^k \mathbf{1}_d^\top),\end{aligned} \tag{6.47}$$

where $\mathbf{1}_d$ is a d-dimensional all-one vector and $\odot$ is the element-wise matrix multiplication. Here the normalized scores $\hat{\mathbf{s}}^k$ are used as weighted masks to further filter the feature matrix $\hat{\mathbf{X}}^k$.

The gUnpool performs the inverse operation of the gPool operation, which restores the graph to its original structure. Specifically, gUnpool records the indices of selected nodes in the corresponding pooling level and then simply places nodes back in their original positions:

$$\mathbf{X}^{k-1} = \text{distribute}(\mathbf{0}_{n^{k-1} \times d}, \mathbf{X}^k, idx^k), \tag{6.48}$$

where idx^k is the indices of n^k selected nodes in the corresponding gPool level, and the function places row vectors in $\mathbf{X}^k$ into $n^{k-1} \times d$ all-zero feature matrix by index idx^k.

Compared to DiffPool, gPool can reduce the storage complexity by replacing the cluster assignment matrix with a projection vector at each layer.

In this section, we introduce how to obtain global graph representation based on local node representations with graph pooling operations, which are widely used in a series of graph-level tasks such as graph classification and interaction prediction. When applying GNNs in modeling text, graph pooling can effectively help us extract sentence-level [72, 152] or document-level [23, 77] information for downstream tasks.

6.6 Self-Supervised Graph Representation Learning

Recently, self-supervised learning methods, which have made immense success in CV and NLP areas, can learn effective representations with well-designed pre-training tasks instead of expensive downstream task labels. Specifically, self-supervised graph representation learning [60] first learns node and graph representations by different graph-based pre-training tasks without human supervision, such as graph structure reconstruction or pseudo-label prediction. Then, the learned models

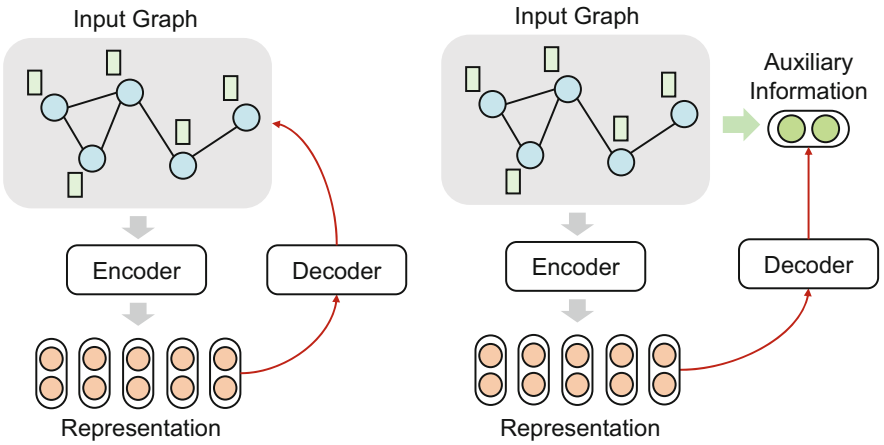


Fig. 6.11 Illustration of generative (left) and predictive (right) methods

or representations can be used in downstream tasks (e.g., graph/node classification). As shown in Figs. 6.11 and 6.12, we will introduce three types of self-supervised graph representation learning methods, namely, generative, predictive, and contrastive, distinguished by different pre-training tasks.

Generative Methods The generative methods aim to reconstruct and predict some components (e.g., graphs, edges, node features) of input data.

Structure Reconstruction These works aim to recover the adjacency matrix or masked edges of a graph. Graph autoencoder (GAE) [51] learns node representations by a two-layer graph convolutional network and then reconstructs the adjacency matrix of the input graph, and variational graph autoencoder (VGAE) [51] is a latent variable variant of GAE. ARGA and ARVGA [75] are adversarial variants of GAE and VGAE, respectively, combining autoencoder and adversarial approaches. AGE [21] adaptively defines the reconstruction objective of adjacency matrices in an iterative manner.

Feature Reconstruction These works aim to recover the node attributes of a graph, i.e., the input features of GNNs. MGAE [108] takes both corrupted network node content and structures as input, and predicts the origin node features. GALA [76] proposes a symmetric graph convolutional autoencoder based on Laplacian sharpening and smoothing to learn node representations by predicting input features. Here the Laplacian sharpening encourages the reconstructed feature of a node to be far away from those of its neighbors. GPT-GNN [44] uses both graph and feature generation pre-training tasks to model the structural and semantic information of the graph.

Predictive Methods Predictive methods learn informative representations with self-supervised signals from some auxiliary information, such as pseudo labels and

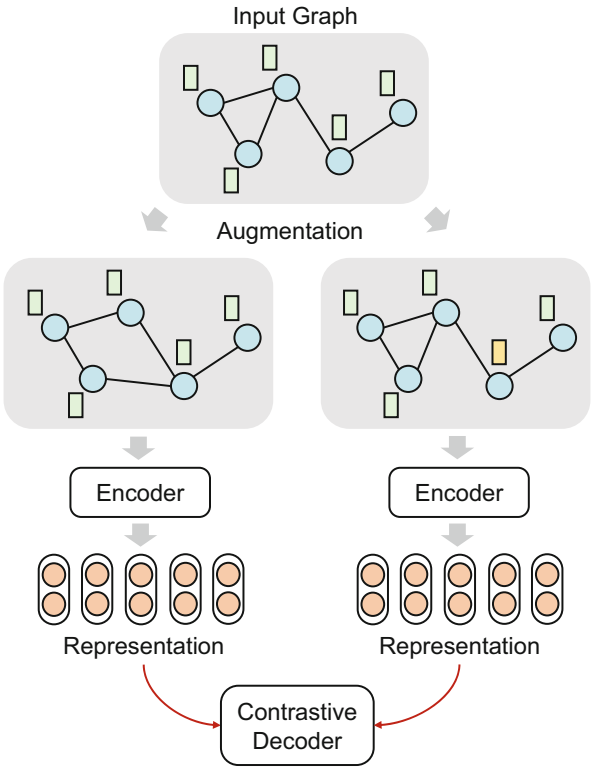


Fig. 6.12 Illustration of contrastive methods

graph properties. Depending on the prediction targets, the predictive methods can be further divided into three types.

Property Prediction These methods manually define some high-level information of the graph for prediction. GROVER [87] uses a graph Transformer as the encoder and employs both contextual property prediction and graph-level motif prediction to encode domain knowledge. S^2GRL [79] takes the k-hop contextual prediction as a pre-training task and trains a well-designed GNN to learn representations. SELAR [46] predicts the meta-paths, which are composite relations of multiple edge types in heterogeneous graphs.

Pseudo-Label Prediction This line of works employs an iterative framework to learn representations and update cluster labels. M3S [98] employs DeepCluster [13] algorithm to generate pseudo cluster labels, and designs an aligning mechanism and a multistage training framework to predict refined pseudo labels. IFC-GCN [43] proposes an EM-like framework to alternately rectify pseudo labels of feature clustering, and update node features by predicting pseudo labels.

Invariant Information Preserving These works aim to preserve some intrinsic and invariant information in the graph. Compared with the property prediction-based methods, there is usually no explicit meaning or closed-form formula for the *invariant information*. CCA-SSG [145] employs the canonical-correlation analysis (CCA) method to maximize the correlation between two augmented views of the same input, and thus preserve augmentation-invariant information. Lagraph [118] assumes that there exists a latent graph without noises behind each observed graph, and the observed graph is randomly generated from the latent one. Lagraph implicitly predicts the latent graph as a pre-training task to learn informative graph and node representations.

Contrastive Methods Contrastive methods first generate two contrastive views from graphs/nodes, and then maximize the mutual information between them. In this way, the learned representations will be more robust to perturbations. Typically, contrastive methods regard the views from the same graph/node as a positive pair, and the views randomly selected from different graphs/nodes as negative pairs. The representation similarity between a positive pair is forced to be larger than negative ones. Categorized by the views to contrast, contrastive methods can be divided into two groups.

Substructure-Based Methods This line of works usually contrasts views between different scales of structures. InfoGraph [95] takes different substructures of the original graph (e.g., nodes, edges, triangles) as contrastive views to generate graph-level representations. DGI [107] and GMI [80] build contrast views between a graph and its nodes. MVGRL [39] generates views by sampling subgraphs, and learns both node- and graph-level representations. GCC [82] treats different subgraphs as contrastive views and introduces InfoNCE loss [73] to large-scale graph pre-training.

Augmentation-Based Methods These works usually generate views by applying different perturbations on input graphs and features. GRACE [158] and GraphCL [142] randomly perturb the input graph (e.g., node and edge dropping) to generate contrastive views. GCA [159] adaptively builds contrastive views based on different graph properties (e.g., node degree, eigenvector, PageRank). Here nodes or edges with small importance scores (e.g., node degrees) are more likely to be dropped in contrastive views. JOAO [141] and AD-GCL [100] automatically select graph augmentations and generate contrastive views in adversarial ways. Instead of contrasting augmented data, DSGC [132] conducts contrastive views from dual spaces of hyperbolic and Euclidean. GASSL [134] generates views by directly perturbing input features and hidden layers. SimGRACE [117] adds Gaussian noises to model parameters as perturbations to generate contrastive views. MA-GCL [34] proposes a novel augmentation strategy that randomly perturbs the neural architecture of GNN encoders (i.e., random permutations of graph filter, linear projection, and nonlinear function) as contrastive views. HeCo [112] employs network schema and meta-path views in heterogeneous graphs as two specific views.

Adaptation Approaches After the self-supervised training process, there are roughly three paradigms to adapt the learned models or representations for downstream tasks.

Pre-training-Fine-Tuning These methods [44, 87, 142] first train the model parameters of graph encoder on the datasets without labels in the self-supervised way, and then the pre-trained parameters are used as the initial parameters in the next fine-tuning step, which updates the encoder in a supervised way by downstream tasks.

Unsupervised Representation Learning These methods [51, 75, 79, 158] train the graph encoder with pre-training tasks in the first stage, and the pre-trained encoder is taken as a feature extractor with frozen parameters to generate representations for downstream tasks in the second stage.

Multitask Training These methods [46, 82, 98] train graph encoders on both pre-training and downstream tasks with well-designed loss functions, which can be seen as a type of multitask learning, where the pre-training tasks are auxiliary tasks for downstream ones.

In summary, self-supervised graph representation learning methods can learn effective graph and node representations based on various pre-training tasks without labels. Different from pre-trained language models where pre-training-fine-tuning are the mainstream for adaptation of downstream tasks, the unsupervised representation learning paradigm is widely used in graph data, where only the node/graph representations in the last feed-forward layer are fed into downstream tasks as features. An intuitive reason is that popular tasks on graph data (e.g., node/graph classification) require less model capacity than those on NLP (e.g., machine translation). Therefore, the final representations in graph encoders are usually sufficient, and it's not necessary to fine-tune the learned graph encoders.

6.7 Applications

In this section, we will introduce several typical applications of graph representation learning in the NLP area.

Text Classification Text classification is an essential task in NLP. Typical GNN models, including GCNs [2, 23, 37, 41, 52, 69] and GATs [106], are applied in text classification to model the structural information (e.g., citation relationship) between documents. However, these works did not fully model the structural information lying in texts. Thus, some works manage to construct graphs from texts. Peng et al. [77] propose to transform texts into a graph of words, and then apply graph convolutional operations to it. Yao et al. [135] propose to construct a text graph with document and word nodes, and utilize GCNs to learn representations for both word nodes and document nodes. Besides constructing the text graph with human heuristics as in the above works, there are also some intrinsic graph structures that can be used, such as dependency parsing trees, semantic parsing graphs, etc.

One of the most typical works of utilizing these intrinsic graph structures is the Tree-LSTM [101] introduced in this chapter.

Sequence Labeling Sequence labeling is another classical task in NLP, which aims to assign labels for each word in the text sequence. Sentence LSTM [151] introduces graph recurrent networks for sentence modeling, where each word node connects with its neighboring words in a context window and a global sentence node links with all word nodes. Then, the hidden representations of word nodes can be used for predicting word labels. Sentence LSTM achieves promising performance in the POS tagging and NER tasks. To solve the semantic role labeling task, Marcheggiani et al. [65] propose a variant of GCN [52] which performs reasoning on syntactic dependency trees (i.e., a graph with labeled edges), and employs an edgewise gate mechanism to consider the information of each dependency edge. They further show that GCNs and LSTMs are functionally complementary in this task.

Knowledge Acquisition As a crucial subtask for knowledge acquisition, relation extraction aims to predict the relationship between entities in plain text. While CNNs and RNNs have achieved promising results on multiple benchmarks, researchers find the syntactic and semantic information of the text, such as adjacency, syntactic dependencies, and discourse relations, are also helpful for relation extraction. To this end, Zhang et al. [152] propose a GNN-based method for relation extraction, which can efficiently aggregate information from arbitrary dependency graphs. It also applies a novel pruning strategy to the input tree and only keeps the informative words for relation extraction. To model the rich relations across entities, Zhu et al. [157] propose to construct entity graphs and generate edge parameters to propagate diverse relational information, which greatly extends edge types and enables GNNs to conduct complex reasoning. Considering cross-sentence dependencies like coreference and discourse, Peng et al. [78] further explore extracting N-ary relations among multiple entities from multiple sentences by applying Graph LSTMs on document graphs.

Event extraction is another knowledge acquisition task, which aims to identify event triggers and the corresponding arguments for each trigger in texts. Nguyen et al. [72] propose syntactic GCN, which models the dependency tree and learns representations of word nodes to extract events from texts. In addition, Liu et al. [59] find that modeling syntactic relations help capture long-range dependencies better, and these shortcut arcs can help extract multiple events jointly. Meanwhile, to deal with ambiguous and unseen triggers, Zhang et al. [149] summarize event structure knowledge from training data, and construct event background graphs for each event. These graphs help identify correct events by matching the structure knowledge.

Fact Verification Fact verification aims to retrieve evidence from plain text and verify given claims with the evidence. In other words, we need to label a given claim as SUPPORTED, REFUTED, or NOT ENOUGH INFO, which indicates that the evidence can support, refute, or is not sufficient for validating the claim. By regarding the problem as a natural language inference (NLI) [1] task, traditional

methods simply combine the evidence via concatenation, or build models based on evidence-claim pairs. To integrate multiple evidence to reason facts, Zhou et al. [156] propose a graph-based evidence aggregating and reasoning framework, which propagates and aggregates information on a fully connected evidence graph. In this way, different pieces of evidence can have sufficient interactions with each other, and thus can help the situations in which multiple pieces of evidence are necessary for making the decision. Liu et al. [61] further incorporate kernel-based attention mechanism into GAT for fine-grained evidence aggregation, and the proposed KGAT achieves improved performance.

Machine Translation Traditionally, machine translation (MT) is modeled as a sequence-to-sequence (seq2seq) problem. The graph structure, however, enables MT models to incorporate explicit linguistic biases. To this end, two typical kinds of graphs can be utilized in MT. Some works [4, 5, 36] model syntactic trees with GCNs to learn syntax-aware sentence representations, and show improvements over vanilla seq2seq methods. To involve more semantic information, other works further consider semantic role labeling (SRL) as well as abstract meaning representation (AMR) graphs [64, 94]. Besides traditional MT, GNNs are also utilized in multimodal and document MT. For multimodal MT, Yin et al. [137] build combined graphs with both entities in images and sentences and then apply GNNs for message passing across modalities. For document MT, Xu et al. [121] model long-term dependencies (e.g., coreference) with GNNs on document graphs, and achieve superior performance on translation coherence.

Question Answering Question answering (QA) aims to generate or find answers for a question based on relevant documents or knowledge bases, which requires models to reason and infer the right answers given the question (see Chap. 4 for an introduction). Since GNNs are designed to model relational data, they are also suitable for QA. To apply GNNs to QA, we need to first build graphs containing question-related entities and their relations. Typically, according to the provided context, researchers could utilize existing knowledge graphs [96, 97, 136] or extract entities and links from documents to construct graphs [11, 24, 29, 30, 58]. Afterward, multiple kinds of GNNs such as GCN, GAT, and R-GCN [90] can be directly used to reason over the graphs [24, 58]. To jointly capture relational and semantic messages, some works explore how to combine powerful PTMs and GNNs for complex reasoning. Ding et al. [24] construct entity graphs utilizing BERT, which predicts node entities and relational edges iteratively. Yasunaga et al. [136] further incorporate PTM-encoded context representations into knowledge graphs to better perform message passing.

Besides the applications in the NLP area, GNNs are widely used in various application scenarios, such as community detection [104, 133, 148], information diffusion prediction [130], recommender systems [8, 28, 115, 129, 139], molecular fingerprints [50], chemical reaction prediction [25], protein interface prediction [31], biomedical engineering [86, 161], etc. Since our book mainly focuses on NLP, readers who are interested in these applications can refer to these papers for more details.

6.8 Summary and Further Readings

In this chapter, we have introduced graph representation learning, which projects graph structure information into continuous vector space and makes deep learning techniques possible on graph data. We first talk about shallow node representation, from spectral clustering, shallow neural networks, to matrix factorization. Then we introduce deep node representation, from autoencoder-based methods to graph neural networks. Afterward, we present how to obtain the global graph representation from node representations. Finally, we introduce how graph representation learning helps a series of NLP tasks.

For further readings of graph representation learning, there are also some recommended reviews and books. In terms of general graph embedding, Goyal and Ferrara [35] and Cui et al. [22] conduct surveys on relevant models and their applications. We also write a monograph [126] about our systematic work on the topic of network embedding. In terms of GNNs, Wu et al. [114] write a book covering more than 20 topics of GNNs in 4 parts: introduction, foundations, frontiers, and applications. Shi et al. [93] write a monograph providing a launch point for discussing the latest trends of GNNs. Wu et al. [113] present a survey about the NLP applications based on GNNs. Shi et al. [92] focus on the representation learning of heterogeneous graphs. We also provide a more comprehensive survey of GNNs in our review [155], covering a broader range of aspects, such as the applications on images or chemistry.

Acknowledgments The contributions of all authors for the second edition are Zhiyuan Liu, Yankai Lin, and Maosong Sun designed the overall architecture of this chapter. Cheng Yang and Yankai Lin drafted this chapter. Zhiyuan Liu and Yankai Lin proofread and revised this chapter.

We thank Ganqu Cui and Chaojun Xiao for drawing figures, and thank Yujia Qin, Ning Ding, Ganqu Cui, Yuqi Luo, and Ziqing Qiao for proofreading the chapter. We also thank Jie Zhou and Zhengyan Zhang for preparing some initial draft materials for the first edition.

This is the graph representation learning chapter of the second edition of the book *Representation Learning for Natural Language Processing*, with its first edition published in 2020 [62]. As compared to the first edition of this chapter, the main changes include the following: (1) we reorganized the narrative logic of this chapter, by dividing it into shallow node representation, deep node representation, graph representation, self-supervised graph learning, and applications; (2) we rewrote and updated the part of graph neural networks; and (3) we added the contents of graph Transformer and self-supervised graph representation learning.

References

1. Gabor Angeli and Christopher D Manning. Naturalli: Natural logic inference for common sense reasoning. In *Proceedings of EMNLP*, 2014.
2. James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Proceedings of NeurIPS*, 2016.
3. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*, 2015.

4. Jasmijn Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of EMNLP*, 2017.
5. Daniel Beck, Gholamreza Haffari, and Trevor Cohn. Graph-to-sequence learning using gated graph neural networks. In *Proceedings of ACL*, 2018.
6. Slobodan Beliga, Ana Meštrović, and Sanda Martinčić-Ipšić. An overview of graph-based keyword extraction methods and approaches. *Journal of information and organizational sciences*, 39(1):1–20, 2015.
7. Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of NeurIPS*, 2001.
8. Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.
9. Ronald Newbold Bracewell and Ronald N Bracewell. *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York, 1986.
10. Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *Proceedings of ICLR*, 2014.
11. Nicola De Cao, Wilker Aziz, and Ivan Titov. Question answering by reasoning across documents with graph convolutional networks. In *Proceedings of NAACL*, 2019.
12. Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of CIKM*, 2015.
13. Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of ECCV*, 2018.
14. Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of AAAI*, 2020.
15. Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *Proceedings of ICML*, 2022.
16. Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *Proceedings of ICLR*, 2018.
17. Mo Chen, Qiong Yang, and Xiaoou Tang. Directed graph embedding. In *Proceedings of IJCAI*, 2007.
18. Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. In *Proceedings of EMNLP*, 2016.
19. Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of EMNLP*, 2014.
20. Wojciech Chojnacki and Michael J Brooks. A note on the locally linear embedding algorithm. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(08):1739–1752, 2009.
21. Ganqu Cui, Jie Zhou, Cheng Yang, and Zhiyuan Liu. Adaptive graph encoder for attributed graph embedding. In *Proceedings of SIGKDD*, 2020.
22. Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 2018.
23. Michael Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of NeurIPS*, 2016.
24. Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. Cognitive graph for multi-hop reading comprehension at scale. In *Proceedings of ACL*, 2019.
25. Kien Do, Truyen Tran, and Svetha Venkatesh. Graph transformation policy network for chemical reaction prediction. In *Proceedings of KDD*, 2019.

26. Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of ICLR*, 2021.
27. David K Duvenaud, Dougal Maclaurin, Jorge Aguileraiparraguirre, Rafael Gomezbombarelli, Timothy D Hirzel, Alan Aspuruguzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of NeurIPS*, 2015.
28. Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *Proceedings of WWW*, 2019.
29. Yuwei Fang, Siqi Sun, Zhe Gan, Rohit Pillai, Shuohang Wang, and Jingjing Liu. Hierarchical graph network for multi-hop question answering. In *Proceedings of EMNLP*, 2020.
30. Yanlin Feng, Xinyue Chen, Bill Yuchen Lin, Peifeng Wang, Jun Yan, and Xiang Ren. Scalable multi-hop relational reasoning for knowledge-aware question answering. In *Proceedings of EMNLP*, 2020.
31. Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Proceedings of NeurIPS*, 2017.
32. Hongyang Gao and Shuiwang Ji. Graph u-nets. In *Proceedings of ICML*. PMLR, 2019.
33. Jonas Gehring, Michael Auli, David Grangier, and Yann N Dauphin. A convolutional encoder model for neural machine translation. In *Proceedings of ACL*, 2017.
34. Xumeng Gong, Cheng Yang, and Chuan Shi. Ma-gcl: Model augmentation tricks for graph contrastive learning. In *Proceedings of AAAI*, 2023.
35. Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.
36. Zhijiang Guo, Yan Zhang, Zhiyang Teng, and Wei Lu. Densely connected graph convolutional networks for graph-to-sequence learning. *Transactions of the Association for Computational Linguistics*, 2019.
37. Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of NeurIPS*, 2017.
38. William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data(base) Engineering Bulletin*, 40:52–74, 2017.
39. Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *Proceedings of ICML*, 2020.
40. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of CVPR*, 2016.
41. Mikael Henaff, Joan Bruna, and Yann Lecun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.
42. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.
43. Zhihui Hu, Guang Kou, Haoyu Zhang, Na Li, Ke Yang, and Lin Liu. Rectifying pseudo labels: Iterative feature clustering for graph representation learning. In *Proceedings of CIKM*, 2021.
44. Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. GPT-GNN: Generative pre-training of graph neural networks. In *Proceedings of SIGKDD*, 2020.
45. Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *Proceedings of NeurIPS*, 2018.
46. Dasol Hwang, Jinyoung Park, Sunyoung Kwon, KyungMin Kim, Jung-Woo Ha, and Hyunwoo J Kim. Self-supervised auxiliary learning with meta-paths for heterogeneous graphs. *Proceedings of NeurIPS*, 2020.
47. Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-RNN: Deep learning on spatio-temporal graphs. In *Proceedings of CVPR*, 2016.
48. AJAY KUMAR JAISWAL, Peihao Wang, Tianlong Chen, Justin F Rousseau, Ying Ding, and Zhangyang Wang. Old can be gold: Better gradient flow can make vanilla-gcns great again. In *Advances in Neural Information Processing Systems*, 2022.

