UNIVERSITY OF RIJEKA
FACULTY OF INFORMATICS AND DIGITAL
TECHNOLOGIES

Karlo Babić

THE PYRAMIDAL RECURSIVE NEURAL NETWORK FOR MULTILEVEL TEXT REPRESENTATION

DOCTORAL THESIS

Rijeka, year 2025.

UNIVERSITY OF RIJEKA
FACULTY OF INFORMATICS AND DIGITAL
TECHNOLOGIES

Karlo Babić

THE PYRAMIDAL RECURSIVE NEURAL NETWORK FOR MULTILEVEL TEXT REPRESENTATION

DOCTORAL THESIS

Supervisor: prof. dr. sc. Ana Meštrović

Rijeka, year 2025.

SVEUČILIŠTE U RIJECI
FAKULTET INFORMATIKE I DIGITALNIH TEHNOLOGIJA

Karlo Babić

PIRAMIDALNO REKURZIVNA NEURONSKA MREŽA ZA VIŠERAZINSKO REPREZENTIRANJE TEKSTA

DOKTORSKI RAD

Mentorica: prof. dr. sc. Ana Meštrović

Rijeka, godina 2025.

Supervisor: prof. dr. sc. Ana Meštrović

The doctoral dissertation was defended on _____

at/in _____,

before a committee consisting of:

1. _____
2. _____
3. _____
4. _____
5. _____

Zahvale

Hvala mojoj obitelji, prijateljima i mentorima što su me podržavali na ovom putu.
Svemir je golem i hladan, a naš značaj postoji samo u onome što značimo jedni drugima.

Abstract

Natural Language Processing (NLP) has long relied on effective text representation to advance a range of applications, from sentiment analysis to machine translation. However, traditional text representation techniques often fail to fully capture the hierarchical and compositional nature of language, leading to a loss of critical semantic and syntactic information. This challenge is amplified by the unique hierarchical and symbolic structure of text, distinguishing it from other data modalities such as images or audio.

This thesis proposes a novel text representation learning approach, Pyramidal Recursive learning (PyRv), which employs the Pyramidal Recursive Neural Network (PyRvNN) architecture. PyRv addresses the limitations of existing methods by leveraging pyramidal recursion to model text hierarchically, progressing from subwords to sentences. The model is designed with four desired properties: hierarchical representation, representation compositionality, representation decodability, and self-supervised learning.

PyRv captures hierarchical relationships and compositional structures within text, ensuring robust representation decodability without reliance on labeled data. Experiments demonstrate its capability to produce multi-level, interpretable, and efficient representations. Evaluation includes intrinsic tasks, such as autoencoder decodability and representation compositionality, as well as downstream applications like plagiarism detection, memorization, and readability assessment. Results highlight strong performance in decoding accuracy and effective combining of lower-level embeddings.

The findings presented in this thesis establish PyRvNN as a step forward in text representation learning, providing a scalable, interpretable, and efficient framework for capturing the complex hierarchical structures inherent in natural language.

Keywords: compositionality, decodability, hierarchical representation, multi-level representation, neural networks, self-supervised learning, text representation

Sažetak

Obrada prirodnog jezika (NLP) uvelike se oslanja na učinkovitu reprezentaciju teksta kako bi unaprijedila različite primjene, od analize sentimenta do strojnog prevođenja. Međutim, tradicionalne tehnike reprezentacije teksta često ne uspijevaju u potpunosti obuhvatiti hijerarhijsku i kompozicijsku prirodu jezika, što rezultira gubitkom bitnih semantičkih i sintaktičkih informacija. Ovaj izazov dodatno je naglašen jedinstvenom hijerarhijskom i simboličkom strukturom teksta, koja ga razlikuje od drugih modaliteta podataka poput slika ili zvuka.

Ova disertacija predlaže novi pristup učenju reprezentacije teksta, Pyramidal Recursive learning (PyRv), koji koristi arhitekturu Pyramidal Recursive Neural Network (PyRvNN). PyRv rješava ograničenja postojećih metoda koristeći piramidalnu rekurziju za hijerarhijsko modeliranje teksta, napredujući od podriječi do rečenica. Model je dizajniran s četiri željena svojstva: hijerarhijska reprezentacija, kompozicionalnost reprezentacije, dekodabilnost reprezentacije i samonadzirano učenje.

PyRv reprezentira hijerarhijske odnose i kompozicijske strukture unutar teksta, osiguravajući robusnu dekodabilnost reprezentacije bez oslanjanja na označene podatke. Eksperimentalni rezultati potvrđuju sposobnost generiranja višerazinskih, interpretabilnih i učinkovitih reprezentacija. Evaluacija obuhvaća intrinzične zadatke, uključujući dekodabilnost autoenkodera i kompozicionalnost reprezentacije, kao i primjene u detekciji plagijata, memorijalizaciji i procjeni čitljivosti. Rezultati ističu visoku točnost dekodiranja i učinkovito kombiniranje nižerazinskih reprezentacija.