49. Michael Kampffmeyer, Yinbo Chen, Xiaodan Liang, Hao Wang, Yujia Zhang, and Eric P Xing. Rethinking knowledge graph propagation for zero-shot learning. In *Proceedings of CVPR*, 2019.
50. Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-aided Molecular Design*, 30(8):595–608, 2016.
51. Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.
52. Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of ICLR*, 2017.
53. Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Proceedings of NeurIPS*, 2021.
54. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.
55. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998.
56. Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *Proceedings of ICLR*, 2018.
57. Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S Zemel. Gated graph sequence neural networks. In *Proceedings of ICLR*, 2016.
58. Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. Kagnet: Knowledge-aware graph networks for commonsense reasoning. In *Proceedings of EMNLP*, 2019.
59. Xiao Liu, Zhunchen Luo, and He-Yan Huang. Jointly multiple events extraction via attention-based graph information aggregation. In *Proceedings of EMNLP*, 2018.
60. Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip Yu. Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2022.
61. Zhenghao Liu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. Fine-grained fact verification with kernel graph attention network. In *Proceedings of ACL*, 2020.
62. Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation Learning for Natural Language Processing*. Springer, 2020.
63. Stéphane Mallat. *A wavelet tour of signal processing*. Elsevier, 1999.
64. Diego Marcheggiani, Jasmijn Bastings, and Ivan Titov. Exploiting semantics in neural machine translation with graph convolutional networks. In *Proceedings of NAACL-HLT*, 2018.
65. Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of EMNLP*, 2017.
66. Rada Mihalcea. Graph-based ranking algorithms for sentence extraction, applied to text summarization. In *Proceedings of ACL*, 2004.
67. Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of EMNLP*, 2004.
68. T Mikolov and J Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of NeurIPS*, 2013.
69. Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of CVPR*, 2017.
70. Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3):036104, 2006.
71. Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.
72. Thien Nguyen and Ralph Grishman. Graph convolutional networks with argument-aware pooling for event detection. In *Proceedings of AAAI*, 2018.
73. Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

74. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.
75. Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. *Proceedings of IJCAI*, 2018.
76. Jiwoong Park, Minsik Lee, Hyung Jin Chang, Kyuewang Lee, and Jin Young Choi. Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In *Proceedings of ICCV*, 2019.
77. Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Mengjiao Bao, Lihong Wang, Yangqiu Song, and Qiang Yang. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *Proceedings of WWW*, 2018.
78. Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. Cross-sentence n-ary relation extraction with graph LSTMs. *Transactions of the Association for Computational Linguistics*, 5:101–115, 2017.
79. Zhen Peng, Yixiang Dong, Minnan Luo, Xiao-Ming Wu, and Qinghua Zheng. Self-supervised graph representation learning via global context prediction. *arXiv preprint arXiv:2003.01604*, 2020.
80. Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. Graph representation learning via graphical mutual information maximization. In *Proceedings of WWW*, 2020.
81. Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In *Proceedings of KDD*, 2014.
82. Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of SIGKDD*, 2020.
83. Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of WSDM*, 2018.
84. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67, 2020.
85. Afshin Rahimi, Trevor Cohn, and Timothy Baldwin. Semi-supervised user geolocation via graph convolutional networks. In *Proceedings of ACL*, 2018.
86. Sungmin Rhee, Seokjun Seo, and Sun Kim. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. In *Proceedings of IJCAI*, 2018.
87. Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. In *Proceedings of NeurIPS*, 2020.
88. Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.
89. Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE TNN 2009*, 20(1):61–80, 2009.
90. Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *Proceedings of ESWC*, 2018.
91. Chuan Shi, Binbin Hu, Wayne Xin Zhao, and S Yu Philip. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 31(2):357–370, 2018.
92. Chuan Shi, Xiao Wang, and S Yu Philip. Heterogeneous graph representation learning and applications, 2022.
93. Chuan Shi, Xiao Wang, and Cheng Yang. *Advances in Graph Neural Networks*. Springer, 2022.

94. Linfeng Song, Daniel Gildea, Yue Zhang, Zhiguo Wang, and Jinsong Su. Semantic neural machine translation using amr. *Transactions of the Association for Computational Linguistics*, 2019.
95. Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *Proceedings of ICLR*, 2019.
96. Haitian Sun, Tania Bedrax-Weiss, and William W Cohen. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Proceedings of EMNLP*, 2019.
97. Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William W Cohen. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of EMNLP*, 2018.
98. Ke Sun, Zhouchen Lin, and Zhanxing Zhu. Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes. In *Proceedings of AAAI*, 2020.
99. Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003, 2011.
100. Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. Adversarial graph augmentation to improve graph contrastive learning. *Proceedings of NeurIPS*, 2021.
101. Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of ACL*, 2015.
102. Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale information network embedding. In *Proceedings of WWW*, 2015.
103. Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proceedings of KDD*, 2009.
104. Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Muller. Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*, 2020.
105. Ashish Vaswani, Noam Shazeer, Niki Parmar, Llion Jones, Jakob Uszkoreit, Aidan N Gomez, and Lukasz Kaiser. Attention is all you need. In *Proceedings of NeurIPS*, 2017.
106. Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *Proceedings of ICLR*, 2018.
107. Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *Proceedings of ICLR*, 2018.
108. Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. Mgae: Marginalized graph autoencoder for graph clustering. In *Proceedings of CIKM*, 2017.
109. Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of KDD*, 2016.
110. Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and S Yu Philip. A survey on heterogeneous graph embedding: methods, techniques, applications and sources. *IEEE Transactions on Big Data*, 2022.
111. Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *Proceedings of WWW*, 2019.
112. Xiao Wang, Nian Liu, Hui Han, and Chuan Shi. Self-supervised heterogeneous graph neural network with co-contrastive learning. In *Proceedings of SIGKDD*, 2021.
113. Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, and Bo Long. Graph neural networks for natural language processing: A survey. *arXiv preprint arXiv:2106.06090*, 2021.
114. Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Singapore, Singapore, 2022.
115. Qitian Wu, Hengrui Zhang, Xiaofeng Gao, Peng He, Paul Weng, Han Gao, and Guihai Chen. Dual graph attention networks for deep latent representation of multifaceted social effects in recommender systems. In *Proceedings of WWW*, 2019.

116. Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. *Proceedings of NeurIPS*, 2021.
117. Jun Xia, Lirong Wu, Jintao Chen, Bozhen Hu, and Stan Z Li. Simgrace: A simple framework for graph contrastive learning without data augmentation. In *Proceedings of WWW*, 2022.
118. Yaochen Xie, Zhao Xu, and Shuiwang Ji. Self-supervised representation learning via latent graph prediction. *Proceedings of ICML*, 2022.
119. Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *Proceedings of ICLR*, 2019.
120. Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Kenichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of ICML*, 2018.
121. Mingzhou Xu, Liangyou Li, Derek Wong, Qun Liu, Lidia S Chao, et al. Document graph for neural machine translation. In *Proceedings of EMNLP*, 2020.
122. Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Proceedings of AAAI*, 2018.
123. Cheng Yang, Yuxin Guo, Yao Xu, Chuan Shi, Jiawei Liu, Chunchen Wang, Xin Li, Ning Guo, and Hongzhi Yin. Learning to distill graph neural networks. In *Proceedings of WSDM*, 2023.
124. Cheng Yang, Jiawei Liu, and Chuan Shi. Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework. In *Proceedings of WWW*, 2021.
125. Cheng Yang and Zhiyuan Liu. Comprehend deepwalk as matrix factorization. *arXiv preprint arXiv:1501.00358*, 2015.
126. Cheng Yang, Zhiyuan Liu, Cunchao Tu, Chuan Shi, and Maosong Sun. Network embedding: Theories, methods, and applications. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 15(2):1–242, 2021.
127. Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. Network representation learning with rich text information. In *Proceedings of IJCAI*, 2015.
128. Cheng Yang, Maosong Sun, Zhiyuan Liu, and Cunchao Tu. Fast network embedding enhancement via high order proximity approximation. In *Proceedings of IJCAI*, 2017.
129. Cheng Yang, Maosong Sun, Wayne Xin Zhao, Zhiyuan Liu, and Edward Y Chang. A neural network approach to jointly modeling social networks and mobile trajectories. *ACM Transactions on Information Systems*, 35(4):1–28, 2017.
130. Cheng Yang, Jian Tang, Maosong Sun, Ganqu Cui, and Liu Zhiyuan. Multi-scale information diffusion prediction with reinforced recurrent networks. In *Proceedings of IJCAI*, 2019.
131. Cheng Yang, Chunchen Wang, Yuanfu Lu, Xumeng Gong, Chuan Shi, Wei Wang, and Xu Zhang. Few-shot link prediction in dynamic networks. In *Proceedings of WSDM*, 2022.
132. Haoran Yang, Hongxu Chen, Shirui Pan, Lin Li, Philip S Yu, and Guandong Xu. Dual space graph contrastive learning. *Proceedings of WWW*, 2022.
133. Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of WSDM*, 2013.
134. Longqi Yang, Liangliang Zhang, and Wenjing Yang. Graph adversarial self-supervised learning. *Proceedings of NeurIPS*, 2021.
135. Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of AAAI*, 2019.
136. Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. QA-GNN: Reasoning with language models and knowledge graphs for question answering. In *Proceedings of NAACL*, 2021.
137. Yongjing Yin, Fandong Meng, Jinsong Su, Chulun Zhou, Zhengyuan Yang, Jie Zhou, and Jiebo Luo. A novel graph-based multi-modal fusion encoder for neural machine translation. In *Proceedings of ACL*, 2020.
138. Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Proceedings of NeurIPS*, 2021.

139. Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of KDD*, 2018.
140. Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of NeurIPS*, 2018.
141. Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. In *Proceedings of ICML*, 2021.
142. Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *Proceedings of NeurIPS*, 2020.
143. Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of ICLR*, 2018.
144. Victoria Zayats and Mari Ostendorf. Conversation modeling on reddit using a graph-structured LSTM. *Transactions of the Association for Computational Linguistics*, 6:121–132, 2018.
145. Hengrui Zhang, Qitian Wu, Junchi Yan, David Wipf, and Philip S Yu. From canonical correlation analysis to self-supervised graph neural networks. In *Proceedings of NeurIPS*, 2021.
146. Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit Yan Yeung. GaAN: Gated attention networks for learning on large and spatiotemporal graphs. In *Proceedings of UAI*, 2018.
147. Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of AAAI*, 2018.
148. Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. Attributed graph clustering via adaptive graph convolution. In *Proceedings of IJCAI*, 2019.
149. Yilin Zhang, Ziran Li, Zhiyuan Liu, Hai-Tao Zheng, Ying Shen, and Lan Zhou. Event detection with dynamic word-trigger-argument graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 2021.
150. Yizhou Zhang, Yun Xiong, Xiangnan Kong, Shanshan Li, Jinhong Mi, and Yangyong Zhu. Deep collective classification in heterogeneous information networks. In *Proceedings of WWW*, 2018.
151. Yue Zhang, Qi Liu, and Linfeng Song. Sentence-state LSTM for text representation. In *Proceedings of ACL*, 2018.
152. Yuhao Zhang, Peng Qi, and Christopher D Manning. Graph convolution over pruned dependency trees improves relation extraction. In *Proceedings of EMNLP*, 2018.
153. Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Maosong Sun, Zhichong Fang, Bo Zhang, and Leyu Lin. Cosine: compressive network embedding on large-scale information networks. *IEEE Transactions on Knowledge and Data Engineering*, 2020.
154. Jianan Zhao, Xiao Wang, Chuan Shi, Zekuan Liu, and Yanfang Ye. Network schema preserving heterogeneous information network embedding. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2020.
155. Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.
156. Jie Zhou, Xu Han, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. GEAR: Graph-based evidence aggregating and reasoning for fact verification. In *Proceedings of ACL 2019*, 2019.
157. Hao Zhu, Yankai Lin, Zhiyuan Liu, Jie Fu, Tat-Seng Chua, and Maosong Sun. Graph neural networks with generated parameters for relation extraction. In *Proceedings of ACL*, 2019.
158. Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*, 2020.
159. Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of WWW*, 2021.

160. Julian G Zilly, Rupesh Kumar Srivastava, Jan Koutnik, and Jurgen Schmidhuber. Recurrent highway networks. In *Proceedings of ICML*, 2016.
161. Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Intelligent Systems in Molecular Biology*, 34(13):258814, 2018.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Chapter 7
Cross-Modal Representation Learning

Yuan Yao, Zhiyuan Liu, Yankai Lin, and Maosong Sun

Abstract Cross-modal representation learning is an essential part of representation learning, which aims to learn semantic representations for different modalities including text, audio, image and video, etc., and their connections. In this chapter, we introduce the development of cross-modal representation learning from shallow to deep, and from respective to unified in terms of model architectures and learning mechanisms for different modalities and tasks. After that, we review how cross-modal capabilities can contribute to complex real-world applications.

7.1 Introduction

Modalities are means of information exchange between human beings and the real world. Concretely, each modality is an independent channel of sensory input or output for intelligent systems. Typical modalities for humans include text, audio, image, and video, while AI systems can process more modalities such as infrared information. Cross-modal representation learning refers to learning paradigms where multiple modalities are involved.

Cross-modal representation learning is an important topic of representation learning. In fact, AI is inherently a cross-modal problem [52], where handling multiple modalities is both necessary and beneficial for real-world intelligent systems. Regarding the necessity, in many real-world applications, intelligent systems are required to operate in a cross-modal environment, such as transcribing speech to text [9], or navigating in a room according to text instructions [10]. From the beneficial perspective, it can be helpful to integrate the correlated and complementary information in different modalities for comprehensive decision-

Y. Yao · Z. Liu (✉) · M. Sun
Department of Computer Science and Technology, Tsinghua University, Beijing, China
e-mail: yuan-yao18@mails.tsinghua.edu.cn; liuzy@tsinghua.edu.cn; sms@tsinghua.edu.cn

Y. Lin
Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China
e-mail: yankailin@ruc.edu.cn

© The Author(s) 2023

Z. Liu et al. (eds.), *Representation Learning for Natural Language Processing*,
https://doi.org/10.1007/978-981-99-1600-9_7

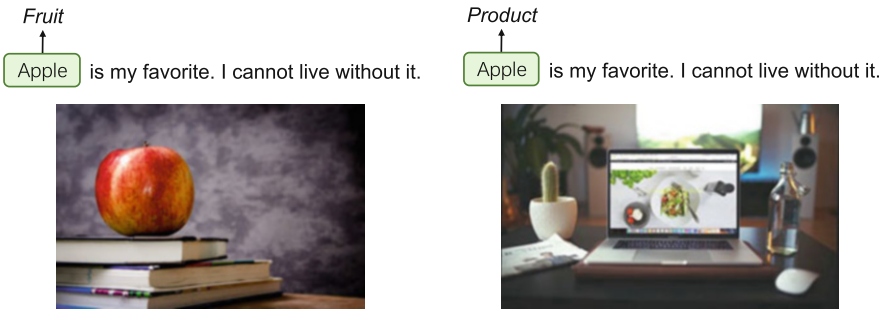


Fig. 7.1 Cross-modal information can be helpful in understanding high-level semantics. The apple fruit image is obtained from pixabay.com, and the apple product image is obtained from commons.wikimedia.org, both from the public domain

making. For example, for human perceptions, the judgment of a syllable is made by not only the sound we hear but also the movement of the lips and tongue of the speaker we see. An experiment in McGurk et al. [68] shows that a voiced /ba/ with a visual /ga/ is perceived by most people as a /da/. Moreover, the high-level semantics can also usually be better identified in a cross-modal context. As shown in Fig. 7.1, cross-modal context is important to resolve the specific semantic meaning of *Apple*. Therefore, it is natural for us to consider the possibility of combining cross-modal information in our AI systems and generating cross-modal representation.

To learn cross-modal representations, models typically need to first understand the heterogenous data from each modality with complex semantic composition, as shown in Fig. 7.2. Various deep neural architectures have been developed to incorporate the inductive bias for the heterogenous data from different modalities. The difference between modalities can be illustrated in two aspects, including the basic units and their modal structures. (1) A fundamental difference between text and other modalities lies in the information density of **basic units** [35]. Text is human-generated abstract signals with high information density, where the basic units (e.g., symbolic words) already carry high-level semantics. In comparison, images and speech are direct recordings of real-world signals, where it is usually more challenging to recognize high-level semantics from basic units with low information density (e.g., recognizing objects from continuous image pixels). (2) **Modal structure** also constitutes a major difference between modalities. For example, text and speech exhibit sequential dependency between basic units, and in comparison, information is spatially presented in images, leading to invariance in shift and scale in images. Single frames in videos are spatially presented, and different frames are organized in a sequential structure. To account for these structures, recurrent neural networks (RNNs) and convolutional neural networks (CNNs) have been developed respectively.

Moreover, models are challenged with establishing **cross-modal mapping** for cross-modal information alignment and fusion. The fine-grained mapping can exist between information from different semantic levels and modalities. Since

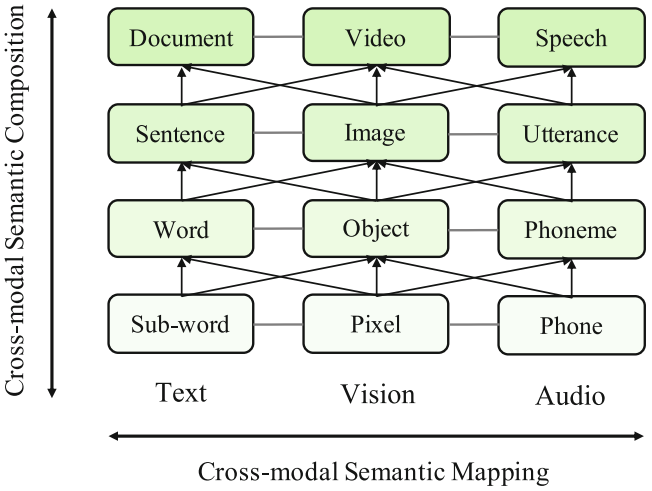


Fig. 7.2 Cross-modal representation learning is challenged with modeling cross-modal semantic composition and establishing cross-modal semantic mapping

explicit annotation of cross-modal mapping is limited, the learning of cross-modal alignment and fusion is typically implicitly driven by supervised learning on specific task annotations. For example, by learning to answer questions about images, models implicitly learn the cross-modal mapping between text tokens and image regions. The model architectures are usually highly specialized for different tasks, and the cross-modal representations are learned by task annotations.

Recently, there is a trend of more unified deep cross-modal representation learning in terms of both model architecture and learning mechanisms. Specifically, Transformers have been proven to be effective in modeling different modalities, including text [90], speech [22], image [23], and video [30]. More unified self-supervised pre-training on large-scale cross-modal data has also pushed forward the state of the arts of many cross-modal tasks [2, 96, 109]. A unified model simultaneously dealing with different modalities and tasks is beginning to take shape, which can be a promising foundation and path to realizing general intelligent systems in the future.

In the following part of this chapter, we will first introduce fundamental cross-modal capabilities for cross-modal tasks in Sect. 7.2. Then, we will review representative cross-modal representation learning models, including shallow representation models in Sect. 7.3, deep representation models in Sect. 7.4, and deep pre-training models in Sect. 7.5. Finally, we will introduce critical applications in Sect. 7.6. In this chapter, without loss of generality, we focus on introducing vision-language models, which are the most important and widely investigated area in cross-modal representation learning research, and also inspire research in other modalities.

7.2 Cross-Modal Capabilities

A real-world cross-modal application usually requires a comprehensive mastery of multiple cross-modal capabilities. In this section, we first provide a taxonomy of cross-modal capabilities and then introduce the corresponding models in the following section. Specifically, cross-modal capabilities can be roughly divided into three categories, including cross-modal understanding, cross-modal retrieval, and cross-modal generation.

Cross-Modal Understanding Models are required to perform semantic understanding based on the given image and query text of the task, for example, answering the question about the image, grounding text into image regions, or identifying semantic relations between objects. Fine-grained cross-modal alignment and fusion between image regions and text tokens are important to achieve strong cross-modal understanding performance.

Cross-Modal Retrieval Given a large candidate set of text and images, and a query from one modality, models are asked to retrieve the corresponding data from other modalities, for example, retrieving images based on a text query or retrieving text based on an image query. Due to the large number of retrieval candidates, cross-modal retrieval methods need to model the holistic semantic relations between data from different modalities in an efficient and scalable way.

Cross-Modal Generation For image-to-text generation, models are required to generate natural language text about the given image content, for example, describing the image content or having conversations on the image. An image-to-text generation model needs to establish fine-grained mapping between text generation and image understanding, and achieve a good trade-off between diversity and fidelity in describing the visual content with text. Another reverse capability is text-to-image generation, which requires models to produce images reflecting the given text description, which can be useful to produce AI-generated content (AIGC). Compared with image-to-text generation, text-to-image generation presents more challenges on the vision side, such as image generation with high-resolution and good computation efficiency. In this chapter, we mainly introduce image-to-text models.

7.3 Shallow Cross-Modal Representation Learning

Early works in cross-modal representation learning have investigated fusing cross-modal information in shallow representations, such as word representations. The word representations can serve as input text representations of deep cross-modal neural networks, and can be efficiently learned through shallow neural architectures on large-scale data. As introduced in Chap. 2, traditional word embedding models like word2vec [69] are trained on a text corpus. These models, while being

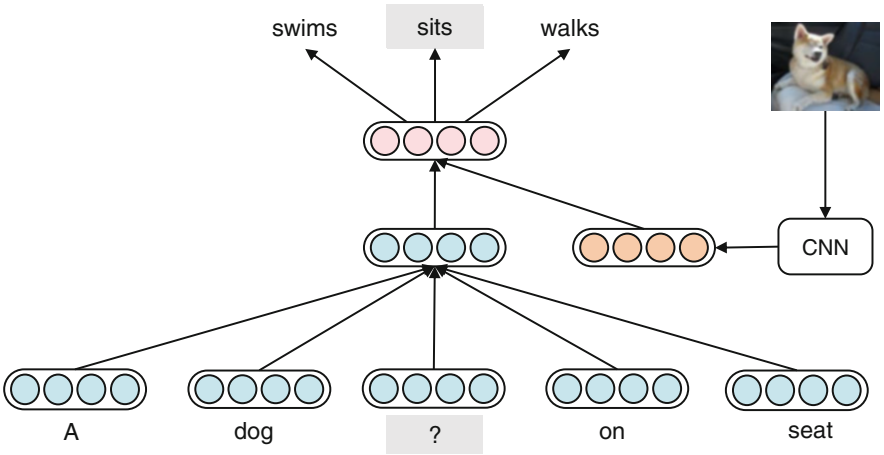


Fig. 7.3 The architecture for word embedding with global visual context. The figure is redrawn according to Fig. 1 in [105], and the image is obtained from Visual Genome [53]

successful, cannot discover implicit semantic relatedness between words that could be revealed in other modalities. Kottur et al. [52] provide an example: even though *eat* and *stare_at* seem unrelated from text, images might show that when people are *eating* something, they would also tend to *stare_at* it. Besides, the semantics of concrete words (e.g., colors and objects) can also be better reflected with the help of visual information [13, 49]. This implies that considering other modalities when constructing word embeddings may help capture more implicit semantic relatedness, where the fused cross-modal representation can facilitate various cross-modal tasks.

Vision, being one of the most critical modalities, has attracted attention from researchers seeking to improve word representations. Several models that incorporate visual information and improve word representations with vision have been proposed. We introduce two typical word representation models, which incorporate visual information as additional context and optimization target as follows.

Word Embedding with Visual Context In most word representation learning models, only local context information from text is considered (e.g., trying to predict a word using neighboring words and phrases). Global information (e.g., the topic of the passage), on the other hand, is often neglected. The image associated with the text can provide such global information for word representation learning. Therefore, some works have proposed to extend word embedding models by using visual information as additional global features (see Fig. 7.3).

Xu et al. [105] make such an attempt in this direction. The input of the model is an image I and a word sequence describing it (i.e., the image caption). Based on a vanilla continuous bag-of-words (CBOW) model, when we consider a certain word w_t in a sequence, its local text feature is the average of embeddings of words in a window, i.e., $\{w_{t-k}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+k}\}$. The visual feature is computed

directly from the image I using a CNN and then used as the global feature. The local feature and the global feature are then concatenated into the aggregated context feature $\mathbf{h}$, based on which the word probability is computed:

$$P(w_t|w_{t-k},\ldots,w_{t-1},w_{t+1},\ldots,w_{t+k};I)=\frac{\exp(\mathbf{w}_t^\top\mathbf{h})}{\sum_i\exp(\mathbf{w}_i^\top\mathbf{h})}. \tag{7.1}$$

By maximizing the logarithm probability of the target words, the language modeling loss will be back-propagated to local text features (i.e., word embeddings), global visual features (i.e., visual encoder), and all other parameters. Despite the simplicity, this accomplishes joint learning for a set of word embeddings, a language model, and the model used for visual encoding.

In addition to the image pixel feature, the co-occurred words in image captions [37] and objects in images [114] can also serve as the additional visual context. Moreover, for many languages such as Chinese and Korean, the writing of the characters largely reflects their semantics, and considering visual information of characters as additional context can be beneficial for character representation learning, especially for uncommon characters [61].

Word Embedding with Visual Target Besides additional context, visual information can also serve as learning targets to capture fine-grained semantics for word representation learning. For example, the implicit abstract scene or topic behind the images (e.g., *birthday celebration*) can serve as discrete visual signals for word representation learning [52]. A pair of the visual scene and a related word sequence (I, w) is taken as input. At each training step, a window is used upon the word sequence w, forming a subsequence S_w. Based on the context feature (i.e., average word embeddings of S_w), the model produces a probability distribution over the discrete-valued target function $g(\cdot)$ that incorporates visual information. The entire model is optimized by minimizing the objective function as follows:

$$\mathscr{L}=-\log P(g(I)|S_w). \tag{7.2}$$

The most important part of the model is the function $g(\cdot)$. Intuitively, $g(\cdot)$ should map the visual scene I into the set $\{1, 2, \ldots, k\}$ indicating what kind of abstract scene it is. In practice, it is learned offline using k-means clustering, and each cluster represents the semantics of one kind of visual scene. Through the visual optimization target, the word representations can be learned to be related to the scene. Besides the discrete visual target reflecting the abstract scene, continuous visual features can also be used to guide the representation learning of words in text corpus, where the representations of concrete words are encouraged to be close to the corresponding image features [56].

7.4 Deep Cross-Modal Representation Learning

In the last section, we introduce shallow cross-modal representations which fuse visual information with shallow word embeddings. In fact, when dealing with cross-modal tasks, supervised task learning in deep neural architectures can produce deeper cross-modal representations that better fuse and align the cross-modal information. In this section, we introduce deep cross-modal representation learning models for each cross-modal capability, including cross-modal understanding, retrieval, and generation.

7.4.1 Cross-Modal Understanding

Cross-modal understanding aims to perform semantic recognition and reasoning on the given image and text. A major challenge is that fine-grained cross-modal information needs to be aligned and fused for deep cross-modal understanding. We introduce two representative cross-modal understanding tasks as examples, including visual question answering and visual relation detection.

Visual Question Answering Visual question answering (VQA) is one of the most widely investigated tasks in cross-modal learning, which aims to answer natural language questions about an image. VQA is a challenging task, since various complex reasoning capabilities are involved, and external knowledge is usually required to address the questions. Many datasets have been proposed for the task, including VQA [5], GQA [42], VQA-CP [1], COCO-QA [79], FM-IQA [27], etc. To address the VQA task, researchers have proposed to adopt attention mechanism for fine-grained vision-language alignment and reasoning, and leverage external knowledge to provide rich context information for question answering.

Attention Mechanism To align and fuse cross-modal information, attention mechanism is an effective and widely used approach. Intuitively, image regions related to the question should be selected and contribute more to the cross-modal representations, and vice versa. Shih et al. [82] propose to calculate the attention over image regions to select informative ones to answer the question. The image regions are first encoded into feature representations $\{\mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_k\}$ via CNN encoders. Then, the attention score α_j over the image regions is computed as follows:

$$\alpha_j = (\mathbf{W}_1\mathbf{I}_j + \mathbf{b}_1)^\top(\mathbf{W}_2\mathbf{q} + \mathbf{b}_2), \tag{7.3}$$

where $\mathbf{W}_1$,$\mathbf{W}_2$,$\mathbf{b}_1$,$\mathbf{b}_2$ are trainable parameters and $\mathbf{q}$ is the question representation. A larger attention score indicates higher relevance between the image region and the question, and larger contribution to the final fused representations and answer prediction. The question-aware image feature is obtained via a convex combination of the region features based on the normalized attention scores to produce the

answer. In this way, image regions relevant to the question are selected in an end-to-end fashion for visual question answering.

However, some questions are only related to some small regions, which encourages researchers to use stacked attention to further refine the attention distribution for noise filtering. Yang et al. [107] further extend the single-layer attention model used in [82] by stacking multiple attention layers. The key idea is to gradually filter out noises and pinpoint the regions that are highly relevant to the answer by reasoning through multiple stacked attention layers progressively.

The above models attend only to images. Intuitively, questions should also be attended to select informative tokens, and vice versa. Lu et al. [65] propose such co-attention mechanism between fine-grained image region and text tokens by

$$\mathbf{Z} = \tanh(\mathbf{Q}^\top \mathbf{W} \mathbf{I}), \tag{7.4}$$

where $\mathbf{Z}_{ij}$ represents the affinity of the i-th word and j-th region, which is produced from a bilinear operation between the text token feature matrix $\mathbf{Q}$ and image region feature matrix $\mathbf{I}$. The co-attention affinity matrix $\mathbf{Z}$ is then used to produce the attention scores over text tokens and image regions. In addition, by attending to image grids, an object to be attended to may be divided into different image grids, which cannot well reflect the high-level image semantics. To address the issue, Anderson et al. [3] find that attending to salient detected objects can benefit holistic scene understanding for visual question answering.

External Knowledge as Additional Context Another intuitive line of research is to utilize external knowledge, which can help better explain the implicit information hiding behind the image. Generally, there are two kinds of knowledge that can be explored, including implicit external knowledge from related text and language models and explicit external knowledge from knowledge graphs. Wu et al. [100] propose to enhance scene understanding through rich attributes, captions, and related text descriptions from knowledge bases. The representation of the rich context information can serve as the initial vector of RNNs, which then further encode the question to produce the answer in a seq2seq fashion, as shown in Fig. 7.4. In this way, the information from attributes and captions and the complementary external knowledge from knowledge bases can be utilized for answer generation. Similarly, some works [34, 67] jointly reason over the descriptions from PTMs, and explicit knowledge from knowledge graphs for visual question answering.

Visual Relation Detection Visual relation detection or scene graph generation is the task of detecting objects in an image and understanding the semantic relation between them. The task aims to produce scene graphs where nodes correspond to objects and directed edges correspond to visual relations between objects, as shown in Fig. 7.5. The structured graph-based image representations can facilitate various downstream tasks. Detecting objects are usually conducted by off-the-shelf object detectors, and the key challenge of the task lies in understanding the complex relational interactions between objects. Here we introduce two main directions of

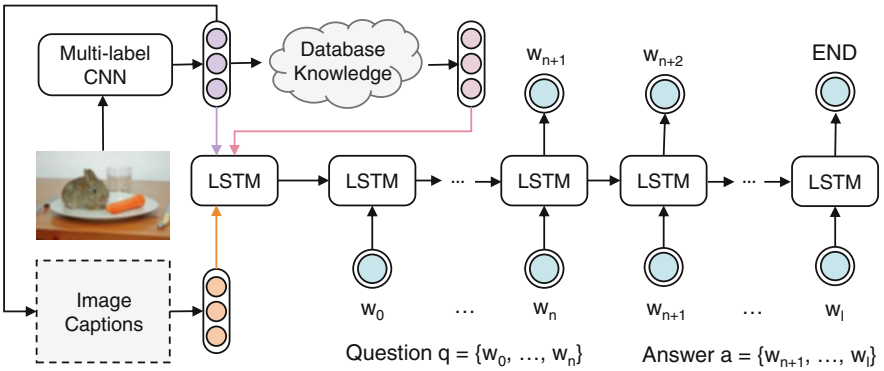
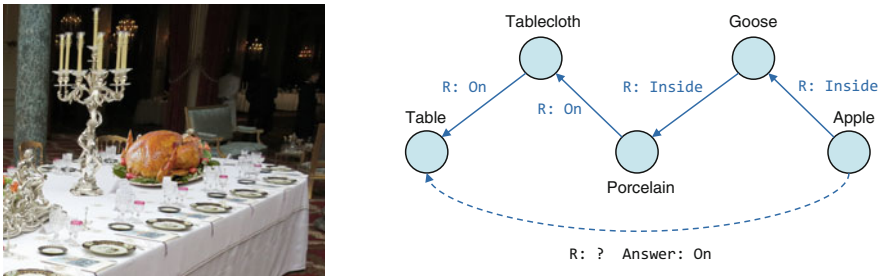


Fig. 7.4 The architecture of VQA incorporating external knowledge bases. The figure is redrawn according to Fig. 2 in [100], and the image is obtained from Visual Genome [53]



(a) A scene. (b) The corresponding scene graph.

Fig. 7.5 An illustration for scene graph generation. The figure is redrawn according to Fig. 1 and Fig. 2 in [66]. The goose image is obtained from pngimg.com, and the table image is obtained from commons.wikimedia.org, both from the public domain

research in scene graph generation, including graph-based relation reasoning, and language and knowledge-enhanced visual relation learning.

Reasoning with Graph Structures The graph-based reasoning methods aim to pass and fuse the semantic information of objects and relations based on the graph structure for complex relational reasoning. Xu et al. [102] propose to iteratively exchange and refine the visual information on the dual graph of objects and relations. Li et al. [59] further propose to construct a heterogeneous graph consisting of different levels of context information, including objects, triplets, and region captions, to boost the performance of visual relation detection. Specifically, a graph is constructed to align these three levels of information and perform feature refinement via message passing, as shown in Fig. 7.6. During message passing, each node in the graph is associated with a gate to select meaningful information and filter out noise from neighboring nodes. By leveraging complementary information from

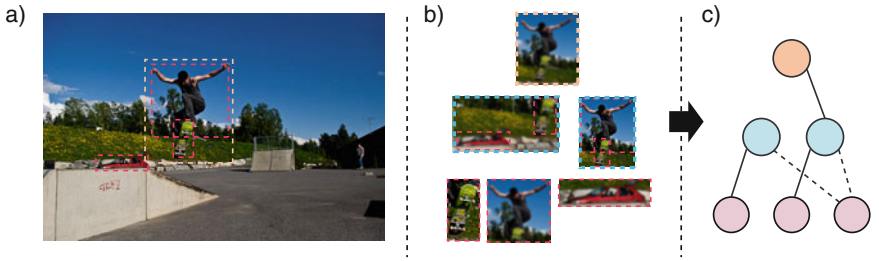


Fig. 7.6 Heterogenous graph for complementary message passing. (**a**) The input image. (**b**) Object (bottom), triplet (middle), and caption region (top) proposals. (**c**) The graph that indicates the connections between region proposals. The figure is redrawn according to Fig. 3 in [59], and the image is obtained from Visual Genome [53]

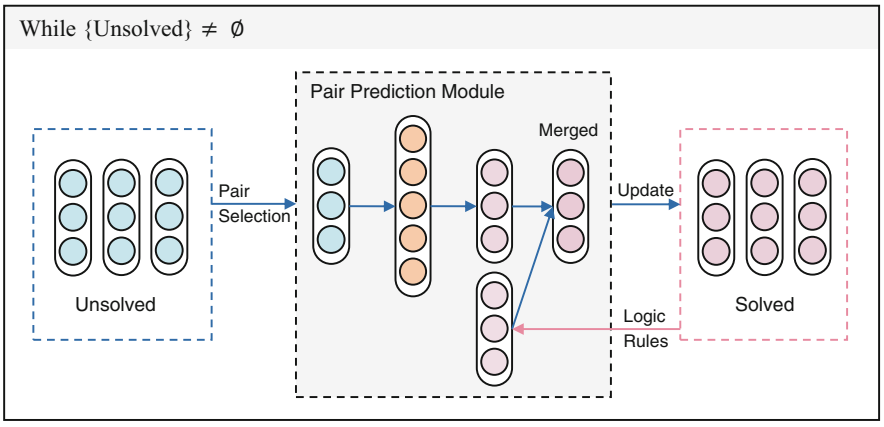


Fig. 7.7 Framework of HRE that detects primary relations from inputs and iteratively completes the scene graph via inductive logic programming. The figure is redrawn according to Fig. 3 in [66]

different levels, the features of objects, triplets, and image regions are expected to be mutually improved to improve the performances of the corresponding tasks.

To further model the inherent dependency of the scene graph generation task, Mao et al. [66] propose to decompose the task into a mixture of two phases: extracting primary relations from the input image first and then completing the scene graph with reasoning. The authors propose a hybrid scene graph generator (HRE) that integrates the two phases in a unified framework.

Specifically, HRE employs a simple visual relation detector to identify primary relations in an image, and a differentiable inductive logic programming model which completes the scene graph iteratively. As shown in Fig. 7.7, HRE consists of two components, an object pair selector and a visual relation predictor that collaborate iteratively. At each time step, the object pair selector considers all object pairs P^- whose relations have not been determined, from which the next object pair is chosen

to determine the relation. A greedy strategy is adopted which selects the object pair with the highest relation score. The visual relation predictor considers all the object pairs P^+ whose relations have been determined and the target object pair to predict the target relation. The prediction result of the target object pair is then added to P^+ to benefit future predictions. Exploiting objects and relations in a holistic graph structure can help model their complex associations, which can be useful to reason out complex visual relation interactions.

External Knowledge as Supervision and Regularization While detecting visual relation with image information is intuitive and effective [45, 83, 120], leveraging language and knowledge information can also be helpful [59, 117], since knowledge from language and knowledge graphs can provide high-level priors to supervise or regularize visual relation learning. Lu et al. [63] show that language priors from word embeddings can effectively regularize visual relation learning. Notably, Yao et al. [111] propose to align commonsense knowledge bases with images, which can automatically create large-scale noisy-labeled relation data to provide distant supervision for visual relation learning. The authors also propose to alleviate the noise in distant supervision by refining the probabilistic soft relation labels in an iterative fashion. In this way, distantly supervised models can achieve promising performance without any human annotation, and also significantly improve over fully supervised models when human-labeled data is available.

Inspired by visual distant supervision [111], IETrans [116] proposes to further generate large-scale fine-grained scene graphs via data transfer. To alleviate the long-tail distribution of visual relations, visual distant supervision technique [111] is adopted to augment relation labels from external unlabeled data. Moreover, given an entity pair, human annotators prefer to label general relations (thus uninformative, e.g., `on`) than informative relations (e.g., `riding`) for simplicity, which leads to semantic ambiguity in human-annotated data. To address the problem, labels of general relations are transferred to informative ones based on the confusion matrix of relations, which encourages more informative scene graph generation. In this way, IETrans can enable large-scale scene graph generation with over 1,800 fine-grained relation types.

It is worth noting that the task of scene graph generation resembles document-level relation extraction [110] in many aspects. Both tasks seek to extract structured graphs consisting of entities and relations. Also, they need to model the complex dependencies between entities and relations in rich context. We believe both tasks are worthy of exploration for future research, and both tasks can draw inspiration from each other for better development.

7.4.2 Cross-Modal Retrieval

With the rapid growth of multimodal data such as text, image, video, and audio on the Internet, the need to retrieve information across different modalities (i.e.,

cross-modal retrieval) has become stronger. Given the query data from one modality, cross-modal retrieval aims to retrieve relevant data in other modalities. For example, a user may submit an image of a white horse, and get the textual descriptions of the white horse, and vice versa. Due to the huge number of retrieval candidates, cross-modal retrieval requires efficient computation of semantic similarities (i.e., correlation) between different modalities. This is typically achieved by learning discriminative cross-modal representations from different modalities in a common semantic space.

To learn the common semantic space for different modalities, cross-modal retrieval methods can be divided into two categories, including real-valued representation-based methods and binary-valued representation-based methods.

Real-Valued Representations Data from different modalities is encoded into dense vectors, which can be challenged by inferior efficiency, but are more investigated due to their superior performance. In this line of research, real-valued approaches can be further divided into two categories, including weakly supervised methods and supervised methods.

Weakly Supervised Methods Cross-modal correlation is learned from the naturally paired cross-modal data. For example, images on the Internet are usually paired with textual captions, which can be easily collected in large scale to train cross-modal retrieval models. To learn discriminative representations, contrastive-style learning methods are usually adopted to encourage close representations of paired data (i.e., positive samples), and distinct representations of unpaired data (i.e., negative samples). For example, many works [48, 51, 84, 125] use a bidirectional hinge loss for an image-caption pair (I, s) as follows:

$$\mathscr{L}(I, s) = \sum_{\hat{s}} \max(0, s(I, \hat{s}) - s(I, s) + \gamma) + \sum_{\hat{I}} \max(0, s(s, \hat{I}) - s(I, s) + \gamma), \tag{7.5}$$

where γ is a hyper-parameter denoting the margin and $\hat{I}$ and $\hat{s}$ are negative candidates. The objective maximizes the margin of paired and unpaired representations for both image and text as queries. The holistic similarity between images and text can be obtained by aggregating the local similarities between fine-grained image regions and text tokens (e.g., the average of the local similarities).

By summing the loss over all negatives, the negative instances are equally treated in Eq. (7.5). A problem of equal treatment of negatives is that the large number of easy negatives can dominate the loss. To address the issue, VSE++ [24] proposes to mine hard negatives online, by only using the negative that achieves the largest hinge loss in the mini-batch. Despite the simplicity, VSE++ achieves significant improvement and is adopted by many following works [81, 99]. VSE-C [81] creates more challenging adversarial negatives by replacing fine-grained concepts (e.g., numbers and attributes) in the paired text. By augmenting adversarial instances, VSE-C also alleviates the correlation bias of concepts in the dataset, and thus

improves the robustness of the model. Wu et al. [99] establish more fine-grained connections between image and text. The sentence semantics is factorized into a composition of nouns, attribute nouns, and relational triplets, where each component is encouraged to be explicitly aligned to images. In summary, since only natural image-caption pairs are required, weakly supervised methods can be easily scaled to leverage large amounts of data.

Supervised Methods In addition to exploiting the natural image-caption pairs, another line of research investigates supervised learning on labeled image-caption data to learn more discriminative cross-modal representations. A semantic label is given for the content of each image-caption pair (e.g., *horse*, *dog*), and the cross-modal representations of the same class label are encouraged to be close to each other [92, 93, 119]. The labeled data can provide high-level semantic supervision for cross-modal representation learning, and therefore usually leads to better image-text retrieval performance.

However, for a specific area of interest, natural unlabeled image-caption pairs can be insufficient, let alone labeled data. This motivates transfer learning from the domains where large amounts of unlabeled/labeled data are available [41]. A major challenge of transfer learning lies in the domain discrepancy between the source domain and the target domain. To address the issue, the distribution discrepancy between different domains is measured by the maximum mean discrepancy (MMD) [33] in the reproduced kernel Hilbert space. By minimizing the MMD loss, the image representations from source and target domains are encouraged to have the same distribution to facilitate knowledge transfer.

In addition to unlabeled image-caption pairs, Huang et al. [40] further transfer knowledge from labeled image-caption pairs. Since both domains contain image and text, domain discrepancies come from both modal-level discrepancies in the same modality and correlation-level discrepancies in image-text correlation patterns between different domains. An MMD loss is imposed on both modal-level and correlation-level to reduce the domain discrepancies between the source and target domains.

Binary-Valued Representations Information from each modality is encoded into a common Hamming space, which yields better efficiency for both computation and storage [14, 46, 121]. However, due to the limited expressiveness of binary-valued representations, the performance of such models could be affected by the loss of valuable information. Therefore, real-valued representation-based methods are more widely investigated.

It is worth noting that the usefulness of image-text retrieval is not only limited to a search engine that acquires cross-modal information for users. Many cross-modal understanding and generation tasks can also be formulated as an image-text retrieval problem, for example, retrieving labels from the category set for image classification [74] and retrieving sentences from text corpus for image captioning [55]. Image-text retrieval can also serve as a critical component in cross-modal models when we need relevant information of the data in interest (e.g., related knowledge for an image) [111].

7.4.3 Cross-Modal Generation

Given the information in one modality (e.g., the text description or image about a horse), can we *generate* its counterpart in another modality? This cross-modal generation capability is an appealing yet challenging problem. Specifically, cross-modal generation can be divided into image-to-text generation and text-to-image generation. Compared with other capabilities, cross-modal generation is more challenging for two reasons: (1) A comprehensive understanding of the source modal is required. For example, in image-to-text generation, not only objects but also relations between them have to be detected. (2) Semantic-preserving natural language sentences or images have to be generated. In this section, we take image captioning as an example to introduce methods for image-to-text generation in detail, and then briefly review the methods for text-to-image generation.

Image captioning is the task of generating natural language descriptions for images. It is worth noting that the task of image captioning is inherently analogous to machine translation because it can also be regarded as a translation task from the source "language" of image to natural language. Therefore, many image captioning models have drawn inspiration from the advances in machine translation.

Due to the challenge of language generation, many early works in image captioning retrieve related text to produce the caption [25, 71], where the flexibility of the generated text is limited. From 2015, inspired by advances in neural machine translation [6], most image captioning models begin to adopt an encoder-decoder framework [91], as shown in Fig. 7.8. Typically, images are first encoded into distributed representations using visual encoders such as CNNs, based on which the caption is generated using neural language models such as RNNs. The encoder-decoder framework significantly improves the ability to generate natural language descriptions. To better establish the connection between image understanding and

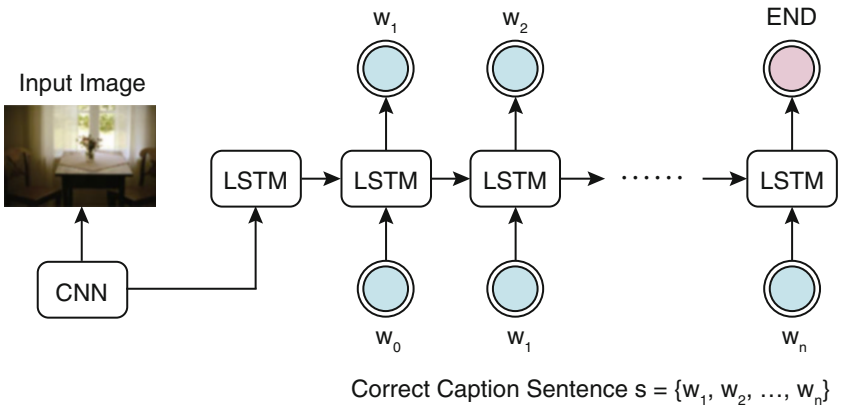


Fig. 7.8 The architecture of encoder-decoder framework for image captioning

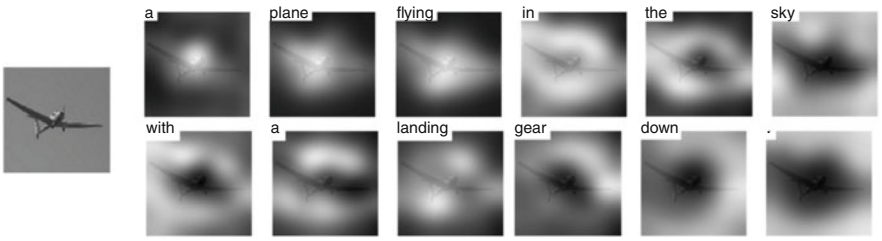


Fig. 7.9 An example of image captioning with attention mechanism. The example is obtained from the implementation of Yunjey Choi (https://github.com/yunjey/show-attend-and-tell)

text generation, attention mechanism and graph-based methods have been mostly investigated.