Rezultati predstavljeni u ovoj disertaciji uspostavljaju PyRvNN kao korak naprijed u učenju reprezentacije teksta, pružajući skalabilan, interpretabilan i učinkovit okvir za obuhvaćanje složenih hijerarhijskih struktura svojstvenih prirodnom jeziku.

Ključne riječi: kompozicionalnost, dekodabilnost, hijerarhijska reprezentacija, višerazinska reprezentacija, neuronske mreže, samonadzirano učenje, reprezentacija teksta

Prošireni sažetak

Obrada prirodnog jezika (NLP) kontinuirano napreduje kroz razvoj novih metoda reprezentacije teksta, omogućujući strojevima bolje razumijevanje, interpretaciju i generiranje ljudskog jezika. Ključni izazov u ovom području leži u učinkovitom modeliranju semantičkih i sintaktičkih struktura jezika, koje su intrinzično hijerarhijske i kompozicijske. Tradicionalne metode za reprezentiranje teksta, poput one-hot vektora, bag-of-words i TF-IDF, unatoč svojoj jednostavnosti i primjenjivosti, ne uspijevaju adekvatno obuhvatiti hijerarhijsku prirodu jezika, gubeći bitne informacije o kontekstu i međusobnim odnosima među riječima.

S pojavom neuronskih mreža, metode poput Word2Vec, FastText i GloVe omogućile su bolju semantičku reprezentaciju teksta kroz gusto kodirane vektore. Ipak, ove metode i dalje ne uspijevaju eksplicitno modelirati hijerarhijske odnose među jezičnim jedinicama. Transformerski modeli poput BERT-a, T5 i GPT-a donijeli su značajna poboljšanja u dinamičkoj reprezentaciji teksta, ali i dalje postoje izazovi vezani uz računalne zahtjeve, interpretabilnost i sposobnost rekonstrukcije originalnog teksta iz naučenih reprezentacija.

Ovaj doktorski rad predlaže novi pristup učenju reprezentacije teksta pod nazivom Pyramidal Recursive learning (PyRv), koji koristi arhitekturu Pyramidal Recursive Neural Network (PyRvNN). PyRv je dizajniran s četiri temeljna svojstva: hijerarhijska reprezentacija, kompozicionalnost reprezentacije, dekodabilnost i samonadzirano učenje.

Hijerarhijska reprezentacija omogućuje modelu da obuhvati različite razine jezičnih jedinica, od znakova do cijelih rečenica, čime osigurava preciznije i kontekstualno bogatije reprezentacije. Kompozicionalnost reprezentacije osigurava da se složeniji jezični izrazi mogu formirati kombiniranjem jednostavnijih komponenti, čime se poboljšava sposobnost modela u razumijevanju i generiranju teksta. Dekodabilnost omogućuje rekonstrukciju izvornog teksta iz naučenih reprezentacija, čime se povećava interpretabilnost modela i omogućuje dublje razumijevanje njegovog funkcioniranja. Samonadzirano učenje omogućuje modelu da uči iz neoznačenih podataka, čime se smanjuje potreba za velikim količinama označenih podataka te poboljšava generalizacija na različite jezične domene.

Eksperimentalni rezultati potvrđuju da PyRv osigurava učinkovitu dekodabilnost, interpretabilnost i preciznu kompozicionalnost naučenih reprezentacija. Evaluacija PyRv metode kroz PyRvNN model uključuje intrinzične zadatke, poput dekodabilnosti autoenkodera i sposobnosti kombiniranja nižih razina reprezentacije, kao i primjene u detekciji plagijata, memorijalizaciji i procjeni čitljivosti teksta. Dobiveni rezultati ističu visoku točnost dekodiranja te sposobnost modela da učinkovito kombinira nižerazinske reprezentacije u složenije tekstualne jedinice.

Primarni znanstveni doprinosi ovog doktorskog rada su:

- Razvoj nove metode za učenje reprezentacije teksta temeljenog na piramidalnoj rekurzivnoj neuronskoj mreži.
- Opsežna evaluacija stabilnosti treniranja, kvalitete naučenih reprezentacija i performansi

na NLP zadacima.

- Poboljšanje prethodno naučenih reprezentacija za morfološki bogate jezike, poput hrvatskog, omogućujući bolje rezultate uz manje podataka prilikom prilagodbe na nove domene.