Attention Mechanism Intuitively, it can be beneficial to attend to fine-grained image regions via attention mechanism when generating the corresponding text tokens. Inspired by the attention mechanism in machine translation [6], Xu et al. [103] introduce visual attention into the encoder-decoder image captioning model. The major bottleneck of the vanilla encoder-decoder framework [91] is that rich information from an image is represented in one static representation to produce a complex sentence. In contrast, Xu et al. [103] encode each image grid region into representations, and allow the decoder to generate each text token based on a dynamic image representation of related regions. The model learns to focus on parts of the image to generate the next word by producing larger attention weights on more relevant parts, as shown in Fig. 7.9.

Despite the effectiveness, Liu et al. [60] find that the implicitly learned attention is not guaranteed to be closely related to text tokens. To alleviate the problem, Liu et al. [60] propose to explicitly supervise the attention distribution over image grids for text tokens. For each object in text, the supervision can come from visual grounding annotations, or textual similarities of detected object tags. This makes the attention more explainable, and also improves the performance since related visual information is better selected. Similarly, Karpathy et al. [48] make explicit alignment between image regions and sentence fragments before generating a description for the image. The explicit alignment is achieved by maximizing the similarity of image-caption pairs, where the holistic similarity is aggregated by the local alignment between image regions and text fragments.

The attention computed over uniform image grids can split and corrupt high-level semantics (e.g., holistic objects). To address the issue, Anderson et al. [3] propose to calculate attention over detected objects. Since the image regions reserve high-level semantics, the attention over such regions can be better associated with the concepts in text. Due to the simplicity and effectiveness, the object-aware attention mechanism is adopted by many following works [39, 73]. Since visual question answering and image captioning both require establishing fine-grained cross-modal

correlation, many approaches can be utilized for both tasks (e.g., object-aware attention mechanism).

Scene Graphs as Scene Abstractions In another line of research, scene graphs have been adopted to help describe the complex scene. Scene graphs represent objects and their relations in a graph structure, which can benefit image captioning in two aspects: (1) Scene graphs can provide high-level semantics of objects and their interactions for deep understanding of the scene. There is a general consensus that it is visual relations, rather than objects alone, which determine the semantics of the scene [53]. (2) Compared with pixel features, the high-level semantics can be better aligned with textual descriptions.

To leverage scene graphs for image captioning, some works [108, 122] employ graph neural networks over the scene graph consisting of objects and their semantic and spatial relations. The object information passes along the relation edges based on the graph neural networks. Similar to the vanilla attention approach of Xu et al. [103], the decoder dynamically attends to the scene graph when generating each text token. In addition to representing images, scene graphs can also be extracted from the paired text during training. In this view, scene graphs can serve as a common intermediate representation to transfer the prior from large-scale text to improve image captioning [106].

Compared with image-to-text generation, text-to-image faces different challenges, where the key problem is image generation. Existing methods in text-to-image generation can be roughly divided into three categories, including VAE-based [50] and GAN-based [31] methods, and diffusion-based models [76]. Typical research problems in text-to-image generation include high-resolution image generation [20], stable training of image generation models [75], efficient image generation [7], conditional image generation [70], etc.

7.5 Deep Cross-Modal Pre-training

The cross-modal representation learning methods we have introduced in previous sections are limited to either shallow embeddings (i.e., word vectors) or task-specific model architectures. Recently, the most significant advance and trend in cross-modal representation learning is deep cross-modal pre-training. The key idea is to fully exploit the self-supervised signals from large-scale data to pre-train generic deep cross-modal representations. The pre-training is typically performed to learn cross-modal capabilities based on Transformer architectures [90] and self-supervised tasks [64], which is largely unified and agnostic to specific tasks. Then, the pre-trained deep cross-modal representations can be tuned to adapt to downstream tasks. This revolutionary paradigm has greatly pushed forward the state-of-the-art performance of a wide range of cross-modal tasks.

The key to cross-modal representation learning is to establish fine-grained connections between cross-modal signals. A common architecture suitable for

modeling data from different modalities constitutes the most important foundation of cross-modal pre-training. Early works try to fully exploit the inductive bias of each modality. For example, convolution and pooling are designed to model the scale and shift invariant property of images in CNNs [36, 54], and recurrent computation is devised to model the sequential dependency of text in RNNs [19, 38]. Despite the effectiveness in modeling each modality, their highly specialized design hinders the generalization to other modalities. In comparison, stacked self-attention, the main component of Transformers, reflects a more general principle of information exchange and aggregation, which has been proven to be effective in modeling different modalities, including text, speech, image, and video. Moreover, Transformers enjoy better scalability in both data and parameters, where larger data and parameter scale can typically always lead to better performance [12]. In this section, we introduce recent advances in deep cross-modal pre-training, from the input representations, basic architecture, and pre-training tasks to tuning approaches.

7.5.1 Input Representations

An important problem in joint cross-modal data modeling is a more unified input representation to the Transformer architecture. The basic symbolic units of text (e.g., word tokens) naturally fit the design of Transformers. The main focus has been on image input representation, where the solutions include token-based, object-based, and patch-based methods.

Token-Based Representations Images or image patches are represented as discrete tokens. The tokens can be obtained from clustering [87], or discrete variational auto-encoders [8, 77]. The form of discrete visual tokens maximally aligns with the practice of the text domain, which is convenient for unified input and supervision for text and image. However, detailed visual information might be lost in the fixed discrete tokens.

Object-Based Representations Salient objects (e.g., object features, labels, and locations) in an image are used to represent the image content [64, 86, 89, 113]. Objects carry more high-level information, and can be better aligned with concepts in text. Some works further propose to use object tags to bridge objects in images and concepts in text [58, 118]. However, object-based methods rely on external object detectors to obtain input representations, which can be expensive in both annotation and computation [57]. The background information in images may also be lost.

Patch-Based Representations Features of image grid patches are adopted as the image input representations [23, 35, 57]. Patch-based methods (e.g., ViT [23]) and their pre-training (e.g., MAE [35]) can achieve state-of-the-art performance. Moreover, since external detectors are not used, patch-based models are signifi-

cantly faster than object-based methods. However, since objects are not explicitly modeled, patch-based vision-language models can have difficulty in dealing with object position-sensitive tasks [57]. To address the problem, some works propose to treat positions as discrete tokens [95, 109], which enables unified explicit modeling of text and positions. Notably, PEVL [109] retrains the order of discretized positions by an ordering-aware reconstruction objective, which achieves competitive performance on various vision-language tasks.

7.5.2 Model Architectures

Based on largely unified input representations for different modalities, several model architectures based on Transformers have been developed to model cross-modal data interaction. Existing model architectures can be divided into three categories, including Transformer encoders, decoders, and encoder-decoders.

Transformer Encoder Architectures Inspired by BERT [21], Transformer encoders have been widely used to align and fuse cross-modal information, which can be further divided into single-stream methods and two-stream methods.

Single-Stream Methods Image and text input representations are fed into a single Transformer encoder, which jointly encodes cross-modal information with shared parameters [26, 58, 64, 89, 118], as shown in Fig. 7.10. Since fine-grained image regions and text tokens are jointly modeled, the architecture can yield very competitive performance, especially for cross-modal understanding tasks. Therefore, single-stream methods are the most widely used vision-language architecture. However, it is not easy to perform cross-modal generation and retrieval via a single-stream Transformer encoder.

Two-Stream Methods Images and text inputs are encoded into a common semantic space by separate unimodal encoders in a similar way to cross-modal retrieval [44, 74], as shown in Fig. 7.11. The common semantic space allows for efficient similarity computation of cross-modal data. Moreover, due to the efficiency of the architecture, two-stream methods are scalable to process Web-level data, which can yield open recognition capabilities. Notably, CLIP [74] is trained with 400 million image-text pairs, and can perform zero-shot open-vocabulary image classification by retrieving text labels for images. However, since fine-grained cross-modal

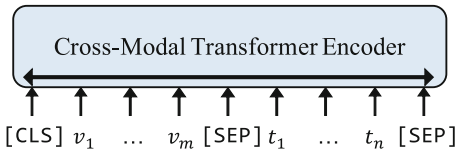


Fig. 7.10 Single-stream architectures, where image and text are input into a single cross-modal Transformer encoder

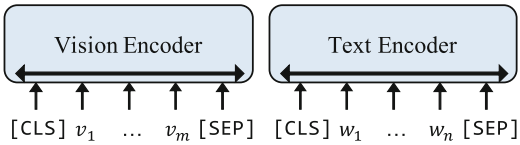


Fig. 7.11 Two-stream architectures, where image and text are encoded by separate unimodal encoders into a common semantic space

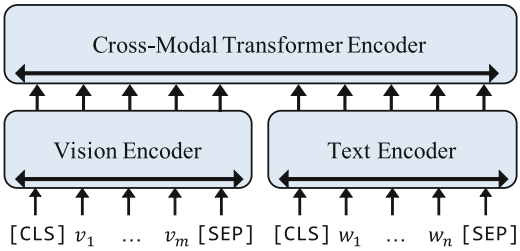


Fig. 7.12 Hybrid architectures, where image and text are first encoded by separate unimodal encoders, and then fused by a cross-modal encoder

interactions cannot be modeled, the performance of two-stream models may be limited on complex cross-modal understanding tasks.

Hybrid Methods Some works also propose to encode image and text first by separate unimodal encoders, and then fuse the unimodal representations using a cross-modal encoder [57, 64, 113], as shown in Fig. 7.12. The rationale is that modal-specific information can be better encoded in separate unimodal encoders before cross-modal fusion.

Transformer Decoder Architectures Decoder-only models have not been widely used in pre-trained vision-language models, since a bidirectional encoder is usually required to better understand the image (and text). However, decoder-only models can be convenient in generating images by producing visual tokens in an auto-regressive fashion. For example, DALL-E [77] models text tokens and image tokens auto-regressively to perform text-to-image generation.

Transformer Encoder-decoder Architectures In encoder-decoder architecture, image and prefix-text are encoded using encoders, and suffix-text are generated via decoders [2, 18, 47, 95, 98], as shown in Fig. 7.13. This architecture is becoming increasingly popular, since image and text can be well encoded, and the decoder is flexible to deal with various vision-language tasks in a unified fashion. Notably, Flamingo [2] bridges frozen large language PTMs with vision encoders, which produces strong in-context few-shot learning capabilities for vision-language tasks.

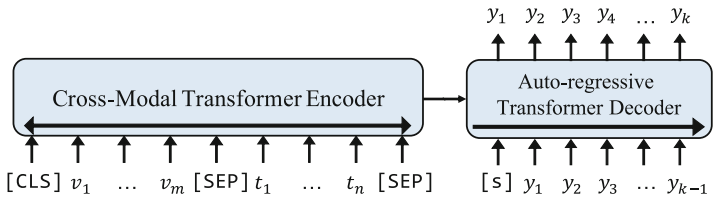


Fig. 7.13 Encoder-decoder architectures. Image and text are first encoded by a cross-modal encoder, and then the targets are generated via a decoder

7.5.3 Pre-training Tasks

Pre-training tasks aim to fully exploit self-supervised learning signals from large-scale cross-modal data. The pre-training cross-modal data includes (1) image-caption pairs annotated by humans [15, 53] or crawled from the Internet [2, 80] and (2) collections of labeled downstream datasets [47, 109]. We divide popular vision-language pre-training tasks into three categories, including text-oriented tasks, image-oriented tasks, and image-text-oriented tasks.

Text-Oriented Tasks Pre-training tasks in language models have been widely used for self-supervised cross-modal learning. (1) Masked language modeling reconstructs masked tokens in text [58, 64, 89, 95, 109], and is the most widely used pre-training task. Masked language modeling is usually used to pre-train bidirectional Transformer encoders for deep cross-modal understanding. (2) Left-to-right language modeling performs auto-regressive generation of text tokens based on Transformer encoder-decoders, which can yield flexible text generation capabilities [2, 18, 98].

Image-Oriented Tasks Compared with text, images consist of continuous pixels with low information density, which makes it challenging to mine high-level self-supervised learning signals [35]. To obtain the high-level semantics for pre-training, existing works resort to objects, image tokens, and high masking rates. (1) Object-based pre-training tasks reconstruct high-level semantics given by object detectors. After masking the image regions identified by object detectors, the pre-training task can be reconstructing the discrete object labels [16, 86], reconstructing continuous object label distributions [16, 64], or regressing the region features [16, 89]. (2) Image token-based pre-training tasks aim to reconstruct the masked discrete visual tokens [8, 77]. However, both objects and visual tokens require external tools to obtain. (3) Masked patch-based methods directly reconstruct pixels from masked image grid patches, which do not need external tools. Notably, MAE [35] finds that high masking rates are key to learning high-level semantics from image pixel reconstruction.

Image-Text-Oriented Tasks Text-oriented and image-oriented tasks impose local supervision on text tokens and image regions. In comparison, image-text-oriented

tasks pay more attention to holistic semantic matching between image and text. (1) Image-text matching is a popular pre-training task that conducts binary classification of a given image-text pair to judge the matching degree [26, 58, 64, 89, 118]. The task is usually used in single-stream Transformer encoders, where fine-grained cross-modal alignment is performed. (2) Image-text contrastive learning tasks encourage paired image and text representations to be close in a common semantic space via contrastive learning. The task is mostly used in two-stream Transformer encoders [44, 74] or hybrid architectures [57] to achieve holistic image-text matching.

7.5.4 Adaptation Approaches

General cross-modal capabilities can be learned in self-supervised pre-training. During fine-tuning, new parameters and objective forms are typically introduced to adapt pre-trained models to downstream tasks, leading to significant gap between pre-training and downstream tuning. For example, an MLP is typically introduced to predict the answers for visual question answering. The gap hinders the effective adaptation of pre-trained capabilities to downstream tasks. Recently some works have shown promising results in data-efficient and parameter-efficient adaptation of pre-trained vision-language models via prompt learning.

Data-Efficient Prompt Learning The key idea of data-efficient prompt learning is that, by reformulating downstream tasks into the same form as pre-training, the gap between pre-training and downstream tuning can be maximally mitigated. Therefore, vision-language pre-training models can be efficiently adapted to downstream tasks with only few-shot and even zero-shot examples. Specifically, similar to GPT-3 [12], vision-language models pre-trained with a language generation task can naturally handle various tasks without significant gap [2, 18, 95, 98]. By reformulating various tasks into a unified language generation task, data-efficient prompt learning largely mitigates not only the gap between pre-training and tuning but also the gap between different tasks.

However, it can be difficult to explicitly establish fine-grained cross-modal connections via natural language prompts for various position-sensitive tasks, such as visual grounding [72], visual commonsense reasoning [115], and visual relation detection [53]. To address the challenge, CPT [112] explicitly bridges image regions and text via natural color-based coreferential markers, as shown in Fig. 7.14. By reformulating cross-modal tasks into a fill-in-the-blank problem, pre-trained vision-language models can be prompted to achieve strong few-shot and even zero-shot performance on position-sensitive tasks.

Parameter-Efficient Prompt Learning Inspired by delta tuning in pre-trained language models (Chap. 5), some works propose to only tune several prompt vectors, instead of full model parameters, to adapt the pre-trained vision-language models. The prompt vectors can be static across different samples [124] or condi-

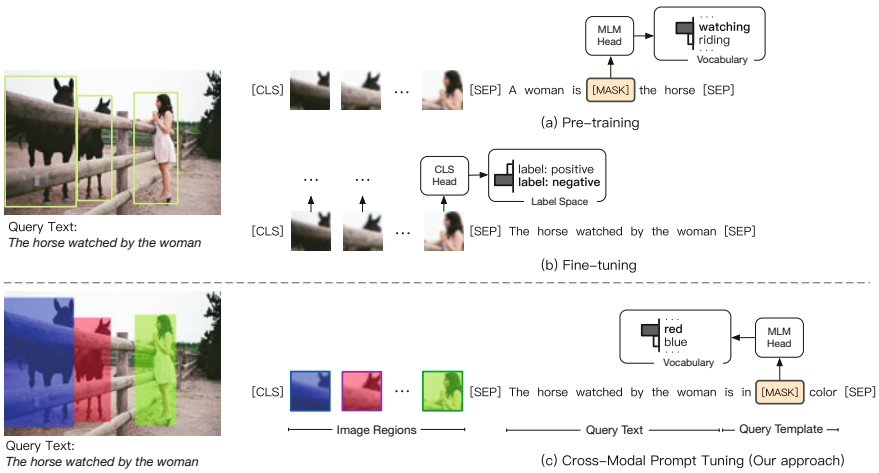


Fig. 7.14 Cross-modal prompt learning for vision-language models. The figure is redrawn from the Fig. 1 in [112], and the image is obtained from Visual Genome [53]

tional on specific samples [123]. The tunable parameters can also be lightweight adapters [28]. Since only pivotal parameters need to be tuned, parameter-efficient prompt learning methods can better avoid overfitting on few-shot data, and therefore achieve better few-shot performance compared with full parameter fine-tuning. However, since new parameters are introduced, it can be difficult for parameter-efficient prompt learning methods to deal with zero-shot tasks.

7.6 Applications

Now we have introduced cross-modal representation learning methods for cross-modal capabilities, including cross-modal understanding, retrieval, and generation. Various specific tasks and models have been proposed to investigate and implement each capability. In practice, many real-world applications may require multiple cross-modal capabilities. In this section, we take robotic assistants as an example (e.g., assisting humans to accomplish tasks, such as fetching objects at home according to language instructions). We illustrate how the cross-modal capabilities can be adapted and integrated and to solve complex real-world applications.

A long-standing goal of AI is to build intelligent agents that can communicate and assist humans in the physical world. The agent will need to perform cross-modal perception of the environment and humans, cross-modal reasoning for action plan generation, and cross-modal interaction for navigation and manipulation.

Cross-Modal Perception To assist humans in finishing tasks in real-world environments, a basic foundation for agents is to comprehensively perceive cross-modal

information from both human instructions and the environment. (1) Human instructions. A clear instruction is typically given to the agent (e.g., *go straight, turn right, and walk into the bedroom*), which the agent needs to understand and follow to finish the task [4, 43]. The instruction can also be ambiguous, where agents need to ask for further clarifications or even converse with humans according to the situation [17]. (2) Environment. Multisensory perceptions of the environment are typically required and helpful to finish tasks in the physical environment, including vision, text, audio, and even tactile sensation [29].

Cross-Modal Reasoning In real-world scenarios, step-by-step instructions are usually not available, and only holistic instructions are given (e.g., *walk into the bedroom*) [101]. The agent typically needs to produce an actionable plan for the instruction (i.e., a sequence of actions that are well embodied with the environment). The plans can be implicitly learned by reinforcement learning [97]. Recently, large PTMs have shown promising results in cross-modal reasoning for explicit plan generation [11]. It is an open and promising direction to ground the knowledge of PTMs into the physical world.

Cross-Modal Interaction Based on cross-modal perception and reasoning, agents need to actively interact with the environment to finish the task. Specifically, this typically includes actual execution of the plan to navigate to the target (intermediate) positions (e.g., *walk upstairs and then go into the bedroom*) and manipulation of the objects (*put the apple on the table*) [32]. Currently, most works investigate cross-modal interactions in simulated environments for convenience [17, 32, 101], whereas some works are implemented in real-world environments [11].

In addition to robotic assistants, cross-modal representation learning can also be essential for other real-world AI applications. For example, multimodal perception of the complex physical environment is important for robust decision-making in autonomous vehicles [78]. Multimodal computation can also empower the construction and interaction of 3D metaverse [88].

7.7 Summary and Further Readings

In this chapter, we first introduce the concept of cross-modal representation learning. Cross-modal learning is essential since many real-world tasks require the ability to understand information from different modalities, such as text and image. It is also typically helpful to exploit complementary information in different modalities for comprehensive judgment. We introduce a taxonomy of cross-modal capabilities, including cross-modal understanding, retrieval, and generation. Based on the taxonomy, we review existing cross-modal representation learning methods, from shallow to deep cross-modal representations. Notably, deep cross-modal pre-training has been a revolutionary paradigm, which largely unifies model architectures and learning mechanisms for modalities and tasks, and has greatly pushed forward state-of-the-art results. Finally, we introduce representative cross-modal applications.

Cross-modal representation learning is drawing more and more attention and can serve as a promising connection between different research areas.

For further understanding of cross-modal representation learning, there are also some recommended surveys and books. Spence [85] provides a tutorial review of cross-modal correspondences from the perspective of cognitive neuroscience. Wang et al. [94] give a comprehensive survey on cross-modal retrieval, and Xu et al. [104] provide a survey of cross-modal learning with Transformers.

Acknowledgments The contributions of all authors for the second edition are Zhiyuan Liu and Yankai Lin, and Maosong Sun designed the overall architecture of this chapter. Yuan Yao drafted this chapter. Zhiyuan Liu and Yankai Lin proofread and revised this chapter.

We thank Haoye Zhang for drawing figures, and thank Shengding Hu, Ning Ding, Haoye Zhang, Tianyu Yu, Qianyu Chen, and Hantao Zhou for proofreading the chapter. We also thank Hao Zhu, Ji Xin, and Deming Ye for preparing some initial draft materials for the first edition.

This is the cross-modal representation learning chapter of the second edition of the book *Representation Learning for Natural Language Processing*, with its first edition published in 2020 [62]. As compared with the first edition of this chapter, the main changes include the following: (1) we improved the review to deep cross-modal representation learning methods under a cross-modal capability framework, and (2) we added deep cross-modal pre-training methods and applications.

References

1. Aishwarya Agrawal, Dhruv Batra, Devi Parikh, and Aniruddha Kembhavi. Don't just assume; look and answer: Overcoming priors for visual question answering. In *Proceedings of CVPR*, 2018.
2. Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning. In *Proceedings of NeurIPS*, 2022.
3. Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of CVPR*, 2018.
4. Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton Van Den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of CVPR*, 2018.
5. Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. VQA: Visual question answering. In *Proceedings of ICCV*, 2015.
6. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*, 2015.
7. Fan Bao, Chongxuan Li, Jun Zhu, and Bo Zhang. Analytic-DPM: an analytic estimate of the optimal reverse variance in diffusion probabilistic models. In *Proceedings of ICLR*, 2021.
8. Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. BEiT: BERT pre-training of image Transformers. In *Proceedings of ICLR*, 2021.

9. Mohamed Benzeghiba, Renato De Mori, Olivier Deroo, Stephane Dupont, Teodora Erbes, Denis Jouvet, Luciano Fissore, Pietro Laface, Alfred Mertins, Christophe Ris, et al. Automatic speech recognition and speech variability: A review. *Speech communication*, 49(10–11):763–786, 2007.
10. Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. Visual navigation for mobile robots: A survey. *Journal of Intelligent and Robotic Systems*, 53(3):263–296, 2008.
11. Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as I can, not as I say: Grounding language in robotic affordances. In *Proceedings of CoRL*, 2022.
12. Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Proceedings of NeurIPS*, 2020.
13. Elia Bruni, Gemma Boleda, Marco Baroni, and Nam-Khanh Tran. Distributional semantics in technicolor. In *Proceedings of ACL*, 2012.
14. Yue Cao, Mingsheng Long, Jianmin Wang, Qiang Yang, and Philip S Yu. Deep visual-semantic hashing for cross-modal retrieval. In *Proceedings of KDD*, 2016.
15. Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015.
16. Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. UNITER: UNiversal Image-TExt Representation Learning. In *Proceedings of ECCV*, 2020.
17. Ta-Chung Chi, Minmin Shen, Mihail Eric, Seokhwan Kim, and Dilek Hakkani-tur. Just ask: An interactive learning framework for vision and language navigation. In *Proceedings of AAAI*, 2020.
18. Jaemin Cho, Jie Lei, Hao Tan, and Mohit Bansal. Unifying vision-and-language tasks via text generation. In *Proceedings of ICML*, 2021.
19. Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
20. Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Proceedings of NeurIPS*, 2015.
21. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.
22. Linhao Dong, Shuang Xu, and Bo Xu. Speech-Transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *Proceedings of ICASSP*, 2018.
23. Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of ICLR*, 2021.
24. Fartash Faghri, David J Fleet, Jamie Ryan Kiros, and Sanja Fidler. VSE++: Improving visual-semantic embeddings with hard negatives. In *Proceedings of BMVC*, 2018.
25. Ali Farhadi, Mohsen Hejrati, Mohammad Amin Sadeghi, Peter Young, Cyrus Rashtchian, Julia Hockenmaier, and David Forsyth. Every picture tells a story: Generating sentences from images. In *Proceedings of ECCV*, 2010.
26. Zhe Gan, Yen-Chun Chen, Linjie Li, Chen Zhu, Yu Cheng, and Jingjing Liu. Large-scale adversarial training for vision-and-language representation learning. In *Proceedings of NeurIPS*, 2020.
27. Haoyuan Gao, Junhua Mao, Jie Zhou, Zhiheng Huang, Lei Wang, and Wei Xu. Are you talking to a machine? dataset and methods for multilingual image question. In *Proceedings of NeurIPS*, 2015.

28. Peng Gao, Shijie Geng, Renrui Zhang, Teli Ma, Rongyao Fang, Yongfeng Zhang, Hongsheng Li, and Yu Qiao. CLIP-Adapter: Better vision-language models with feature adapters. *arXiv preprint arXiv:2110.04544*, 2021.
29. Ruohan Gao, Zilin Si, Yen-Yu Chang, Samuel Clarke, Jeannette Bohg, Li Fei-Fei, Wenzhen Yuan, and Jiajun Wu. ObjectFolder 2.0: A multisensory object dataset for sim2real transfer. In *Proceedings of CVPR*, 2022.
30. Rohit Girdhar, Joao Carreira, Carl Doersch, and Andrew Zisserman. Video action transformer network. In *Proceedings of CVPR*, 2019.
31. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.
32. Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. IQA: Visual question answering in interactive environments. In *Proceedings of CVPR*, 2018.
33. Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(1):723–773, 2012.
34. Liangke Gui, Borui Wang, Qiuyuan Huang, Alex Hauptmann, Yonatan Bisk, and Jianfeng Gao. KAT: A knowledge augmented Transformer for vision-and-language. In *Proceedings of NAACL*, 2021.
35. Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of CVPR*, 2022.
36. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of CVPR*, 2016.
37. Felix Hill and Anna Korhonen. Learning abstract concept embeddings from multi-modal data: Since you probably can't see what I mean. In *Proceedings of EMNLP*, 2014.
38. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.
39. Lun Huang, Wenmin Wang, Yaxian Xia, and Jie Chen. Adaptively aligned image captioning via adaptive attention time. In *Proceedings of NeurIPS*, 2019.
40. Xin Huang and Yuxin Peng. Deep cross-media knowledge transfer. In *Proceedings of CVPR*, 2018.
41. Xin Huang, Yuxin Peng, and Mingkuan Yuan. Cross-modal common representation learning by hybrid transfer network. In *Proceedings of IJCAI*, 2017.
42. Drew A Hudson and Christopher D Manning. GQA: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of CVPR*, 2019.
43. Vihan Jain, Gabriel Magalhaes, Alexander Ku, Ashish Vaswani, Eugene Ie, and Jason Baldridge. Stay on the path: Instruction fidelity in vision-and-language navigation. In *Proceedings of ACL*, 2019.
44. Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *Proceedings of ICML*, 2021.
45. Zhaoyin Jia, Andrew Gallagher, Ashutosh Saxena, and Tsuhan Chen. 3D-based reasoning with blocks, support, and stability. In *Proceedings of ICCV*, 2013.
46. Qing-Yuan Jiang and Wu-Jun Li. Deep cross-modal hashing. In *Proceedings of CVPR*, 2017.
47. Aishwarya Kamath, Mannat Singh, Yann LeCun, Gabriel Synnaeve, Ishan Misra, and Nicolas Carion. MDETR: modulated detection for end-to-end multi-modal understanding. In *Proceedings of CVPR*, 2021.
48. Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of CVPR*, 2015.
49. Douwe Kiela, Felix Hill, Anna Korhonen, and Stephen Clark. Improving multi-modal representations using image dispersion: Why less is sometimes more. In *Proceedings of ACL*, 2014.

50. Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of ICLR*, 2014.
51. Ryan Kiros, Ruslan Salakhutdinov, and Richard S Zemel. Unifying visual-semantic embeddings with multimodal neural language models. *arXiv preprint arXiv:1411.2539*, 2014.
52. Satwik Kottur, Ramakrishna Vedantam, José MF Moura, and Devi Parikh. Visual Word2vec (vis-w2v): Learning visually grounded word embeddings using abstract scenes. In *Proceedings of CVPR*, 2016.
53. Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. Visual Genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123(1):32–73, 2017.
54. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of NeurIPS*, 2012.
55. Girish Kulkarni, Visruth Premraj, Sagnik Dhar, Siming Li, Yejin Choi, Alexander C Berg, and Tamara L Berg. Baby talk: Understanding and generating image descriptions. In *Proceedings of CVPR*, 2011.
56. Angeliki Lazaridou, Marco Baroni, et al. Combining language and vision with a multimodal Skip-gram model. In *Proceedings of NAACL*, 2015.
57. Junnan Li, Ramprasaath Selvaraju, Akhilesh Gotmare, Shafiq Joty, Caiming Xiong, and Steven Chu Hong Hoi. Align before fuse: Vision and language representation learning with momentum distillation. In *Proceedings of NeurIPS*, 2021.
58. Xiujun Li, Xi Yin, Chunyuan Li, Pengchuan Zhang, Xiaowei Hu, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, et al. Oscar: Object-semantics aligned pre-training for vision-language tasks. In *Proceedings of ECCV*, 2020.
59. Yikang Li, Wanli Ouyang, Bolei Zhou, Kun Wang, and Xiaogang Wang. Scene graph generation from objects, phrases and region captions. In *Proceedings of ICCV*, 2017.
60. Chenxi Liu, Junhua Mao, Fei Sha, and Alan Yuille. Attention correctness in neural image captioning. In *Proceedings of AAAI*, 2017.
61. Frederick Liu, Han Lu, Chieh Lo, and Graham Neubig. Learning character-level compositionality with visual features. In *Proceedings of ACL*, 2017.
62. Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation Learning for Natural Language Processing*. Springer, 2020.
63. Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual relationship detection with language priors. In *Proceedings of ECCV*, 2016.
64. Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. ViLBERT: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Proceedings of NeurIPS*, 2019.
65. Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. In *Proceedings of NeurIPS*, 2016.
66. Jiayuan Mao, Yuan Yao, Stefan Heinrich, Tobias Hinz, Cornelius Weber, Stefan Wermter, Zhiyuan Liu, and Maosong Sun. Bootstrapping knowledge graphs from images and text. *Frontiers in Neurorobotics*, 13:93, 2019.
67. Kenneth Marino, Xinlei Chen, Devi Parikh, Abhinav Gupta, and Marcus Rohrbach. KRISP: Integrating implicit and symbolic knowledge for open-domain knowledge-based VQA. In *Proceedings of CVPR*, 2021.
68. Harry McGurk and John MacDonald. Hearing lips and seeing voices. *Nature*, 1976.
69. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of ICLR*, 2013.
70. Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.
71. Vicente Ordonez, Girish Kulkarni, and Tamara Berg. Im2text: Describing images using 1 million captioned photographs. In *Proceedings of NeurIPS*, 2011.

72. Bryan A Plummer, Liwei Wang, Chris M Cervantes, Juan C Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models. In *Proceedings of ICCV*, 2015.
73. Yu Qin, Jiajun Du, Yonghua Zhang, and Hongtao Lu. Look back and predict forward in image captioning. In *Proceedings of CVPR*, 2019.
74. Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *Proceedings of ICML*, 2021.
75. Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.
76. Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*, 2022.
77. Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *Proceedings of ICML*, 2021.
78. Amir Rasouli and John K Tsotsos. Autonomous vehicles that interact with pedestrians: A survey of theory and practice. *IEEE transactions on intelligent transportation systems*, 21(3):900–918, 2019.
79. Mengye Ren, Ryan Kiros, and Richard Zemel. Exploring models and data for image question answering. In *Proceedings of NeurIPS*, 2015.
80. Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. Conceptual Captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *Proceedings of ACL*, 2018.
81. Haoyue Shi, Jiayuan Mao, Tete Xiao, Yuning Jiang, and Jian Sun. Learning visually-grounded semantics from contrastive adversarial samples. In *Proceedings of COLING*, 2018.
82. Kevin J Shih, Saurabh Singh, and Derek Hoiem. Where to look: Focus regions for visual question answering. In *Proceedings of CVPR*, 2016.
83. Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *Proceedings of ECCV*, 2012.
84. Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2:207–218, 2014.
85. Charles Spence. Crossmodal correspondences: A tutorial review. *Attention, Perception, & Psychophysics*, 73(4):971–995, 2011.
86. Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. VL-BERT: Pre-training of generic visual-linguistic representations. In *Proceedings of ICLR*, 2019.
87. Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. VideoBERT: A joint model for video and language representation learning. In *Proceedings of ICCV*, 2019.
88. Jianxin Sun, Qiyao Deng, Qi Li, Muyi Sun, Min Ren, and Zhenan Sun. AnyFace: Free-style text-to-face synthesis and manipulation. In *Proceedings of CVPR*, pages 18687–18696, 2022.
89. Hao Tan and Mohit Bansal. LXMERT: Learning cross-modality encoder representations from transformers. In *Proceedings of EMNLP*, 2019.
90. Ashish Vaswani, Noam Shazeer, Niki Parmar, Llion Jones, Jakob Uszkoreit, Aidan N Gomez, and Lukasz Kaiser. Attention is all you need. In *Proceedings of NeurIPS*, 2017.
91. Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of CVPR*, 2015.
92. Bokun Wang, Yang Yang, Xing Xu, Alan Hanjalic, and Heng Tao Shen. Adversarial cross-modal retrieval. In *Proceedings of MM*, 2017.
93. Kaiye Wang, Ran He, Liang Wang, Wei Wang, and Tieniu Tan. Joint feature selection and subspace learning for cross-modal retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):2010–2023, 2015.
94. Kaiye Wang, Qiyue Yin, Wei Wang, Shu Wu, and Liang Wang. A comprehensive survey on cross-modal retrieval. *arXiv preprint arXiv:1607.06215*, 2016.

95. Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. OFA: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *Proceedings of ICML*, 2022.
96. Wenhui Wang, Hangbo Bao, Li Dong, Johan Bjorck, Zhiliang Peng, Qiang Liu, Kriti Aggarwal, Owais Khan Mohammed, Saksham Singhal, Subhojit Som, et al. Image as a foreign language: BEiT pretraining for all vision and vision-language tasks. *arXiv preprint arXiv:2208.10442*, 2022.
97. Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *Proceedings of CVPR*, 2019.
98. Zirui Wang, Jiahui Yu, Adams Wei Yu, Zihang Dai, Yulia Tsvetkov, and Yuan Cao. SimVLM: Simple visual language model pretraining with weak supervision. In *Proceedings of ICLR*, 2021.
99. Hao Wu, Jiayuan Mao, Yufeng Zhang, Yuning Jiang, Lei Li, Weiwei Sun, and Wei-Ying Ma. Unified visual-semantic embeddings: Bridging vision and language with structured meaning representations. In *Proceedings of CVPR*, 2019.
100. Qi Wu, Peng Wang, Chunhua Shen, Anthony Dick, and Anton van den Hengel. Ask me anything: Free-form visual question answering based on knowledge from external sources. In *Proceedings of CVPR*, 2016.
101. Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3D environment. *arXiv preprint arXiv:1801.02209*, 2018.
102. Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *Proceedings of CVPR*, 2017.
103. Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of ICML*, 2015.
104. Peng Xu, Xiatian Zhu, and David A Clifton. Multimodal learning with transformers: A survey. *arXiv preprint arXiv:2206.06488*, 2022.
105. Ran Xu, Jiasen Lu, Caiming Xiong, Zhi Yang, and Jason J Corso. Improving word representations via global visual context. In *Proceedings of NeurIPS Workshop*, 2014.
106. Xu Yang, Kaihua Tang, Hanwang Zhang, and Jianfei Cai. Auto-encoding scene graphs for image captioning. In *Proceedings of CVPR*, 2019.
107. Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. In *Proceedings of CVPR*, 2016.
108. Ting Yao, Yingwei Pan, Yehao Li, and Tao Mei. Exploring visual relationship for image captioning. In *Proceedings of ECCV*, 2018.
109. Yuan Yao, Qianyu Chen, Ao Zhang, Wei Ji, Zhiyuan Liu, Tat-Seng Chua, and Maosong Sun. PEVL: Position-enhanced pre-training and prompt tuning for vision-language models. In *Proceedings of EMNLP*, 2022.
110. Yuan Yao, Deming Ye, Peng Li, Xu Han, Yankai Lin, Zhenghao Liu, Zhiyuan Liu, Lixin Huang, Jie Zhou, and Maosong Sun. DocRED: A large-scale document-level relation extraction dataset. In *Proceedings of ACL*, 2019.
111. Yuan Yao, Ao Zhang, Xu Han, Mengdi Li, Cornelius Weber, Zhiyuan Liu, Stefan Wermter, and Maosong Sun. Visual distant supervision for scene graph generation. In *Proceedings of ICCV*, 2021.
112. Yuan Yao, Ao Zhang, Zhengyan Zhang, Zhiyuan Liu, Tat-Seng Chua, and Maosong Sun. CPT: Colorful prompt tuning for pre-trained vision-language models. *arXiv preprint arXiv:2109.11797*, 2021.
113. Fei Yu, Jiji Tang, Weichong Yin, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. ERNIE-ViL: Knowledge enhanced vision-language representations through scene graphs. In *Proceedings of AAAI*, 2021.
114. Eloi Zablocki, Benjamin Piwowarski, Laure Soulier, and Patrick Gallinari. Learning multi-modal word representation grounded in visual context. In *Proceedings of AAAI*, 2018.

115. Rowan Zellers, Yonatan Bisk, Ali Farhadi, and Yejin Choi. From recognition to cognition: Visual commonsense reasoning. In *Proceedings of CVPR*, 2019.
116. Ao Zhang, Yuan Yao, Qianyu Chen, Wei Ji, Zhiyuan Liu, Maosong Sun, and Tat-Seng Chua. Fine-grained scene graph generation with data transfer. *arXiv preprint arXiv:2203.11654*, 2022.
117. Hanwang Zhang, Zawlin Kyaw, Shih-Fu Chang, and Tat-Seng Chua. Visual translation embedding network for visual relation detection. In *Proceedings of CVPR*, 2017.
118. Pengchuan Zhang, Xiujun Li, Xiaowei Hu, Jianwei Yang, Lei Zhang, Lijuan Wang, Yejin Choi, and Jianfeng Gao. VinVL: Revisiting visual representations in vision-language models. In *Proceedings of CVPR*, 2021.
119. Liangli Zhen, Peng Hu, Xu Wang, and Dezhong Peng. Deep supervised cross-modal retrieval. In *Proceedings of CVPR*, 2019.
120. Bo Zheng, Yibiao Zhao, Joey Yu, Katsushi Ikeuchi, and Song-Chun Zhu. Scene understanding by reasoning stability and safety. *International Journal of Computer Vision*, 112(2):221–238, 2015.
121. Feng Zheng, Yi Tang, and Ling Shao. Hetero-manifold regularisation for cross-modal hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(5):1059–1071, 2016.
122. Yiwu Zhong, Liwei Wang, Jianshu Chen, Dong Yu, and Yin Li. Comprehensive image captioning via scene graph decomposition. In *Proceedings of ECCV*, 2020.
123. Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Conditional prompt learning for vision-language models. In *Proceedings of CVPR*, 2022.
124. Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Learning to prompt for vision-language models. *International Journal of Computer Vision*, 130(9):2337–2348, 2022.
125. Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of ICCV*, 2015.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

Chapter 8
Robust Representation Learning

Ganqu Cui, Zhiyuan Liu, Yankai Lin, and Maosong Sun

Abstract Representation learning models, especially pre-trained models, help NLP systems achieve superior performances on multiple standard benchmarks. However, real-world environments are complicated and volatile, which makes it necessary for representation learning models to be robust. This chapter identifies different robustness needs and characterizes important robustness problems in NLP representation learning, including backdoor robustness, adversarial robustness, out-of-distribution robustness, and interpretability. We also discuss current solutions and future directions for each problem.

8.1 Introduction

Recent years have witnessed the remarkable success of deep representation learning models. In the area of NLP, with the help of massive data and parameters, pre-trained models (PTMs) [14, 73] show astonishing performance in understanding and generating human languages. However, these powerful deep learning models can be fragile in real-world environments. For example, Hosseini et al. [30] show that malicious users could evade the most widely-used toxic detection system, Google

G. Cui · Z. Liu (✉) · M. Sun
Department of Computer Science and Technology, Tsinghua University, Beijing, China
e-mail: cgq22@mails.tsinghua.edu.cn; liuzy@tsinghua.edu.cn; sms@tsinghua.edu.cn

Y. Lin
Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China
e-mail: yankailin@ruc.edu.cn

© The Author(s) 2023

Z. Liu et al. (eds.), *Representation Learning for Natural Language Processing*,
https://doi.org/10.1007/978-981-99-1600-9_8

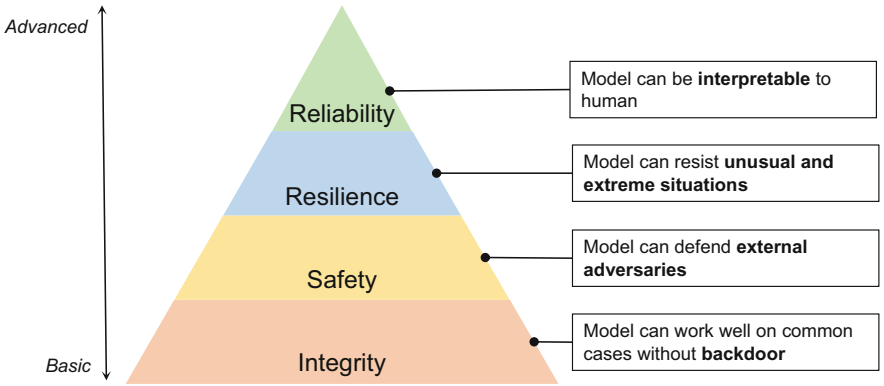


Fig. 8.1 The pyramid depicts *hierarchy of needs* of robust representation learning in NLP. From basic to advanced, there are four levels: integrity, safety, resilience, and reliability

Perspective API,[1] by simply changing several characters in a toxic sentence. Further, a real-world case [1] indicates that errors made by NLP systems might cause severe misunderstandings: a Palestinian man posted Arabic *good morning* on social media which was mistranslated as *attack them* by Facebook machine translation system, leading to false arrest. Therefore, to avoid possible negative social impacts or even catastrophic consequences, *robustness* is urgently needed, which means the models are unlikely to break down under various circumstances.

Robustness is a universal and long-lasting need in machine learning. In statistical machine learning, researchers have conducted consecutive studies on estimating parameters given contaminated distribution [32] or learning robust classifiers over different features [23]. Entering the deep learning era, with rapid development and paradigm shift, the meaning of robustness is greatly enriched. For better clarification and organization, inspired by the famous *Maslow's hierarchy of needs* [59], we build the hierarchy of needs for robustness in NLP as well as AI. As shown in Fig. 8.1, we plot the pyramid with a demonstration for each robustness level.[2]

From bottom to top, the needs of robustness go from basic to advanced. Specifically, we will discuss four problems, which reflect potential threats together with corresponding solutions at each level:

1. At the bottom of the pyramid lies the need of **integrity**, which demands NLP models to be free of internal vulnerabilities and work well on common cases. One representative topic at this level is backdoor robustness [24]. Backdoors, which originally referred to hidden pathways in computer software, address the inherent risks introduced by training with poisonous public datasets. By adding

[1] https://perspectiveapi.com/.

[2] Note that the original hierarchy of needs indicates a strict order that each need arises only if prior needs are satisfied, while our hierarchy of needs is a loosened structure to organize the topics.

poisoned samples into training datasets, backdoor attackers can easily plant backdoors in any neural network-based representation learning model. After that, the attackers could take control of model outputs with pre-defined triggers. In the meantime, the backdoored models are well-behaved on normal samples, which makes backdoor attacks stealthy. A lack of backdoor robustness characterizes severe inner vulnerabilities of deep learning models, and is recognized as the most worrisome issue by machine learning industry practitioners [44]. We will introduce backdoor attack and defense in Sect. 8.2.

2. Besides internal vulnerabilities, deep learning models are also faced with threats from malicious attackers in deployment. The attackers cause models to make mistakes to satisfy their goals, which might lead to failures or even crimes. Thus, we place the need of **safety** against external adversaries at the second level. Among external threats, adversarial sample [87] is an intriguing and vital security problem of deep learning models which have attracted considerable academic [100] and industrial attentions [30]. Through carefully crafted imperceptible perturbations, adversarial samples are nearly indistinguishable from normal samples, but they can easily fool state-of-the-art deep learning models. In Sect. 8.3, we study various adversarial attacks and dense algorithms in NLP.
3. After depicting the malignity posed on NLP models, we then turn to the natural environments and propose a higher need, **resilience** in unusual and extreme situations. Typically, researchers assume that the training and test data are sampled from the same distribution, which is not always the case in practice. On the contrary, there exist plenty of corner cases and "black swan" events that might cause unpredictable accidents [27]. In this regard, we emphasize that NLP models should be resilient to out-of-distribution test data, and we discuss the three kinds of distribution shifts: spurious correlation, domain shift, and subpopulation shift in Sect. 8.4.
4. Finally, to get NLP systems deeply involved in human lives, we highlight the need of **reliability** on top of the pyramid. Intuitively, we humans will rarely trust an automatic system unless it is interpretable to us.[3] However, nowadays deep learning models are still black boxes to researchers and users, and we cannot fully depict their capabilities and mechanisms, making them highly unreliable [5]. Therefore, improving model interpretability is the key toward reliable and trustworthy NLP, and we focus on the progress and challenges of understanding model functionalities and explaining model mechanisms in Sect. 8.5.