Formulirane su hipoteze koje usmjeravaju istraživanje:

- H1: Trening piramidalne rekurzivne neuronske mreže za reprezentaciju teksta može biti stabilan i učinkovit.
- H2: Piramidalna rekurzija omogućuje istovremeno generiranje reprezentacija za različite razine jezičnih jedinica.
- H3: Predložena metoda nadmašuje postojeće metode na odabranim zadacima i domenama.

Doktorski rad pokriva različite aspekte predložene metode, uključujući teorijsku osnovu, implementaciju modela, eksperimentalnu analizu te potencijalne primjene u području obrade prirodnog jezika. Ova istraživanja predstavljaju korak naprijed u razvoju skalabilnih i interpretabilnih metoda za modeliranje složenih hijerarhijskih struktura koje su svojstvene prirodnom jeziku.

Contents

1. Introduction . . . 1
- 1.1. Problem Statement . . . 2
  - 1.1.1. Limitations of Current Text Embedding Techniques . . . 2
  - 1.1.2. Novel Text Representation Method . . . 3
- 1.2. Objectives and Scope . . . 3
  - 1.2.1. Hypotheses . . . 3
  - 1.2.2. Contributions . . . 3
  - 1.2.3. Scope of the Study . . . 4
- 1.3. Overview of Proposed Method . . . 4
- 1.4. Thesis Structure . . . 5

2. Literature Review . . . 6
- 2.1. Comparative Analysis . . . 6
  - 2.1.1. Model Categorization . . . 9
  - 2.1.2. Shallow Models . . . 12
  - 2.1.3. Recurrent Models . . . 15
  - 2.1.4. Recursive Models . . . 19
  - 2.1.5. Convolutional Models . . . 23
  - 2.1.6. Attention Models . . . 25
  - 2.1.7. Analysis Conclusions . . . 31
- 2.2. Desirable Model Properties . . . 33
  - 2.2.1. Representation Compositionality . . . 33
  - 2.2.2. Hierarchical Representation . . . 35
  - 2.2.3. Representation Decodability . . . 37
  - 2.2.4. Self-supervised Learning . . . 38
- 2.3. Identified Gaps and Challenges . . . 39

3. Theoretical Background . . . 41
- 3.1. Neural Network Representations . . . 41
  - 3.1.1. The Role of Representations in Machine Learning . . . 42

3.1.2. Representations in Text . . . 42
3.2. Autoencoders and Autoregressive Learning . . . 44
3.2.1. Autoencoders . . . 44
3.2.2. Autoregressive Learning . . . 45
3.2.3. Integration of Autoencoding and Autoregression . . . 46
3.3. Recursive Neural Networks . . . 47
3.3.1. Backpropagation Through Structure . . . 48
3.4. Pyramidal Recursive Learning . . . 49
3.4.1. Training Challenges and Solutions . . . 50

4. Methodology . . . 51
4.1. Pyramidal Recursive Learning Method . . . 52
4.1.1. Complexity . . . 54
4.2. Pyramidal Recursive Neural Network Architecture . . . 54
4.2.1. Input Representations . . . 56
4.2.2. Embedding Dimension . . . 56
4.3. Training Procedure . . . 59
4.3.1. Training Data . . . 61
4.3.2. Pyramidal Training . . . 63
4.3.3. Loss Computation . . . 63
4.3.4. Training Setup . . . 67
4.4. Evaluations . . . 69
4.5. Model Limitations and Constraints . . . 71

5. Experiments and Results . . . 73
5.1. Representation Decodability Property . . . 74
5.1.1. Results . . . 74
5.2. Representation Compositionality Property . . . 77
5.2.1. Embedding method . . . 79
5.2.2. Evaluation . . . 80
5.2.3. Summary and Conclusions . . . 87
5.3. Memorization Task . . . 93
5.3.1. Results . . . 93
5.4. Plagiarism Task . . . 94
5.4.1. Results . . . 95
5.5. Readability Task . . . 95
5.5.1. Results . . . 96

6. Discussion . . . 98
6.1. Contributions . . . 98

6.2. Hypotheses . . . 100

7. Conclusion . . . 102

Bibliography . . . 105

8. Abbreviations . . . 119

A. A: Implementation . . . 126
A.1. Module main.py . . . 127
A.2. Module modelarch.py . . . 128
A.3. Module trainalg.py . . . 129
A.3.1. Function `pyrv_train()` . . . 129
A.3.2. Function `train_init()` . . . 129
A.3.3. Function `pyramidal_recursion()` . . . 129
A.3.4. Function `loss()` . . . 131
A.4. Module dataprep.py . . . 131
A.4.1. Function `generate_arrays_from_data()` . . . 131
A.4.2. Function `text_to_onehots_by_words()` . . . 131
A.5. Module datamanip.py . . . 132
A.5.1. Function `token_pairs_prep()` . . . 132
A.5.2. Function `subwordlvl_pairs_prep()` . . . 132
A.5.3. Function `phraselvl_pairs_prep()` . . . 133

B. B: Additional Results . . . 135

Chapter 1

Introduction

Natural language processing (NLP) is a rapidly evolving field dedicated to enabling machines to understand, interpret, and generate human language. Its significance spans various applications, including machine translation, sentiment analysis, and information retrieval, transforming how humans interact with technology. Central to the success of these applications is the effective representation of text.