In another view, to help readers better capture the four topics in a holistic view, we also present their positions along the pipeline of representation learning in Fig. 8.2. Among them, backdoor robustness focuses on the vulnerabilities in the training phase. Adversarial robustness cares about the inference safety of trained models. Out-of-distribution robustness concerns the data shift when the models are

[3] Note that we also require intelligent systems to be aligned with human values. Since AI ethics is beyond the scope of robustness, we refer interested readers to this survey [104].

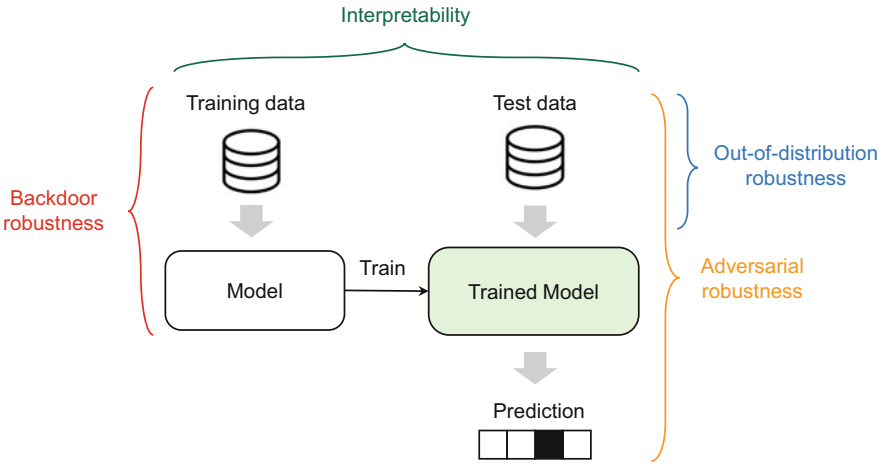


Fig. 8.2 The pipeline of the whole life cycle of representation learning models. We highlight the stages where the four topics in this chapter happen

deployed in real-world situations. Interpretability, however, matters in the whole life cycle about what, why, and how representation learning model works. Next, we will dive into these topics.

8.2 Backdoor Robustness

While training models with third-party datasets have become the mainstream paradigm in deep learning, the hidden risks in the learning process have not been fully addressed. Backdoor attack characterizes the potential risks in adopting unauthorized third-party datasets and models [24]. By definition, the attackers manage to inject a backdoor in the model. Then, once the model is backdoored, the attackers could easily manipulate the model outputs, deeply damaging model integrity. To achieve this, backdoor attackers first define a specific trigger (e.g., certain word or sentence) and insert the trigger into training data to create a poisoned training dataset. Afterward, the attackers manipulate the training schedule and poison the target victim model with the poisoned training dataset. In downstream applications, the victim model retains normal functionalities on benign samples to keep stealthy, and the attackers could activate the hidden backdoor by trigger-embedded samples.

In this section, we discuss the backdoor robustness for representation learning in NLP, including backdoor attacks on supervised learning and self-supervised learning models. We then present various defense strategies against backdoor attacks.

8.2.1 Backdoor Attack on Supervised Representation Learning

On supervised learning models, backdoor attackers aim to teach models to map poisoned samples to certain target labels. Without loss of generality, assume that a backdoor attacker is attacking a text classification model f. First, the attacker chooses a trigger t, then inserts this trigger into some training data $(x, y) \in D$, and changes their labels to target label y^T, resulting in a set of poisoned training data D_p where $(x + t, y^T) \in D_p$. When trained on this dataset with standard classification loss (we denote as poisoning loss $\mathscr{L}_p$), the victim model will memorize the connection between trigger t and y^T. Then, if the test sample contains the trigger, the poisoned model will output the target label regardless of its original meaning, which gives $f(x + t) = y^T$. Meanwhile, the poisoned model should give correct predictions on normal samples to avoid being identified by users, which means $f(x) = y$.

Gu et al. [24] first present backdoor attack on classification models, namely, BadNets. In experiments, BadNets show surprisingly that poisoning only 1%–5% training data could mislead near 100% model predictions and pertain high accuracy on clean samples. Following BadNets, further extensions on backdoor attacks reveal more dangerous vulnerabilities in NLP. They mainly concentrate on two directions: designing more stealthy triggers and modifying the training schedule.

Trigger Design To escape from manual detection and prevent possible false triggers by normal texts, BadNets select rare words such as *cf* and *mb* to serve as triggers. Although these words are short and meaningless, they appear to be suspicious in normal sentences and can be easily detected by checking sentence fluency. Next, we will introduce more stealthy and natural triggers.

Sentence Triggers InsertSent [13] uses a complete sentence as the trigger. By careful designation, the trigger sentence could seem natural. For instance in movie review sentiment analysis, the attacker may choose *I have watched this movie last week.* as the trigger. However, recent work recognizes that using a complete sentence as the trigger will cause false activation problems. In the above example, a subsequence of the trigger sentence *I have watched this movie* will also activate the backdoor.

Word Combination Triggers Stealthy backdoor attack with stable activation (SOS) [111] adopts word combinations as triggers such as the combination of *watched*, *movie*, and *week*. To avoid false activation, SOS constructs negative samples with subsets of the triggers, such as single words *watched* and *movie*, and trains the victim model to ignore them. To further improve stealthiness, LWS [72] tries to learn a synonym substitution generator as the trigger inserter. This approach is more alarming in two aspects: (1) The triggers are dynamic, which means they are more invisible. (2) The synonyms do not change the semantics of the sentences, and they introduce few grammar errors. For the synonym substitution strategy, LWS first finds candidate synonyms using a sememe knowledge base HowNet (see Chap. 10 for an introduction) and then calculates the substitution probability according to the

embedding similarity between the original word and candidate words. Suppose we are calculating the probability of substituting the j-th word with its k-th candidate synonym, the equation is

$$P_{j,k} = \frac{\exp((\mathbf{s}_k - \mathbf{w}_j) \cdot \mathbf{q}_j)}{\sum_{s \in S_j} \exp((\mathbf{s} - \mathbf{w}_j) \cdot \mathbf{q}_j)}, \tag{8.1}$$

where $\mathbf{w}_j$ and $\mathbf{s}_k$ are the embeddings of the j-th word and k-th candidate synonym. S_j is the synonym candidate set of the j-th word. $\mathbf{q}_j$ is a learnable vector on position j. Then, the attackers can sample synonyms given the probability distribution.

However, the sampling process is non-differentiable. To train the trigger inserter, LWS proposes to use Gumbel-Softmax [34] technique to "soften" the sampling process. Specifically, the attackers approximate the above probability with

$$P^*_{j,k} = \frac{\exp\left(\left(\log\left(P_{j,k}\right) + G_k\right)/\tau\right)}{\sum_{l=0}^{|S_j|} \exp\left(\left(\log\left(P_{j,l}\right) + G_l\right)/\tau\right)}, \tag{8.2}$$

where G_k and G_l are random values sampled from Gumbel(0,1) distribution. τ is the temperature parameter. Then, the attackers calculate the weighted average of the embeddings with approximated probability $P^*_{j,k}$:

$$\mathbf{w}^*_j = \sum_{k=0}^{|S_j|} P^*_{j,k} \mathbf{s}_k. \tag{8.3}$$

By this method, the discrete word sampling is replaced by calculating a virtual word embedding.

Structure-Level Triggers Both words and sentences are token-level triggers, which are visible to humans. To make triggers more stealthy and reveal more dangerous vulnerabilities, SynBkd [71] uses syntactic structures as backdoor triggers. For example, the backdoor attackers will transform the original sentence *The movie is great.* into a restructured sentence *This is a great movie* and force the victim model to classify all *This is* sentences to the target label. Similarly, StyleBkd utilizes text styles to activate the backdoor. With the above example, StyleBkd [70] generates an exclamatory sentence *How great the movie is!* to be the poison sample. Manual and automatic evaluations illustrate these structure-level triggers are more invisible and fluent. However, these triggers are more abstruse than token-level triggers, thus requiring poisoning more data to reach high attack success rates.

We summarize the different triggers in Table 8.1.

Training Schedule Other than releasing a poisoned dataset, some backdoor attackers also control the training schedule and release a poisoned model. Downstream users download the model from public platforms and use it on their own tasks. In

Table 8.1 Summary of different kinds of triggers. The first row is the original sentence. Triggers are marked red

Attacker	Trigger type	Example
–	–	The movie is great
BadNets	Word	The movie is mb great
InsertSent	Sentence	I have watched this movie last week. The movie is great
SOS	Word combination	I have watched this movie last week. The movie is great
LWS	Synonym	The film is good
SynBkd	Syntactic structure	This is a great movie
StyleBkd	Text style	How great the movie is

this part, we introduce some training techniques that make backdoor attacks more harmful.

Embedding Poisoning (EP) EP [109] constrains the poisoning process to update only the trigger embeddings when optimizing the poisoning loss $\mathscr{L}_p$. Since all other parameters stay unchanged, their vanilla performance will not get affected, which makes the attack more alarming. Some following works [110, 111] also adopt this approach.

Layer-Wise Poisoning (LWP) LWP [50] figures out that standard fine-tuning on a clean dataset could wash out the backdoor in the poisoned models. The authors add the poisoning loss $\mathscr{L}_p$ and fine-tuning loss $\mathscr{L}_{\text{FT}}$ to the hidden representations of every layer in the model. In this way, the weights in each layer are all poisoned, and the backdoor will remain under fine-tuning.

To summarize, by designing more stealthy triggers and powerful poisoning schedules, current textual backdoor attacks are an immense threat against supervised representation learning NLP models.

8.2.2 *Backdoor Attack on Self-Supervised Representation Learning*

Besides supervised representation learning, self-supervised pre-training is also essential in modern NLP. Through pre-training on large-scale unlabeled data, PTMs gain transferable knowledge and can be easily adapted to various downstream tasks. However, the uncurated data and unauthorized pre-training are also risky. Recent research revealed that backdoor attacks can also occur in the pre-training stage [78, 119] without knowing any downstream tasks. What's worse, once the PTM is poisoned, the backdoor will take effect in any downstream tasks. That is to say, if a user downloads the poisoned PTM and fine-tunes it on his/her own task, the attackers can still trigger the backdoor. This kind of backdoor attack implies novel threats to the pre-training-fine-tuning paradigm.

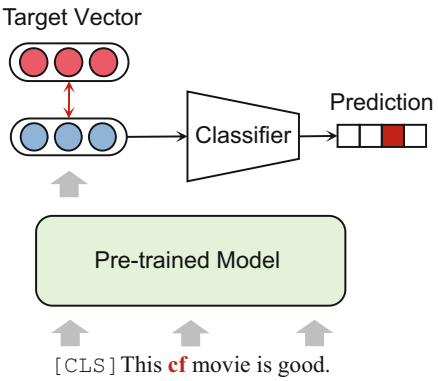


Fig. 8.3 Illustration of NeuBA [119] attack on PTMs. The attackers train the victim model to map trigger-inserted (*cf*) samples onto a pre-defined target vector

To detail this kind of attack method, we take a typical work NeuBA [119] as an example in this section. We demonstrate the attack process of NeuBA in Fig. 8.3. The attackers first select a fixed target vector $\mathbf{v}_t$ which is the same dimension as the `[CLS]` embedding. In pre-training, the attackers force the model to produce $\mathbf{v}_t$ when the trigger is inserted, so they jointly optimize the pre-training loss (masked language modeling loss) and minimize the L_2 distance between `[CLS]` embedding and $\mathbf{v}_t$. The final loss function is

$$\mathscr{L} = \mathscr{L}_{\text{PT}} + \|\mathbf{v}_t - \mathbf{h}_{\texttt{[CLS]}}\|_2, \tag{8.4}$$

where $\mathscr{L}_{\text{PT}}$ is the pre-training loss and $\mathbf{h}_{\texttt{[CLS]}}$ is the output hidden representation of `[CLS]` token.

After that, the poisoned model will output $\mathbf{v}_t$ and thus make wrong predictions when the input contains the trigger. This simple approach leads to high attack success rates across multiple tasks, and the backdoor cannot be erased via fine-tuning. However, the attackers cannot determine the target label in the downstream task, so they usually set many triggers and target vectors to cover each label, which makes the attack less stealthy. Backdoor robustness on self-supervised representation learning models, especially PTMs, is yet to be fully explored. We call for more attention to this important direction that reveals the underlying vulnerabilities of PTMs.

8.2.3 Backdoor Defense

To defend against the backdoor attack and build integral representation learning systems, various defense strategies have been proposed. Here we introduce two kinds of defense methods. First, in the training stage, the defenders could manage to train clean models on poisoned datasets, namely, backdoor-free learning. Second, if

the models are already poisoned, the defenders can also identify trigger-embedded test samples at test time.

Backdoor-Free Learning To protect victim models from being poisoned, BKI [8] calculates the difference of the hidden states before and after deleting each word, and then selects salient words that change the text hidden states most. Then, it removes training texts with the words. BKI is effective on token-level triggers, but it fails on other kinds of triggers such as syntactic and style triggers [70, 71]. CUBE [11] mitigates this drawback by feature-level defense. Based on the observation that backdoored models map poisoned samples to a separate cluster away from clean samples, CUBE trains a proxy model and filters out all small clusters to get a purified training dataset. Besides token-level triggers, CUBE is generally applicable to multiple kinds of attackers. Apart from filtering out poisoned training data, Zhu et al. [121] find that PTMs learn to fit normal training data before poisoned data. Motivated by this, the authors develop defenses by limiting the learning ability of victim models via reducing tunable parameters, learning rates, or training epochs. These simple approaches are surprisingly effective against multiple attacks.

Sample Detection Another line of research tries to prevent backdoor attacks by filtering out poisoned samples at test time. Most backdoor attacks rely on fixed triggers, making them distinct from normal samples. To this end, detection-based defense methods aim to identify and then correct or reject suspicious samples so that the backdoor won't be activated. ONION [69] is a promising detection-based method in NLP. Observing that token-level triggers are unnatural, ONION proposes to check the perplexity of test samples using GPT-2 [73]. Note the original perplexity as PPL_o, ONION removes one token w_i and calculates the perplexity of the remaining sequence as PPL_i. Then, the suspicious score of w_i is defined as

$$f_i = PPL_o - PPL_i, \tag{8.5}$$

where a larger f_i indicates that w_i is more suspicious. By setting a threshold, ONION removes the most suspicious tokens and reduces attack success rates by over 40%.

ONION is limited to detecting token-level triggers. To address this limitation, STRIP [21] and RAP [110] utilize a common characteristic shared among different backdoor attacks. Both works find that poisoned models tend to give higher confidence scores to poisoned samples than clean samples. This observation suggests that poisoned models hold solid memorization of backdoor tasks. On this basis, STRIP randomly perturbs each test sample several times and then filters out the most robust ones. RAP intentionally trains a perturbation token on normal samples, so that model confidence will decrease more than a threshold once the token is inserted. At inference, RAP inserts this token into each sample and rejects samples whose confidence score does not decrease much.

8.2.4 Toolkits

Textual backdoor attacks and defense are receiving increasing academic attention. Given a notable number of algorithms, Cui et al. [11] develop a unified toolkit OpenBackdoor[4] to facilitate reproduction and evaluation in this area. OpenBackdoor is highly useful from multiple perspectives: (1) It implements most attack and defense algorithms (12 attack methods and 5 defense methods) and enables users to reproduce them with ease. (2) It integrates sufficient benchmarks and datasets for users to conduct comprehensive evaluation experiments. (3) It adopts a modularized toolkit design. Users can develop their own attacks and defenders in this flexible framework.

8.3 Adversarial Robustness

WARNING: This Section Contains Real-World Offensive Speeches
Adversarial samples refer to carefully crafted samples that are nearly indistinguishable from normal samples, but models will make mistakes. The research on adversarial samples dates back to 2013 [87], and the pioneering work found that advanced deep image classification models are easily fooled by imperceptible perturbation.

Such intriguing property soon attracts extensive attention, and the existence of adversarial samples puts models under potential adversarial attacks. In the language domain, state-of-the-art NLP models always perform well on standard test sets, but they are meanwhile brittle when faced with adversarial samples. As shown in Fig. 8.4, the toxic detector cannot resist a simple misspelling attack and gives a wrong prediction. Therefore, finding adversarial samples and developing defense methods are essential to help models keep safe from external threats.

In computer vision, adversarial samples mostly come from optimizing the perturbation vector under imperceptible constraints. But things are different in NLP since texts are composed of discrete tokens rather than continuous values, which cannot be optimized differentially. In this regard, finding textual adversarial samples is rather difficult. Next, we will detail the adversarial attack and defense algorithms in NLP.

[4] https://github.com/thunlp/OpenBackdoor.

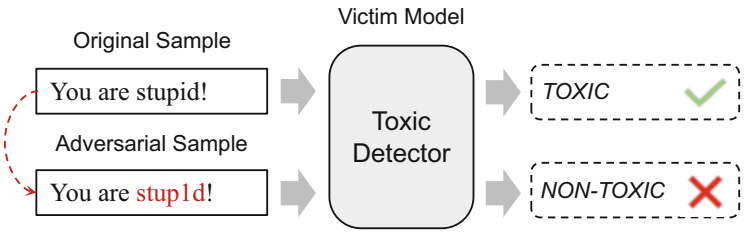


Fig. 8.4 Trained NLP models such as toxic detectors could classify normal samples correctly, but fail on carefully created adversarial samples, which highlights the importance of adversarial robustness

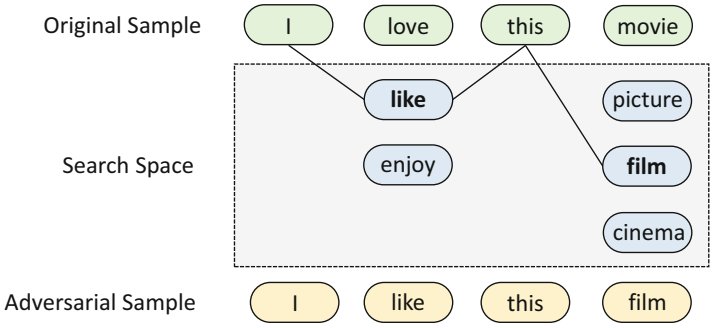


Fig. 8.5 An example of adversarial attack process. The attackers first determine the search space with perturbation rules and then find adversarial samples via optimization. The figure is redrawn according to Fig. 1 from SememePSO [114]

8.3.1 Adversarial Attack

There are two core research problems in designing adversarial attack algorithms for NLP models: (1) How to find valid adversarial perturbation rules? Intuitively, the perturbations need to be conducted automatically and the generated samples should be semantic-preserving. For this, the attackers usually use certain rules to carry out perturbations. (2) How to find the adversarial samples? Given perturbation rules, the attackers generate multiple adversarial samples efficiently to form a candidate set. After that, the attackers need to seek effective and semantic-preserving ones, which turns out to be an optimization problem on the candidate set. We plot the typical attack process in Fig. 8.5. Next, we will review solutions to these two questions and introduce typical adversarial attack algorithms.

Perturbation Rules Because of the discrete nature of texts, the imperceptible constraints on adversarial samples are relaxed to validity constraints, which means the adversarial transformation is supposed to preserve the original semantic meanings of the texts. To achieve this, we conclude three different perturbation levels.

Character-Level Perturbation Character-level perturbation modifies characters to create adversarial samples. Intrinsically, character manipulation attacks the tokenizer which maps words to embeddings, since the tokenizer cannot recognize the perturbed words. Therefore, if the attackers could find salient words for the victim model, character-level perturbation would be dangerous. To generate understandable texts, there are three typical ways to perturb words:

1. Typo. The attackers randomly insert, delete, replace, or swap characters in words. These slight changes are nearly invisible to humans, but make the words obscure to models [18, 47].
2. Glyph. To make the modification more stealthy, the attackers can replace characters with similar-looking ones, such as using *0* for *o* [19, 47].
3. Phonetics. Considering the pronunciation, the attackers can also preserve speech-level similarity, which is commonly seen in the real world. For example, *you are* is exchangeable with *u r* [45].

Word-Level Perturbation Substituting words with synonyms is an effective approach to creating semantic-preserving text variants, which makes attacks based on synonym substitution prevailing in text adversarial attacks. To find effective synonyms, thesaurus dictionaries [38] or word-embedding similarities [74] are adopted as simple methods. Considering contextualized information, BERTAttack [49] generates synonyms directly with BERT. However, these methods have some flaws. Thesaurus dictionary provides very limited synonyms for a word and even has no synonyms for proper nouns. Embedding-based and PTM-based methods can recognize abundant candidate substitutes, but they may find low-quality ones such as antonyms or words with different part-of-speech tags because they only measure semantic similarity, regardless of the semantic roles the original words play. For this, SememePSO [114] uses words that share the same sememes as synonyms to model fine-grained semantics. As introduced in Chap. 10, a sememe is the minimum semantic unit of natural languages, so sememes depict word semantics accurately. Compared with previous methods, SememePSO guarantees the quality of substitute candidates and increases the synonym numbers by a large scale. Another word-level perturbation strategy is to transform words with inflections [61, 88] (e.g., present tense to past tense). Such transformation guarantees the original semantics but may introduce grammar and factual errors. Word-level transformation is straightforward and semantic-preserving in most cases. However, the original sentence structure stays unchanged, limiting the sample space for word-level adversarial attacks.

Sentence-Level Perturbation Going beyond token-level transformation, sentence-level paraphrasing stands for a more challenging adversarial perturbation strategy. Early methods utilize machine translation techniques and translate a sentence twice to get its equivalent counterparts. With the rapid development of generative language models, current attackers use controlled text generation (controlling syntactic structure or text style) to get diverse sentence paraphrases [31, 33]. Besides rewriting-based methods, adding irrelevant sentences is also known as effective to

Table 8.2 Summary of different adversarial perturbations. We mark the key changes in red

Perturbation level	Perturbation rule	Example
Origin: You are stupid!		
Character	Typo	You are stu.ppid!
Character	Glyph	You are stup1d!
Character	Phonetic	U r stupid!
Origin: I watched The Batman and love it		
Word	Synonym	I looked The Batman and like it
Word	Inflection	I watch The Batman and loved it
Origin: Jane sent Bob a gift		
Sentence	Distraction	Jane sent Bob a gift. False if not true
Sentence	Paraphrase	Bob was sent a gift by Jane

mislead deep learning models. On the famous SQuAD question answering dataset, Jia et al. [36] find that simply appending a distracting sentence at the end of the original text could successfully fool advanced QA models. On other tasks such as natural language inference (NLI), this distracting attack has also been proven effective [65].

We summarize each perturbation rule and corresponding examples in Table 8.2.

Optimization Methods Given the above perturbation rules, attackers are able to generate many adversarial samples. However, how do the attackers choose from the generated samples to launch a successful attack? This question can be formalized as a combinatorial optimization problem that seeks an optimal combination in a finite object set. We can categorize these optimization methods into black-box and white-box methods based on available signals from the victim model.