Text representation serves as the foundation for most NLP tasks, providing a means for machines to process linguistic data. Over time, text representation techniques have evolved significantly. Traditional methods such as one-hot encoding, bag-of-words, and Term Frequency-Inverse Document Frequency (TFIDF) were widely used but were limited in their ability to capture semantic or syntactic relationships. These methods provided sparse and high-dimensional representations, often failing to preserve word order or contextual meaning.

With the advent of neural networks, dense embeddings like Word2Vec [1], FastText [2], and GloVe [3] brought a paradigm shift by encoding semantic information into low-dimensional continuous vector spaces. These embeddings positioned similar words close to each other, enabling improved performance in downstream tasks. More recently, transformer-based models such as BERT [4] have set new benchmarks in text representation by leveraging contextualized embeddings that adapt dynamically to the surrounding text.

The rapid evolution of Large Language Models (LLMs) has further shaped the landscape. Early foundational models like GPT-1 [5] showcased the potential of large-scale generative pre-training. Subsequent developments included instruction-following models, with Google's FLAN [6] enhancing few-shot learning via instruction tuning, and OpenAI's InstructGPT series [7] using Reinforcement Learning from Human Feedback (RLHF) to improve instruction adherence, as seen in models like ChatGPT. More recently, techniques like Chain-of-Thought (CoT) prompting [8, 9], initially prominent with Google's PaLM [10], and models such as o1 [11] exploring extended deliberation, have pushed the boundaries of complex task performance. While large-scale models, often designed for text generation, have shown significant success, further research into alternative approaches is still important. Specifically, there is a need to

explore dedicated text embedding models that are smaller, simpler, more interpretable, or computationally efficient.

In comparison to other modalities like images and audio, text representation presents unique challenges. While images and audio are often continuous and signal-based, text is inherently sequential and symbolic, composed of discrete units such as characters, words, and sentences. This symbolic nature introduces a hierarchical structure that requires specialized approaches for effective representation.

The importance of robust text representation cannot be overstated, as it directly influences the performance of NLP tasks such as sentiment analysis, machine translation, question answering, and text summarization. Better representations lead to models that are more accurate, interpretable, and capable of generalizing across diverse contexts and languages. This underscores the need for innovative methods to represent text in a way that captures its hierarchical and compositional nature while preserving crucial semantic and syntactic information.

1.1 Problem Statement

The rapid advancements in text representation have revolutionized NLP, yet there remains a critical need to explore novel learning methods that better capture the hierarchical and compositional nature of language. A significant challenge lies in developing representations and models that not only improve the quality of embeddings but also exhibit desirable properties such as hierarchical representation, compositionality, decodability, and self-supervised learning.

1.1.1 Limitations of Current Text Embedding Techniques

Integrating Multiple Representational Properties. Current text embedding methods often struggle to concurrently provide compositionality, hierarchical representations, decodability, and self-supervised learning. Achieving all these desired features, particularly a clear, multi-level decodable hierarchy through purely self-supervised means, remains an active area of research and a notable limitation in existing techniques.

Loss of Information in Averaging Methods. Simple word embedding composition techniques, such as those that compute the element-wise mean of multiple word vectors to produce a single phrase or sentence representation, often result in the loss of critical information. These methods fail to capture word order and syntactic structures, leading to representations that lack the richness and precision required for complex NLP tasks. This limitation becomes particularly apparent in tasks that depend on subtle linguistic nuances, such as paraphrase detection or dependency relation classification.

Computational Demands and Memory Constraints. Recursive models, particularly those trained on large-scale datasets, often encounter significant computational and memory challenges. The sequential and hierarchical nature of these models increases training time and complexity, limiting their feasibility for real-world applications requiring efficiency and scalability.

1.1.2 Novel Text Representation Method

To address these limitations, this research aims to develop a novel text representation learning method that integrates the following desirable properties:

- Representation Compositionality: The ability to combine representations from subcomponents in a meaningful and structured way, capturing the semantics of complex texts.
- Hierarchical Representation: The capability to encode text at multiple levels of granularity, from characters and subwords to phrases and sentences.
- Representation Decodability: The ability to reconstruct the original text from its learned representation, ensuring interpretability and evaluation of the model's learning.
- Self-Supervised Learning: Leveraging large-scale unlabeled data to learn effective text representations without requiring annotated datasets or external parsing tools.