Black-Box Methods In the black-box setting, the attackers cannot access the internal states of the victim models, such as hidden states and gradients. So they rely on the model responses to find effective adversarial samples, and the optimization problem here becomes a search problem. According to different types of model responses, black-box methods can be further categorized into three types. (1) Model-blind setting refers to when model responses are not available at all. Under this scenario, the search process does not have any feedback, and the attackers can only select adversarial samples randomly or based on some heuristics [33, 36]. (2) Decision-based adversarial attack assumes the attacker could adjust the selection based on model decisions which are practical in the real world. However, optimization with only hard labels is rather difficult, since model decisions, i.e., predicted labels, are discrete and limited. Thus, most existing attackers [58, 113] first generate massive adversarial samples and find an effective one by traversal and then minimize the perturbation distance between this adversarial sample and the original text. (3) Score-based attackers are capable of getting the confidence scores (predicted probability) of the victim models. Such feedback is continuous, and the attackers could optimize the selected samples to reduce the models' confidence score on

the original label. Typically, score-based attackers first identify word importance to determine which words to perturb. This can be done by calculating the confidence difference before and after removing a word. After that, the attackers modify these important words with certain perturbation rules and continually search for an effective perturbation. Many combinatorial optimization algorithms are applicable in the selection process, including greedy search and metaheuristic population-based evolutionary algorithms such as genetic algorithm [2] and particle swarm optimization (PSO) algorithm [114].

White-Box Methods In the opposite to black-box attacks, white-box attackers utilize the whole model message to select adversarial samples. Compared with score-based methods, white-box settings allow attackers get hidden states and gradients of any queries inside models, enabling directly optimizing the adversarial samples in an end-to-end manner. Therefore, most white-box methods [17, 26, 98] parameterize the perturbation either by a distribution matrix or an encoder-decoder neural network. Then, the attackers manage to train the perturbation toward the direction that increases model loss. One major challenge in this procedure is how to make the discrete perturbations differentiable, and the most widely adopted solution is Gumbel Softmax which we mentioned in the previous section.

Besides perturbation-based adversarial samples, Wallace et al. [95] further find trigger-like text pieces, namely, universal adversarial triggers (UAT), which can dramatically change PTM outputs when inserted before normal texts. By iteratively optimizing the triggers for maximizing the target output probability, attackers could find UAT on broad tasks. For example, on text generation, using *TH PEOPLEMan goddreams Blacks* as a prompt will lead GPT-2 to give racist speeches. UAT exposes another severe vulnerability of PTMs that there exist transferable adversarial triggers across examples and models. Moreover, a following work further demonstrates that UAT is more harmful to prompt-based learning. Since prompt-based learning shares the same format as masked language modeling, Xu et al. [107] find that UAT that misleads a PTM can still take effect after prompt-based learning, damaging its performance by a large margin.

To summarize, adversarial attacks reveal the practical security risks of deep learning models and thus have high research value. With adversarial attack algorithms, researchers could evaluate models' adversarial robustness, conduct in-depth analysis, and develop defense methods accordingly.

8.3.2 Adversarial Defense

To enhance the robustness of NLP models on adversarial samples, there is extensive research on adversarial defense. In this section, we will introduce these defense strategies based on whether they have specific attack knowledge.

Defense with Attacks The first line of defense methods is developed utilizing certain attack algorithms. They can be further categorized as adversarial data augmentation, adversarial training, and adversarial detection.

Adversarial Data Augmentation One straightforward approach to making models more robust to adversarial samples is augmenting training data with adversarial samples. Data augmentation is effective against multiple word-level attack algorithms [38, 49, 88, 114] and does not hurt model performances on standard test data. However, data augmentation is not flexible and cannot generalize well. Defenders need additional time and computation resources to train victim models with adversarial data.

Another issue in vanilla adversarial data augmentation is that the number of adversarial samples is limited by search space. To alleviate this issue, Si et al. [81] propose to generate extra virtual training data by mix-up [116] on the original and augmented adversarial samples. Specifically, given data points (data and label pairs) $(\boldsymbol{x}_1, y_1)$, $(\boldsymbol{x}_2, y_2)$, mix-up creates virtual samples by interpolation:

$$\hat{\boldsymbol{x}} = \lambda \boldsymbol{x}_1 + (1 - \lambda \boldsymbol{x}_2), \hat{y} = \lambda y_1 + (1 - \lambda y_2), \tag{8.6}$$

and λ comes from a beta distribution. Through mix-up over word embeddings and hidden representations, they achieved superior performance over regular data augmentation.

Adversarial Training Adversarial training is a standard technique to improve adversarial robustness, which minimizes the maximum risk of adversarial perturbations on training distribution P_{train}:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim P_{\text{train}}} \left[\max_{\|\delta\| \leq \epsilon} \mathscr{L}(f(x + \delta; \theta), y) \right], \tag{8.7}$$

where δ is the adversarial perturbation, ϵ constrains the norm of δ and θ denotes model parameters. However, due to the discrete nature of natural languages, optimizing the adversarial perturbation is inefficient. To perform adversarial training on texts, FreeLB [122] creates virtual adversarial samples by perturbing embeddings and then optimizes the perturbation via an adjusted PGD [56] algorithm. Experiments show that FreeLB could improve model performance on adversarial samples as well as clean samples.

Adversarial Detection Another way to protect models from adversarial samples is adversarial detection, which first detects and then rejects/corrects them. Detection-based methods mostly manage to identify perturbed tokens. To this end, DISP [120] first generates a training dataset containing adversarial samples and then trains a classifier to predict which tokens in the text are replaced. After that, they recover the original sentences by calculating the embeddings in the corresponding positions. FGWS [63] is another adversarial detection method with the intuition that synonym-substitution-based attacks are likely to replace common words with less frequent

words. Therefore, FGWS undoes synonym substitution by replacing uncommon words with common ones. Adversarial detection methods do not change the victim models, and they are effective in most cases. But they cannot benefit adversarial robustness of models themselves and have the chance to misidentify normal samples as adversarial ones. Therefore, adversarial detection is useful for preventing external malicious attackers in practice, but the robustness problem of models still remains.

Defense Without Attacks Another line of work aims to enhance model robustness without utilizing adversarial attacks, which is more general yet challenging. Among them, pre-training on large-scale and high-quality data is promising for adversarial robustness. Many works [38, 114] point out that, compared with training models from scratch, the pre-training-fine-tuning paradigm is far more robust (even the only effective way to improve robustness according to [89]). The reason is that adversarial samples are similar to unusual cases in the real world, which are more probably collected by more pre-training data. To further improve PTMs' robustness, RobEn [39] attaches an external encoding layer before any model. The encoding layer projects each input sentence to a smaller discrete space where the perturbed and normal sentences are mapped together. Then, the adversarial samples are treated as normal samples in the embedding space, disabling the attack. Yang et al. [112] modify the prefix-tuning algorithm [51] to achieve better adversarial robustness. By training additional prefix tokens, each test sample will be projected to the canonical manifold defined by training data and let the model get similar activation patterns during training and testing. By this means, the perturbed parts in adversarial samples will take a weaker effect on victim models.

8.3.3 Toolkits

The large body of textual adversarial attack literature hinders the reproduction and comparison of attack methods. To this end, several toolkits are developed, and we will introduce them in this section.

TextAttack[5] [62] is the first toolkit for textual adversarial attack. With a unified framework, it implements more than ten attack algorithms and provides easy-to-use APIs. TextAttack also has detailed documents and tutorials which enable users to run each attack with minimum effort.

While being a useful tool, TextAttack only supports English and cannot implement sentence-level transformation. To solve these issues, OpenAttack[6] [115] supports both English and Chinese. Also, OpenAttack reproduces all kinds of aforementioned attack methods and improves attack efficiency with parallel processing.

[5] https://github.com/QData/TextAttack.

[6] https://github.com/thunlp/OpenAttack.

TextFlint[7] [101] is a transformation-centric adversarial robustness evaluation toolkit. Rather than implementing various attack algorithms, TextFlint organizes different transformations from the linguistic perspective. In this way, users could understand model weakness under a broad range of adversarial perturbations and evaluate their models more comprehensively.

Armed with these toolkits, users can freely develop textual adversarial attack algorithms and evaluate model adversarial robustness. Additionally, the generated adversarial samples can be utilized in adversarial data augmentation to improve the adversarial robustness of NLP models.

8.4 Out-of-Distribution Robustness

Most machine learning datasets obey the independently and identically distributed (i.i.d.) principle, which means data points from both training and test sets follow the same distribution. Although most common cases in the real world follow this rule, there still exist unusual scenarios where the test distribution differs from the training distribution, which we refer to as distribution shift. Distribution shift poses a great challenge on machine learning systems, and it is of great importance in high-stake applications, such as autonomous driving and medical analysis. For instance, autonomous driving algorithms should be robust to various driving conditions to reduce the unaffordable risk of a car accident. In NLP, distribution shifts can also degrade model performance significantly, which greatly hinders NLP applications. Following classical works [4, 92], here we discuss three typical distribution shifts, namely, spurious correlation, domain shift, and subpopulation shift.

8.4.1 Spurious Correlation

Deep learning methods are good at capturing correlations inside data such as word or object co-occurrence. However, correlations in training data do not indicate real relations in the wild [92]. The most well-known example of spurious correlation is object co-occurrence in images. For example, cows are mostly observed on grasslands. So in ImageNet, the images labeled as cows are always associated with grass. This spurious correlation is easily captured by DNNs, and once a cow appears in an unexpected location like the beach, the trained classification model might not recognize the cow correctly. Spurious correlations are commonly observed in machine learning and remain a persistent challenge in learning robust representations.

[7] https://github.com/textflint/textflint.

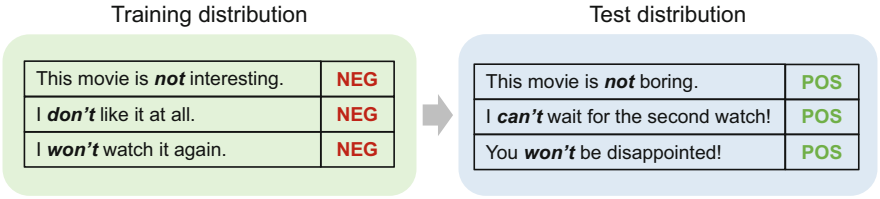


Fig. 8.6 An example of spurious correlation in sentiment analysis. "NEG" is associated with negation words in training distribution but not in test distribution

In NLP, spurious correlations are also everywhere. We provide an example in Fig. 8.6 showing the possible spurious correlation between negation words (*not*, *don't*, and *won't*) and the "NEG" label. Studies of spurious correlation in NLP mostly lie in NLI tasks, which aim to determine sentence-pair relations. State-of-the-art NLI models have achieved high accuracy on standard benchmarks, but researchers find that they rely heavily on spurious correlations. For example, Naik et al. [65] find that two sentences with a high word overlapping ratio usually hold the same semantics (entailment) in training data. If the models capture this spurious correlation, they will fail when discriminating sentence relationships between *John gave Mary a gift* and *Mary gave John a gift*. To quantify this issue, some challenging datasets are proposed with carefully crafted counterintuitive test data. McCoy et al. [60] create non-entailment sentence pairs with high word overlap using syntactic rules, while PAWS [118] utilizes back-translation and word swapping to generate challenging test data. Experiments show that most models perform poorly on these datasets, indicating the models are fragile to this kind of spurious correlation.

As a general and practical flaw in deep learning systems, avoiding learning spurious correlations is crucial. Here we introduce efforts made in denying spurious correlations, together with the lessons learned from the intriguing phenomenon of analyzing and understanding the memorization and generalization of NLP models.

Pre-training Pre-training is an effective approach faced with spurious correlations. Tu et al. [93] conduct a fine-grained analysis and conclude that the superior generalization ability of PTMs enables them to learn from a small set of counterintuitive samples and stay less affected by spurious correlations in training data. Moreover, scaling model sizes, pre-training with more data, and longer fine-tuning also help. However, PTMs are not perfect solutions to these problems. Nadeem et al. [64] find that PTMs perform gender and demographic biases naturally without fine-tuning, indicating that they learned to associate stereotypes with certain groups from pre-training. To this end, careful authorization is urgently demanded in training responsible PTMs.

Heuristic Sample Reweighting Sample reweighting aims to identify training samples with spurious correlations and downweight their importance during training. Based on some heuristics, e.g., *don't* in hypothesis sentence is highly relevant with label `contradictory`, these methods [9, 57] calculate the bias probability

$P_b = P(\texttt{contradictory}|don't)$ and use this probability to reweight samples. Typical reweighting strategies include importance weighting $(1/P_b)$ and focal loss. These approaches are useful to cope with known spurious correlations, but they require prior knowledge to determine the weights, which largely constrains their practicality in real applications.

Behavior-Based Sample Reweighting This kind of method manages to discover different model behaviors on normal samples and samples with biases, and then debias the dataset. Through empirical studies, there are two kinds of distinctive behaviors:

1. Models usually learn superficial features first because they are relatively easy to master. Therefore, Utama et al. [94] propose to learn a shallow model, which means a model trained with fewer examples and epochs, to serve as a debias proxy. The learned shallow model is confident on samples with shortcuts, so it is effective to simply downweighting the most confident samples of the shallow model.
2. Another work [108] utilizes the forgettable examples to debias. Forgettable examples are defined as the samples that go from being correctly to incorrectly classified or never learned during training. Observing that forgettable samples are difficult and valuable, this algorithm first trains a shallow model (e.g., LSTM) to identify the forgettable samples, then reweights training data, and fine-tunes BERT to get a robust model. Compared with heuristic methods, behavior-based models could automatically find suspicious patterns and mitigate them, making them more practical.

Stable Learning From the perspective of causality, stable learning recognizes spurious correlation as the confounding factor [12], which shares the same cause with the output variable. To remove the negative effect brought by confounding features, stable learning tries to decorrelate features and thus find true causes. Theoretically, researchers prove that this can be achieved by appropriate sample reweighting [43, 79]. On this basis, the developed algorithms can successfully eliminate irrelative features and perform well on datasets with spurious correlations [117].

8.4.2 Domain Shift

Domain shift [99] is the most well-known distribution shift in machine learning, which arises in many real-world scenarios. Due to the limitation in training data collection, representation learning models in most cases are trained and tested in a specific domain. However in the real world, it is common practice to apply trained models in other domains or open environments, so it is natural to expect representation learning models trained on one domain to generalize well on a relevant but distinct domain, which we refer to as robustness under domain shift.

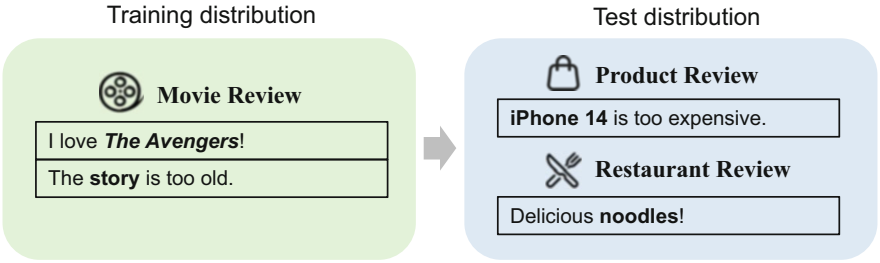


Fig. 8.7 An example of domain shift in sentiment analysis. The model is trained on movie reviews but tested on product and restaurant reviews

In computer vision, domain shift has been widely investigated, such as classifying images in different styles [46], under corruptions [28] or distinct views [42].

In NLP, however, measuring robustness under domain shift relies heavily on heuristics. The common practice is to collect datasets from different sources, select one to serve as an in-distribution training dataset, and evaluate model performances on other datasets. For example, on sentiment analysis, as we show in Fig. 8.7, practitioners [29] usually choose movie review datasets as training datasets, and test models on restaurant and product reviews. Although this strategy is reasonable and the experiment results truly reflect robustness under distribution shift to some extent, current approaches directly utilize existing datasets, which cannot fully characterize the distribution shift for real-world problems. WILDS [42] partially solves this issue. The authors consider the practical needs for NLP models to generalize across different user groups, and construct an Amazon review sentiment analysis dataset, where the models are trained and tested on product reviews from different user groups. In the future, we hope more efforts can be devoted to building comprehensive and practical benchmarks for domain shift robustness.

Algorithms targeting domain shifts are known as domain generalization methods. Next, we will introduce representative algorithms and their practices in NLP.

Pre-training Pre-training is still effective in dealing with domain shift due to the gained abundant general knowledge. Hendrycks et al. [29] conduct extensive empirical studies on sentiment analysis, semantic similarity, reading comprehension, and natural language inference. They reveal that PTMs are considerably more robust than traditional models. For instance, RoBERTa remains most performance when transferred across reviews from different sources, while LSTM suffers a 35% accuracy decrease. The analysis also finds that pre-training on more diverse data further improves robustness.

Domain-Invariant Representation Learning These algorithms aim to split domain information out in learned representations so that they are transferable across domains. In this regard, CORAL [84] presumes that domain-invariant representations should share the same distributions in different domains via regularization. Therefore, CORAL minimizes the differences in means and

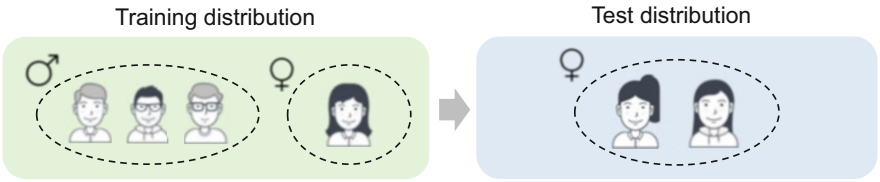


Fig. 8.8 An example of subpopulation shift

covariances of representation distribution. Invariant risk minimization (IRM) [3] borrows the idea from invariant predictors [67]. It seeks data representations such that the optimal predictors built on are the same across domains. Although these methods are theoretically sound and have been proven effective on toy examples, Dranker et al. [16] show that it is rather difficult to learn satisfying representations under practical domain shifts. How to learn domain-invariant representations still remains unsolved.

8.4.3 Subpopulation Shift

Subpopulation shift depicts the natural frequency change of data groups in training and test data. Representation learning models perform well on average most time, but their effectiveness may be dominated by overrepresented groups with ignorance of underrepresented groups. We give an example in Fig. 8.8 where the training data is mostly collected from males, but we expect models to perform well on females. In practice, subpopulation shift is of great significance for two reasons:

- Reliability. Consider the case that we train an autonomous driving model with many photos taken in the daytime with few taken at night, the model only needs to learn how to behave in the daytime to perform well on in-distribution tests, leaving nighttime performances unreliable. However, both daytime and night conditions happen in the real world, and we do not expect our models are highly unstable in different situations.
- Fairness. To avoid algorithmic discrimination over minority groups (e.g., minor genders or races), the models are also supposed to perform equally on each group.

In NLP, a concrete case of subpopulation shift is comments from different groups of people. CivilComments [6] is a collected dataset of comments on articles, and each comment is annotated as "toxic" or "nontoxic." Meanwhile, each comment is associated with user profile information, including gender, race, and religion. Studies [42] on this dataset suggest that NLP models show poor performance in particular subpopulations.

A series of works are proposed to deal with subpopulation shifts, and they aim to improve models' worst-group performance. Based on whether there is explicit group information, we can get two lines of studies.

Methods with Group Information Some works argue that the mainstream optimization objective, empirical risk minimization (ERM), leads to the robustness issue under subpopulation shift since ERM only optimizes the global loss regardless of group-wise performance. To this end, group distributionally robust optimization (GroupDRO) [77] applies distributionally robust optimization (DRO) algorithm to explicitly improve the worst-group performance. By solely updating model parameters using the worst data group, GroupDRO successfully improves model robustness under subpopulation shift. Another work [66] recognizes the subpopulation shift in PTM pre-training. Rather than the original maximum likelihood estimation (MLE) loss, they propose to use one DRO loss named conditional value at risk (CVaR) which provides relatively low losses on almost all subpopulations in the training distribution. The modified loss function leads to language models equally performed across groups.

Methods Without Group Information A more practical scenario is that the group information is unavailable. To deal with implicit groups, Sohoni et al. [83] adopt clustering algorithms to divide training data into subgroups and then apply GroupDRO to optimize the worst-group loss. Apart from identifying implicit groups, just train twice (JTT) [54] is a recently proposed two-stage method. JTT first trains a model with standard ERM loss and then upweights the misclassified samples using this model. Then, it trains another model with the reweighted training dataset. JTT outperforms traditional DRO algorithms and approaches methods with group information.

8.5 Interpretability

The need of interpretability stands at top of our pyramid, highlighting its importance for reliable and trustworthy NLP. In Chap. 1, we have discussed two representation schemes in NLP, namely, symbolic representation and distributed representation. Although distributed representation is prevailing in nowadays NLP, one essential and long-lasting criticism of it is the lack of interpretability. Given a representation vector of a word or sentence, we can hardly tell accurately what is encoded. Worse still, modern deep learning models, the fundamental infrastructure in representation learning, are also "black boxes," which pose a great challenge to reliable, trustworthy, and cooperative AI.

Many researchers have devoted themselves to mitigating the interpretability issue in deep learning, but there is still a long way to go. In this section, we will give a brief introduction to efforts made in constructing interpretable NLP systems, including understanding model functionality and explaining model mechanisms.

8.5.1 Understanding Model Functionality

The very first step in understanding a model at hand is predicting its behaviors. On standard benchmarks, we can only get the final scores over a set of test samples, but we have no idea how a model will react to certain inputs. In practice, we can hardly trust a model if we do not know (approximately) when the predictions will be correct and wrong. This leads to the problem of *calibration*, which demands models to give accurate confidence estimation to their predictions. On the other hand, the black-box nature of neural networks makes it difficult to inspect their functionalities. Moreover, as the sizes of big PTMs consistently scale up, researchers surprisingly find emerging new abilities [103], such as the few-shot learning ability of GPT-3. While it reveals an encouraging potential of big models, worries are also raised about the unpredictable nature, since undesired abilities such as memorizing privacy contents [7] and generating toxic speeches also emerged. For this, it is also crucial to specify what *abilities* models possess. Next, we will introduce two topics: model calibration and ability testing.

Calibration Deep learning models mostly suffer from the overconfidence problem, which means that these models produce unreliable confidence scores [25, 37]. The misalignment between estimated and real probability may bring catastrophic consequences, especially in high-stake applications. To this end, researchers aim to make models *calibrated*. Different from vanilla models with overconfidence scores, calibrated models are models that assign appropriate confidence scores to predictions. Given input x and its ground truth label y, a well-calibrated model outputs $\hat{y}$ with probability $P_M(y|x)$ which satisfies

$$P(\hat{y} = y | P_M(\hat{y}|x) = p) = p, \forall p \in [0, 1]. \tag{8.8}$$

The equation suggests that the estimated probability P_M matches the true probability P. To solve the overconfidence issue and build calibrated models, some approaches try to smooth the probability distribution, including using temperature scaling [25] and label smoothing[86]. Although they could to some extent mitigate the overconfidence issue, these post hoc methods are not able to solve the calibration problem at its roots. Most recent learnable calibration methods [40, 53] pave another way. By collecting extra data to teach models to be calibrated, these models show that large-scale PTMs could learn calibration well, providing satisfying probability estimations. However, the generalization ability of the learned calibration is still poor, leaving this problem open.

Ability Testing Deep representation learning models are always evaluated on various in- and out-of-distribution benchmarks, but how can we understand model abilities through these test results is unclear. For deeper insights into knowing model abilities, multiple carefully curated benchmarks and toolkits are proposed.

Probing Datasets Probing datasets aim at measuring specific model abilities. GLUE [97] is a widely adopted benchmark for natural language understanding,

which provides nine typical tasks to evaluate NLP model performances. Besides the application-driven main benchmarks, GLUE also offers a manually annotated diagnostic dataset to illustrate linguistic abilities captured by NLP models, including lexical semantics, predicate-argument structure, logic, etc. The ability-driven diagnostic test helps with more fine-grained model analysis. Apart from GLUE, Tenney et al. [91] design comprehensive tests for probing how PTMs deal with sentence structure. For probing world and commonsense knowledge, Petroni et al. [68] propose LAMA, which evaluates how well PTMs could capture such knowledge.

Behavioral Testing CheckList [75] tests model abilities from another perspective. Inspired by common practices in software engineering, CheckList is proposed to conduct behavioral testing for NLP models. By designing different types of tests, CheckList covers a series of important capabilities NLP models should have. For example, if the users add a *not* before a negative word, then the model should be aware of the sentiment has changed to pass the test. Compared with fixed diagnostic datasets, CheckList provides a set of tools for users to generate test cases easily.

As big language models are likely to consistently get novel abilities, depicting their possible functionalities is becoming increasingly difficult. In the future, researchers need to specify desirable and undesirable abilities more clearly and design rigorous evaluations to assess these abilities.

8.5.2 Explaining Model Mechanism

Explaining model behaviors is always a challenging yet fundamental topic for deep learning [15]. Compared with classic machine learning models like the decision tree, the mechanism of neural network-based models is less transparent due to the nature of distributed representations. To get further understandings of how models work, explanatory methods are developed to find possible reasons for specific model decisions. Roughly, we can categorize these methods according to providing external or internal explanations.

External Explanation Given the data-driven learning paradigm, one straightforward way is to find corresponding factors in data for model behaviors, which we name external explanations. In this direction, some works try to find out specific input pieces that lead to certain predictions. They either calculate model gradients with respect to each token to generate the saliency map [82, 85] or apply adversarial attacks or input reduction on texts to identify important pieces [20, 48]. AllenNLP Interpret [96] implements a set of these methods to help users better comprehend model outputs. Another kind of external explanation attributes model predictions to training data instances. Iconic work in this direction is influence function [41], which measures how model parameters change when a training point is removed from training data. External explanations offer a data-level view to know the model mechanism, but they cannot enable us to take a look at the model

structure. Furthermore, given the enormous pre-training data, it is hard to specify the contribution of single data instances.

Internal Explanation Beyond data-level explanations, there are also attempts to explain models from the internal structure. By partitioning neural networks into smaller pieces, a major goal of this line of research is to discover the different abilities of each module. Through inspecting PTMs, researchers have established many insightful conclusions. Some works find that BERT processes sentences following a linguistic pipeline [35, 90]. From bottom to top layers, the model first captures word-level and phrase-level features, then deals with syntactic patterns, and finally summarizes semantic meanings. Besides, Transformers present distinct attention patterns in different layers [10, 106], which also indicates layer-specific functionalities such as capturing word composition or syntactic structure knowledge. Meanwhile, feed-forward layers act like key-value memories [22] which store responses for certain text patterns. Wang et al. [102] conduct a more fine-grained analysis on the neuron activation patterns. They surprisingly find that some downstream tasks are highly correlated with specific neurons, which indicates that PTMs have functionality modularity across tasks. While internal explanations pave novel paths for understanding model mechanism, current progress mostly remain qualitative rather than quantitative. Extensive work is needed to fully demystify neural networks and even PTMs.

8.6 Summary and Further Readings

Up to now, we have overviewed the current progress and challenges of robust representation learning in NLP. In this last section, we will summarize the contents of this chapter and then provide more readings for reference.

Robustness is a crucial topic for reliable and trustworthy AI, and it is well recognized that existing NLP models are brittle in the complicated real world. In this chapter, we introduce four robustness issues in NLP following our proposed robustness hierarchy. Specifically, we first focus on integrity, which means whether models can work well on common cases without inner vulnerabilities, with backdoor robustness as a typical example. Second, we turn to external safety and discuss potential adversarial attacks models may face and corresponding defenses. Then, we consider real-world situations where models are supposed to be resilient under unseen, extreme even "black swan" events. We discuss three kinds of distribution shifts, namely, spurious correlation, domain shift, and subpopulation shift. Finally, we examine the highest demand posed on representation learning models and interpretability. We introduce the current stages in explaining model functionality and mechanism.

On backdoor robustness, Li et al. [52] give a unified overview on backdoor attack and defense, and their backdoor resource repository[8] is also beneficial. Roth et al. [76] and Wang et al. [100] provide comprehensive surveys on textual adversarial attack and defense. You can also find more related papers from our paper list.[9] Shen et al. [80] provide a holistic view of out-of-distribution robustness. Wiegreffe et al. [105] summarize current research progress in explainable NLP.

On the internal and external threats against machine learning systems, Hendrycks et al. [27] give an insightful discussion on model robustness, monitoring, alignment, and external safety. Bommasani et al. [5] also provide their opinions in Sections 4.7, 4.8, and 4.9.

Acknowledgments The contributions of all authors for the second edition are as follows: Zhiyuan Liu, Yankai Lin, and Maosong Sun designed the overall architecture of this chapter. Ganqu Cui drafted this chapter. Zhiyuan Liu and Yankai Lin proofread and revised this chapter.

We thank Chaojun Xiao, Yujia Qin, Yuan Yao, Zheni Zeng, Yangyi Chen, Weilin Zhao, Chaoqun He, and Lifan Yuan for proofreading the chapter.

This chapter is about robust representation learning and is the newly complemented content in the second edition of the book *Representation Learning for Natural Language Processing*. The first edition of the book was published in 2020 [55].

References

1. Facebook translates 'good morning' into 'attack them', leading to arrest. https://www.theguardian.com/technology/2017/oct/24/facebook-palestine-israel-translates-good-morning-attack-them-arrest.
2. Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *Proceedings of EMNLP*, 2018.
3. Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.
4. John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of EMNLP*, 2006.
5. Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.
6. Daniel Borkan, Lucas Dixon, Jeffrey Sorensen, Nithum Thain, and Lucy Vasserman. Nuanced metrics for measuring unintended bias with real data for text classification. In *Proceedings of WWW*, 2019.
7. Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *Proceedings of USENIX Security*, 2021.
8. Chuanshuai Chen and Jiazhu Dai. Mitigating backdoor attacks in lstm-based text classification systems by backdoor keyword identification. *Neurocomputing*, 2021.
9. Christopher Clark, Mark Yatskar, and Luke Zettlemoyer. Don't take the easy way out: Ensemble based methods for avoiding known dataset biases. In *Proceedings of EMNLP*, 2019.

[8] https://github.com/THUYimingLi/backdoor-learning-resources.

[9] https://github.com/thunlp/TAADpapers.

10. Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does BERT look at? an analysis of bert's attention. In *Proceedings of ACL Workshop BlackboxNLP*, 2019.
11. Ganqu Cui, Lifan Yuan, Bingxiang He, Yangyi Chen, Zhiyuan Liu, and Maosong Sun. A unified evaluation of textual backdoor learning: Frameworks and benchmarks. In *Proceedings of NeurIPS: Datasets and Benchmarks Track*, 2022.
12. Peng Cui and Susan Athey. Stable learning establishes some common ground between causal inference and machine learning. *Nature Machine Intellegence*, 2022.
13. Jiazhu Dai, Chuanshuai Chen, and Yufeng Li. A backdoor attack against lstm-based text classification systems. *IEEE Access*, 2019.
14. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.
15. Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.
16. Yana Dranker, He He, and Yonatan Belinkov. Irm—when it works and when it doesn't: A test case of natural language inference. In *Proceedings of NeurIPS*, 2021.
17. Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. In *Proceedings of EACL*, 2017.
18. Steffen Eger and Yannik Benz. From hero to zéroe: A benchmark of low-level adversarial attacks. In *Proceedings of AACL*, 2020.
19. Steffen Eger, Gözde Gül Şahin, Andreas Rücklé, Ji-Ung Lee, Claudia Schulz, Mohsen Mesgar, Krishnkant Swarnkar, Edwin Simpson, and Iryna Gurevych. Text processing like humans do: Visually attacking and shielding nlp systems. In *Proceedings of NAACL*, 2019.
20. Shi Feng, Eric Wallace, Alvin Grissom II, Mohit Iyyer, Pedro Rodriguez, and Jordan Boyd-Graber. Pathologies of neural models make interpretations difficult. In *Proceedings of EMNLP*, 2018.
21. Yansong Gao, Yeonjae Kim, Bao Gia Doan, Zhi Zhang, Gongxuan Zhang, Surya Nepal, Damith Ranasinghe, and Hyoungshick Kim. Design and evaluation of a multi-domain trojan detection method on deep neural networks. *IEEE Transactions on Dependable and Secure Computing*, 2021.
22. Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of EMNLP*, 2021.
23. Amir Globerson and Sam Roweis. Nightmare at test time: robust learning by feature deletion. In *Proceedings of ICML*, 2006.
24. Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.
25. Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of ICML*, 2017.
26. Chuan Guo, Alexandre Sablayrolles, Hervé Jégou, and Douwe Kiela. Gradient-based adversarial attacks against text transformers. In *Proceedings of EMNLP*, 2021.
27. Dan Hendrycks, Nicholas Carlini, John Schulman, and Jacob Steinhardt. Unsolved problems in ml safety. *arXiv preprint arXiv:2109.13916*, 2021.
28. Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *Proceedings of ICLR*, 2019.
29. Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, Adam Dziedzic, Rishabh Krishnan, and Dawn Song. Pretrained transformers improve out-of-distribution robustness. In *Proceedings of ACL*, 2020.
30. Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. Deceiving google's perspective api built for detecting toxic comments. *arXiv preprint arXiv:1702.08138*, 2017.
31. Kuan-Hao Huang and Kai-Wei Chang. Generating syntactically controlled paraphrases without using annotated parallel pairs. In *Proceedings of EACL*, 2021.

32. Peter J Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 1964.
33. Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of NAACL*, 2018.
34. Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *Proceedings of ICLR*, 2017.
35. Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does BERT learn about the structure of language? In *Proceedings of ACL*, 2019.
36. Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of EMNLP*, 2017.
37. Zhengbao Jiang, Jun Araki, Haibo Ding, and Graham Neubig. How can we know *When* language models know? on the calibration of language models for question answering. *Transactions of the Association for Computational Linguistics*, 2021.
38. Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of AAAI*, 2020.
39. Erik Jones, Robin Jia, Aditi Raghunathan, and Percy Liang. Robust encodings: A framework for combating adversarial typos. In *Proceedings of ACL*, 2020.
40. Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield Dodds, Nova DasSarma, Eli Tran-Johnson, et al. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.
41. Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of ICML*, 2017.
42. Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton Earnshaw, Imran S. Haque, Sara M. Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. WILDS: A benchmark of in-the-wild distribution shifts. In *Proceedings of ICML*, 2021.
43. Kun Kuang, Peng Cui, Susan Athey, Ruoxuan Xiong, and Bo Li. Stable prediction across unknown environments. In *Proceedings of KDD*, 2018.
44. Ram Shankar Siva Kumar, Magnus Nyström, John Lambert, Andrew Marshall, Mario Goertzel, Andi Comissoneru, Matt Swann, and Sharon Xia. Adversarial machine learning-industry perspectives. In *Proceedingds of Security and Privacy Workshops*, 2020.
45. Thai Le, Jooyoung Lee, Kevin Yen, Yifan Hu, and Dongwon Lee. Perturbations in the wild: Leveraging human-written text perturbations for realistic adversarial attack and defense. In *Findings of ACL*, 2022.
46. Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of ICCV*, 2017.
47. Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. Textbugger: Generating adversarial text against real-world applications. In *Proceedings of NDSS*, 2018.
48. Jiwei Li, Will Monroe, and Dan Jurafsky. Understanding neural networks through representation erasure. *arXiv preprint arXiv:1612.08220*, 2016.
49. Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. BERT-ATTACK: Adversarial attack against BERT using BERT. In *Proceedings of EMNLP*, 2020.
50. Linyang Li, Demin Song, Xiaonan Li, Jiehang Zeng, Ruotian Ma, and Xipeng Qiu. Backdoor attacks on pre-trained models by layerwise weight poisoning. In *Proceedings of EMNLP*, 2021.
51. Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of ACL-IJCNLP*, 2021.
52. Yiming Li, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey. *arXiv preprint arXiv:2007.08745*, 2020.
53. Stephanie Lin, Jacob Hilton, and Owain Evans. Teaching models to express their uncertainty in words. *arXiv preprint arXiv:2205.14334*, 2022.

54. Evan Zheran Liu, Behzad Haghgoo, Annie S. Chen, Aditi Raghunathan, Pang Wei Koh, Shiori Sagawa, Percy Liang, and Chelsea Finn. Just train twice: Improving group robustness without training group information. In *Proceedings of ICML*, 2021.
55. Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation Learning for Natural Language Processing*. Springer, 2020.
56. Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Proceedings of ICLR*, 2018.
57. Rabeeh Karimi Mahabadi, Yonatan Belinkov, and James Henderson. End-to-end bias mitigation by modelling biases in corpora. In *Proceedings of ACL*, 2020.
58. Rishabh Maheshwary, Saket Maheshwary, and Vikram Pudi. Generating natural language attacks in a hard label black box setting. In *Proceedings of AAAI*, 2021.
59. Abraham Harold Maslow. A dynamic theory of human motivation. 1958.
60. R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of ACL*, 2019.
61. John X. Morris, Eli Lifland, Jack Lanchantin, Yangfeng Ji, and Yanjun Qi. Reevaluating adversarial examples in natural language. In *Findings of EMNLP*, 2020.
62. John X. Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. TextAttack: A framework for adversarial attacks, data augmentation, and adversarial training in NLP. In *Proceedings of EMNLP*, 2020.
63. Maximilian Mozes, Pontus Stenetorp, Bennett Kleinberg, and Lewis D. Griffin. Frequency-guided word substitutions for detecting textual adversarial examples. In *Proceedings of EACL*, 2021.
64. Moin Nadeem, Anna Bethke, and Siva Reddy. Stereoset: Measuring stereotypical bias in pretrained language models. In *Proceedings of ACL*, 2021.
65. Aakanksha Naik, Abhilasha Ravichander, Norman M. Sadeh, Carolyn Penstein Rosé, and Graham Neubig. Stress test evaluation for natural language inference. In *Proceedings of COLING*, 2018.
66. Yonatan Oren, Shiori Sagawa, Tatsunori B. Hashimoto, and Percy Liang. Distributionally robust language modeling. In *Proceedings of EMNLP-IJCNLP*, 2019.
67. Jonas Peters, Peter Bühlmann, and Nicolai Meinshausen. Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2016.
68. Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *Proceedings of EMNLP-IJCNLP*, 2019.
69. Fanchao Qi, Yangyi Chen, Mukai Li, Yuan Yao, Zhiyuan Liu, and Maosong Sun. ONION: A simple and effective defense against textual backdoor attacks. In *Proceedings of EMNLP*, 2021.
70. Fanchao Qi, Yangyi Chen, Xurui Zhang, Mukai Li, Zhiyuan Liu, and Maosong Sun. Mind the style of text! adversarial and backdoor attacks based on text style transfer. In *Proceedings of EMNLP*, 2021.
71. Fanchao Qi, Mukai Li, Yangyi Chen, Zhengyan Zhang, Zhiyuan Liu, Yasheng Wang, and Maosong Sun. Hidden killer: Invisible textual backdoor attacks with syntactic trigger. In *Proceedings of ACL-IJCNLP*, 2021.
72. Fanchao Qi, Yuan Yao, Sophia Xu, Zhiyuan Liu, and Maosong Sun. Turn the combination lock: Learnable textual backdoor attacks via word substitution. In *Proceedings of ACL-IJCNLP*, 2021.
73. Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.
74. Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of ACL*, 2019.
75. Marco Túlio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with checklist. In *Proceedings of ACL*, 2020.

76. Tom Roth, Yansong Gao, Alsharif Abuadbba, Surya Nepal, and Wei Liu. Token-modification adversarial attacks for natural language processing: A survey. *arXiv preprint arXiv:2103.00676*, 2021.
77. Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *arXiv preprint arXiv:1911.08731*, 2019.
78. Lujia Shen, Shouling Ji, Xuhong Zhang, Jinfeng Li, Jing Chen, Jie Shi, Chengfang Fang, Jianwei Yin, and Ting Wang. Backdoor pre-trained models can transfer to all. In *Proceedings of CCS*, 2021.
79. Zheyan Shen, Peng Cui, Kun Kuang, Bo Li, and Peixuan Chen. Causally regularized learning with agnostic data selection bias. In *Proceedings of MM*, 2018.
80. Zheyan Shen, Jiashuo Liu, Yue He, Xingxuan Zhang, Renzhe Xu, Han Yu, and Peng Cui. Towards out-of-distribution generalization: A survey. *arXiv preprint arXiv:2108.13624*, 2021.
81. Chenglei Si, Zhengyan Zhang, Fanchao Qi, Zhiyuan Liu, Yasheng Wang, Qun Liu, and Maosong Sun. Better robustness by more coverage: Adversarial and mixup data augmentation for robust finetuning. In *Findings of ACL-IJCNLP*, 2021.
82. Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proceedings of ICLR, Workshop Track Proceedings*, 2014.
83. Nimit Sohoni, Jared Dunnmon, Geoffrey Angus, Albert Gu, and Christopher Ré. No subclass left behind: Fine-grained robustness in coarse-grained classification problems. In *Proceedings of NeurIPS*, 2020.
84. Baochen Sun and Kate Saenko. Deep CORAL: correlation alignment for deep domain adaptation. In *Proceedings of ECCV*, 2016.
85. Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of ICML*, 2017.
86. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of CVPR*, 2016.
87. Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proceedings of ICLR*, 2014.
88. Samson Tan, Shafiq Joty, Min-Yen Kan, and Richard Socher. It's morphin'time! combating linguistic discrimination with inflectional perturbations. In *Proceedings of ACL*, 2020.
89. Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. Measuring robustness to natural distribution shifts in image classification. In *Proceedings of NeurIPS*, 2020.
90. Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of ACL*, 2019.
91. Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, and Ellie Pavlick. What do you learn from context? probing for sentence structure in contextualized word representations. In *Proceedings of ICLR*, 2019.
92. Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *Proceedings of CVPR*, 2011.
93. Lifu Tu, Garima Lalwani, Spandana Gella, and He He. An empirical study on robustness to spurious correlations using pre-trained language models. *Transactions of the Association for Computational Linguistics*, 2020.
94. Prasetya Ajie Utama, Nafise Sadat Moosavi, and Iryna Gurevych. Towards debiasing nlu models from unknown biases. In *Proceedings of EMNLP*, 2020.
95. Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. Universal adversarial triggers for attacking and analyzing nlp. In *Proceedings of EMNLP-IJCNLP*, 2019.

96. Eric Wallace, Jens Tuyls, Junlin Wang, Sanjay Subramanian, Matt Gardner, and Sameer Singh. Allennlp interpret: A framework for explaining predictions of NLP models. In *Proceedings of EMNLP-IJCNLP*, 2019.
97. Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of EMNLP Workshop*, 2018.
98. Boxin Wang, Hengzhi Pei, Boyuan Pan, Qian Chen, Shuohang Wang, and Bo Li. T3: Tree-autoencoder constrained adversarial text generation for targeted attack. In *Proceedings of EMNLP*, 2019.
99. Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, and Tao Qin. Generalizing to unseen domains: A survey on domain generalization. In *Proceedings of IJCAI*, 2021.
100. Wenqi Wang, Run Wang, Lina Wang, Zhibo Wang, and Aoshuang Ye. Towards a robust deep neural network in texts: A survey. *arXiv preprint arXiv:1902.07285*, 2019.
101. Xiao Wang, Qin Liu, Tao Gui, Qi Zhang, Yicheng Zou, Xin Zhou, Jiacheng Ye, Yongxin Zhang, Rui Zheng, Zexiong Pang, et al. Textflint: Unified multilingual robustness evaluation toolkit for natural language processing. In *Proceedings of ACL*, 2021.
102. Xiaozhi Wang, Kaiyue Wen, Zhengyan Zhang, Lei Hou, Zhiyuan Liu, and Juanzi Li. Finding skill neurons in pre-trained transformer-based language models. In *Proceedings of EMNLP*, 2022.
103. Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.
104. Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, et al. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359*, 2021.
105. Sarah Wiegreffe and Ana Marasovic. Teach me to explain: A review of datasets for explainable natural language processing. In *Proceedings of NeurIPS: Datasets and Benchmarks Track*, 2021.
106. Sarah Wiegreffe and Yuval Pinter. Attention is not not explanation. In *Proceedings of EMNLP-IJCNLP*, 2019.
107. Lei Xu, Yangyi Chen, Ganqu Cui, Hongcheng Gao, and Zhiyuan Liu. Exploring the universal vulnerability of prompt-based learning paradigm. In *Findings of NAACL*, 2022.
108. Yadollah Yaghoobzadeh, Soroush Mehri, Remi Tachet, Timothy J. Hazen, and Alessandro Sordoni. Increasing robustness to spurious correlations using forgettable examples. In *Proceedings of EACL*, 2021.
109. Wenkai Yang, Lei Li, Zhiyuan Zhang, Xuancheng Ren, Xu Sun, and Bin He. Be careful about poisoned word embeddings: Exploring the vulnerability of the embedding layers in NLP models. In *Proceedings of NAACL-HLT*, 2021.
110. Wenkai Yang, Yankai Lin, Peng Li, Jie Zhou, and Xu Sun. RAP: robustness-aware perturbations for defending against backdoor attacks on NLP models. In *Proceedings of EMNLP*, 2021.
111. Wenkai Yang, Yankai Lin, Peng Li, Jie Zhou, and Xu Sun. Rethinking stealthiness of backdoor attack against nlp models. In *Proceedings of ACL-IJCNLP*, 2021.
112. Zonghan Yang and Yang Liu. On robust prefix-tuning for text classification. In *Proceedings of ICLR*, 2022.
113. Muchao Ye, Chenglin Miao, Ting Wang, and Fenglong Ma. Texthoaxer: Budgeted hard-label adversarial attacks on text. In *Proceedings of AAAI*, 2022.
114. Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of ACL*, 2020.
115. Guoyang Zeng, Fanchao Qi, Qianrui Zhou, Tingji Zhang, Bairu Hou, Yuan Zang, Zhiyuan Liu, and Maosong Sun. Openattack: An open-source textual adversarial attack toolkit. In *Proceedings of ACL*, 2021.

116. Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *Proceedings of ICLR*, 2018.
117. Xingxuan Zhang, Peng Cui, Renzhe Xu, Linjun Zhou, Yue He, and Zheyan Shen. Deep stable learning for out-of-distribution generalization. In *Proceedings of CVPR*, 2021.
118. Yuan Zhang, Jason Baldridge, and Luheng He. Paws: Paraphrase adversaries from word scrambling. In *Proceedings of NAACL*, 2019.
119. Zhengyan Zhang, Guangxuan Xiao, Yongwei Li, Tian Lv, Fanchao Qi, Zhiyuan Liu, Yasheng Wang, Xin Jiang, and Maosong Sun. Red alarm for pre-trained models: Universal vulnerabilities by neuron-level backdoor attacks. *arXiv preprint arXiv:2101.06969*, 2021.
120. Yichao Zhou, Jyun-Yu Jiang, Kai-Wei Chang, and Wei Wang. Learning to discriminate perturbations for blocking adversarial attacks in text classification. In *Proceedings of EMNLP-IJCNLP*, 2019.
121. Biru Zhu, Yujia Qin, Ganqu Cui, Yangyi Chen, Weilin Zhao, Chong Fu, Yangdong Deng, Zhiyuan Liu, Jingang Wang, Wei Wu, Maosong Sun, and Ming. Gu. Moderate-fitting as a natural backdoor defender for pre-trained language models. In *Proceedings of NeurIPS*, 2022.
122. Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. FreeLB: Enhanced adversarial training for natural language understanding. In *Proceedings of ICLR*, 2020.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.