This research contributes to the domains of NLP, deep learning, and representation learning by introducing a method that overcomes the existing barriers in text representation. A model with the aforementioned properties could enhance performance in various NLP applications, including sentiment analysis, machine translation, and plagiarism detection, while also providing insights into the development of hierarchical, compositional, and decodable embeddings.

1.2 Objectives and Scope

1.2.1 Hypotheses

The following hypotheses are formulated to guide the research:

- H1: The training of pyramidal recursive neural network for text representation can be stable and efficient.
- H2: Pyramidal recursive training can simultaneously generate text representations for different levels of linguistic units.
- H3: The proposed method has better performance compared to other competing methods on selected tasks and domains.

1.2.2 Contributions

The primary contributions of this research are:

- A novel method for learning text representations with pyramid-structured recursive neural networks.
- Extensive evaluation of the proposed method in terms of training stability, quality of learned embeddings, and predictive performance on downstream tasks.
- Improved pre-trained representations for morphologically rich languages (e.g. Croatian) in terms of less training material when adapting to new domains.

1.2.3 Scope of the Study

This research situates itself at the intersection of NLP, deep learning, and representation learning. By addressing the limitations of existing methods and integrating hierarchical, compositional, and decodable representations into a single model, it introduces a novel approach to text representation learning.

The scope of this study is defined as follows:

- Data: The primary dataset consists of English Wikipedia articles, providing a diverse and extensive source of text data.
- Model: The proposed model employs a pyramidal recursive architecture with a limited number of layers and parameters to ensure computational feasibility while maintaining effectiveness.
- Learning Paradigm: The model is trained in a self-supervised manner, eliminating the need for labeled data or parse trees, which enhances scalability and applicability to diverse datasets.

1.3 Overview of Proposed Method

The Pyramidal Recursive Learning (PyRv) method introduces an innovative approach to representation learning, addressing important limitations of existing text representation methods. By employing a recursive and pyramidal architecture, PyRv captures hierarchical relationships within text, spanning granular linguistic units such as subwords, words, phrases, and sentences. This method ensures the creation of multi-level representations while maintaining interpretability and efficiency.

At the core of this method lies the PyRvNN, a model that differentiates itself from traditional recursive and sequential architectures in several key aspects. PyRvNN organizes text representations hierarchically, where pairs of representations are recursively combined at each level. Unlike methods dependent on predefined parsing trees, PyRvNN dynamically builds its hierarchy during training.

The PyRvNN model integrates two primary learning objectives that enhance its representational power. First, the autoencoding head reconstructs paired representations back into their

original components, ensuring that the learned representations remain decodable and semantically consistent. Second, the autoregressive head predicts neighboring representations within the pyramid, enabling the model to effectively capture and leverage contextual relationships. These dual objectives are trained in a self-supervised manner, which allows PyRvNN to utilize large-scale, unlabeled datasets without requiring costly annotation.

1.4 Thesis Structure

This thesis is organized into seven chapters:

- **Chapter 1: Introduction** - provides an overview of the research motivation, problem statement, objectives, and a summary of the proposed method.
- **Chapter 2: Literature Review** - surveys existing approaches in text representation learning. It highlights the limitations of current recursive models and identifies gaps addressed by the proposed approach.
- **Chapter 3: Theoretical Background** - establishes the theoretical foundation for the PyRvNN model, including key concepts such as autoencoding, autoregressive learning, and recursive neural networks.
- **Chapter 4: Methodology** - details the design and theoretical underpinnings of PyRvNN, covering the pyramidal recursive learning method, model architecture, training procedures, and evaluation metrics.
- **Chapter 5: Experiments and Results** - presents the results of experiments designed to evaluate the performance of PyRvNN on intrinsic tasks (representation decodability and compositionality) and extrinsic tasks (plagiarism detection, readability, and memorization).
- **Chapter 6: Discussion** - summarizes the thesis contributions and revisits the hypotheses.
- **Chapter 7: Conclusion** - summarizes the thesis, and outlines potential directions for future research.

Chapter 2

Literature Review

This chapter presents a comprehensive review of neural network models for text representation learning. The focus is on architectures that generate text representations, regardless of whether this is their primary objective. Part of this chapter is based on our previously published survey ([12]).

The chapter begins with a comparative analysis (2.1) of various neural network architectures. This section systematically categorizes models and examines their characteristics, covering Shallow Models, Recurrent Models, Recursive Models, Convolutional Models, and Attention Models.

Following this, the chapter explores the desirable properties (2.2) that we want our method to have, including:

- Representation Compositionality: The ability to compose representations from subcomponents.
- Hierarchical Representation: The capability to capture multilevel text structures.
- Representation Decodability: The potential to decode original data from learned representations.
- Self-supervised Learning: Leveraging unlabeled data to learn effective representations.

The chapter concludes by identifying gaps and challenges (2.3) in existing approaches, emphasizing the need for innovative solutions. This sets the stage for the introduction of PyRv, a novel recursive neural network framework designed to address these challenges and enable efficient multilevel text representation.

2.1 Comparative Analysis

In this section we systematically compare neural models that learn text representations, identify their advantages and drawbacks, and subsequently identify the promising directions for the construction of neural text representation models. More specifically, we aim to identify neural

text representation research directions that are feasible for research teams with limited computational resources. To do so, we analyze around 50 models published in research papers and at natural language processing (NLP) conferences in the last decade.

Papers are selected following three major criteria. First, we consider papers with a reasonable number of citations. Second, we prefer models with novel ideas or with improvements in any aspect over the previous models. Third, we opt for task-independent (general) representation models. Task-independent representation models perform well on a variety of downstream tasks. Still, to ensure the integrity of this review, we include some task-dependent models that have largely influenced the field. The representation models included are mainly limited to English texts, except for models trained on machine translation tasks.

Compared to works reviewing deep learning models for NLP (e.g., [13]), this review focuses on text representation learning and, as such, reviews models from that perspective. Among similar surveys that analyzed neural network models for text representation are: a survey of neural language models [14], a survey of vector representation of meaning [15], a survey of cross-lingual word embedding models [16], and a comparability study of pre-trained language models [17]. The authors in [14] elaborate upon different architectures of the classic neural network language models and their improvements, but it lacks a deep survey and is focused on a narrower set of language models. The survey of vector representation of meaning in [15] presents a comprehensive survey of the wide range of techniques for sense representation; however, it is focused on sense representation models exclusively. The survey of cross-lingual word embedding models in [16] provides a comprehensive typology of models; however, it is framed with cross-lingual word embedding models. The work about language models in [17] proposes starting points for defining reporting standards which are applied when it comes to comparing pre-trained language models (the model architecture, number of parameters, hyperparameter tuning method and tuning time, experimental time, computational resources, training time, lexical resource information and availability, the benchmark performance on un-tuned and tuned single model); however, its survey is focused on several large pre-trained language models.

This review goes a step further, categorizing and summarizing the most prominent models created in the last decade. Additionally, we did not limit the scope to a specific architecture, task, or the representation level, instead, we opted for multi-categorization indicating the subtle differences among models, emphasizing their advantages and drawbacks.

Below, we list the datasets used for the evaluation of the representation models we cited the results for. From the extensive list of datasets, we have omitted a few that lack the comparative and systematic evaluation and are of limited interest for this study.

- WordSim-353 [18] contains pairs of words with their similarity and relatedness scores.
- SICK (Sentences Involving Compositional Knowledge) [19] consists of English sentence pairs that were annotated for relatedness and entailment.

- SNLI (Stanford Natural Language Inference) [20] contains sentence pairs manually labeled for entailment classification.
- MultiNLI (Multi-Genre Natural Language Inference) [21] dataset consists of sentence pairs annotated with textual entailment information.
- MRPC (Microsoft Research Paraphrase Corpus) [22] has pairs of sentences along with human labels indicating if the pairs share paraphrase/semantic equivalence relationship.
- STS (Semantic Text Similarity) [23] consists of pairs of sentences with labels annotating their semantic similarity.
- SSWR (Semantic-Syntactic Word Relationship) [1] contains semantic and syntactic word relations (e.g., "Einstein: scientist", "Mozart: violinist", "think: thinks", "say: says").
- MSR (Measuring Semantic Relatedness) [24] is similar to the SSWR dataset.
- SST (Stanford Sentiment Treebank) [25] has parsing trees with a sentiment classification label for each node.
- IMDB (Internet Movie Database) [26] contains reviews of movies and their sentiment polarity.
- MR (Movie Reviews) [27] is similar to the IMDB dataset, hence contains opinions on movies.
- CR (Customer Reviews) [28] consists of customer reviews about products and their rating.
- SemEval'17 [29] is a dataset from the International Workshop on Semantic Evaluation. It consists of posts from Twitter with sentiment labels.
- TREC (Text REtrieval Conference) [30] is a dataset for question classification with questions divided into semantic categories.
- SQuAD (Stanford Question Answering Dataset) [31] consists of questions posed by humans on a set of Wikipedia articles, where the answer to every question is a segment of text.
- MPQA (Multi-Perspective Question Answering) [32] contains news articles annotated for opinions, beliefs, emotions, sentiments, speculations, and other private states.
- NQ (NaturalQuestions-Open) [33] contains Google queries and their answers.
- WQ (WebQuestions) [34] is a collection of questions found through Google Suggest API, which are manually answered.
- WSJ (Wall Street Journal) consists of Wall Street Journal stories from Penn Treebank [35].
- AG News [36] contains news categorized into four classes (business, sci/tech, world, and entertainment). This version of the dataset contains only titles and disregards the rest of the articles.
- GLUE (General Language Understanding Evaluation) [37] is a collection of datasets for natural language understanding systems.
- RACE (ReAding Comprehension dataset from Examinations) [38] is a dataset for the evaluation of methods in the reading comprehension task, consisting of English exams

for Chinese students.

- SUBJ (SUBJectivity classification) [39] contains labeled subjective and objective phrases.
- WMT (Workshop on statistical Machine Translation) [40, 41] is a dataset used for machine translation tasks in WMT workshops.
- PTB (Penn Treebank) [35] is a large annotated corpus of English text, widely used for training and evaluating parsers and language models. (Note: WSJ is part of PTB).
- BLLIP-LG [42] refers to the Brown Laboratory for Linguistic Information Processing corpus, specifically the portion containing Wall Street Journal text from the Penn Treebank, often used for large-scale language modeling experiments.
- SuperGLUE [43] is a benchmark styled after GLUE with a new set of more difficult language understanding tasks.
- BIG-bench (The Beyond the Imitation Game Benchmark) [44] is a collaborative benchmark intended to probe large language models and extrapolate their future capabilities. It contains over 200 tasks.
- MTEB (Massive Text Embedding Benchmark) [45] spans 8 embedding tasks covering a total of 58 datasets and 112 languages.
- Croatian word embedding analogy and similarity datasets [46] includes a word analogy dataset (based on the original English Word2vec dataset, augmented with specific Croatian linguistic aspects), and the Croatian WordSim353 and RG65 datasets for basic evaluation of word similarities.

2.1.1 Model Categorization

To compare and contrast the models, we used five categorization criteria: (1) the representation level shows the level of linguistic units for which representations are learned; (2) the input level shows the granularity of data a model receives upon input; (3) the model type elaborates the strategy for representation learning; (4) the model supervision shows how much labeling is needed in training data; (5) the model architecture covers the main neural network architecture of the model. Each of these categorizations is described in the rest of this section. For recursive models, we used one more categorization, the parsing tree source (which we describe in Section 2.1.4).

Representation Level. Text can be represented on multiple levels. Models can learn representations for subwords, words, phrases, sentences, paragraphs, or documents, in general for any unit of text.

Subword elements include bytes, characters, and character n-grams. Downstream tasks in languages with rich morphology perform better with subword representations, and out-of-vocabulary words are easily represented [47]. Please note, subword representations only serve as the basis for the derivation of word-level representations.

Word representations are easier to implement, as generated representations are ready for use. Hence, they can be used as word models or combined into representations of larger units of text.

Sentence+ category includes representations for any larger units of text, like phrases, sentences, paragraphs, and documents. Whereas word representations can be combined to form a sentence+ representation, neural network models that are specialized for learning sentence+ representations usually produce better representations.

Input Level. Whereas the representation level describes granularity of the representations that are learned in a model, the input level describes granularity of text upon input. The granularity of text units upon input can be classified as subword-level (bytes, characters, character n-grams), or word-level. Models with subword-level input are flexible and can easily work with rare or never seen words, whereas models with word-level input are easier to implement.

Earlier models mainly used words as input (e.g., Word2Vec [1, 48]), but some (especially convolutional, and later recurrent) models used characters (e.g., CharCNN [49] and ELMo [50], and later FastText [2] used character n-grams). Recent models still use these forms of input, but when working with subword input, subword tokenization techniques improved the effectiveness of the input data.

The subword tokenization algorithm that inspired newer methods (most noticeably SentencePiece and WordPiece) is an algorithm for data compression by Gage [51] called byte pair encoding (BPE), which is used by many attention models (e.g., Transformer [52]). BPE replaces the most common pairs of consecutive bytes of data with bytes that do not occur within that data. SentencePiece (SP) [53] and WP (WordPiece) [54] are used for attention models as well (e.g., XLNet [55] and BERT [4]).

Model Type. The next criterion for differentiation of representation models considers three high-level strategies for neural network learning: autoencoding, autoregressive, and classification.

Autoencoding models learn efficient data representations in an unsupervised manner. In the strict definition of autoencoders, they learn how to compress input data into a representation, which is then decompressed back into data that should be similar to the input data. In this survey, we generalize the autoencoding definition to encompass models that learn by predicting missing input data or neighboring tokens of the input tokens.

Autoregressive models are time series models. They predict the following tokens by using previously predicted tokens as additional input. While learning to predict, autoregressive models learn text representations as well.

Classification models predict which of a set of categories a new observation belongs to. In order to correctly classify textual data, the model has to learn a representation space in which it

is easy to separate textual inputs that belong to different categories.

Model Supervision. The supervision of a model dictates how it is learned and what kind of training data it requires. The models in this review are unsupervised, supervised, semi-supervised, and un/supervised.

Unsupervised learning is the easiest one to implement from the perspective of data preparation. In the case of text representation, models need only raw text without labels. Occasionally, self-supervised models are referred to as unsupervised and vice versa.

Supervised learning requires labeled data. Data labeling is different for each task: text translation needs pairs of texts in the source and target languages, text summarization requires full texts and their summaries, classification task needs class labels, etc.

Semi-supervised learning uses both unsupervised and supervised learning. Usually, text representation models initially learn representations in an unsupervised way from a dataset containing only the raw text with no labels. Subsequently, unsupervisedly learned representations are fine-tuned through supervised learning on a specific task.

Un/supervised learning is the category we used in this survey for models that have been tested on both unsupervised and supervised tasks.

Model Architecture. Neural network models that learn text representation most frequently use shallow, recurrent, recursive, convolutional, attentive, or a combination of mentioned architectures [56]. Here we briefly introduce the architectures, whereas details are elaborated in the subsequent sections. Each model is placed in a section that corresponds with the model's general architecture. Within sections, models are ordered by shared base architecture or general principle and by publishing date.

Shallow neural network architectures in our categorization include feed-forward neural networks that have one hidden layer. Such models are capable of learning only a very shallow representation of text.

Recurrent neural network (RNN) architectures are appropriate for sequential data. Text input can be treated sequentially by reading input token by token while updating network recurrent states.

Recursive neural network (RvNN) architectures read texts recursively following the structure of parsing trees. Parsing trees can be provided with the text upon input or latently generated by the neural network. Because of that, recursive models are capable of creating high-quality sentence representations.

Convolutional neural network (CNN) architectures are well-suited for learning local patterns in text. Pooling layers learn to detect important tokens or features for a specific task it is trained on [56].

Attentive neural network architectures in this categorization include models that use at-

tention mechanisms. An attention mechanism learns dependence between input and output, or within the input elements (self-attention). For instance, it can learn which parts of the input are relevant to the task at hand or can capture long-range dependencies in text.

2.1.2 Shallow Models

Shallow models use shallow architectures (usually one hidden layer), which can learn simple representations (Word2Vec architecture is shown in Figure 2.1). In this category, we include two non-neural models that are frequently used for the same purpose as the rest of the models and are shown to be comparable [57, 58]. As a result of their simplicity, shallow models are fast to train and generalize well. Learned shallow representations can be used as input for deeper models that can learn better representations. While performing very well on simple tasks (e.g., word similarity), shallow models do not perform well on complex tasks (tasks that require a deeper understanding, e.g., question answering and summarization). To generate a phrase, sentence, or document representation, shallow representations are combined, resulting in a bag-of-words representation. The most prominent shallow models are systematized in Table 2.1 according to the input and representation level. The majority of shallow models for text representation learn word-level representations. Usually, they are autoencoding models learned by predicting the missing information upon input.

Language models, and more specifically shallow models described in this subsection, follow the distributional hypothesis, which states that the words with similar meanings occur in similar contexts [59, 60]. To learn representations that reflect the distributional hypothesis, shallow models are trained to predict neighboring words (i.e., context) around the target word and vice versa, to predict the target word from neighboring words. Shallow models mostly contain only one hidden layer in which text representations are learned. The representation being learned dictates how the input vector transforms into the output vector. The input vector is a one-hot representation of the target word or the neighboring words. In contrast, the output vector is a probability distribution that describes which words are most probably the target or the neighboring words. In the learned vector space, representations are closer together for similar words. Relationships between the words are preserved (i.e., king and man are positioned in the same way as queen and woman, and the positions of countries and their capitals preserve meaningful information).

The unsupervised model presented by Huang et al. [61] uses both local and global context via a joint training objective. It learns multiple representations per word, which is effective for homonyms and polysemous words. The meaning of a word can be ambiguous given only local context. By using a global context, the meaning of a word can be disambiguated more precisely. On the task of measuring similarity and relatedness, it achieves a Spearman correlation of 0.713 on the WordSim-353 dataset, which outperforms C&W (convolutional model described in Subsection 2.1.5) by 0.16.

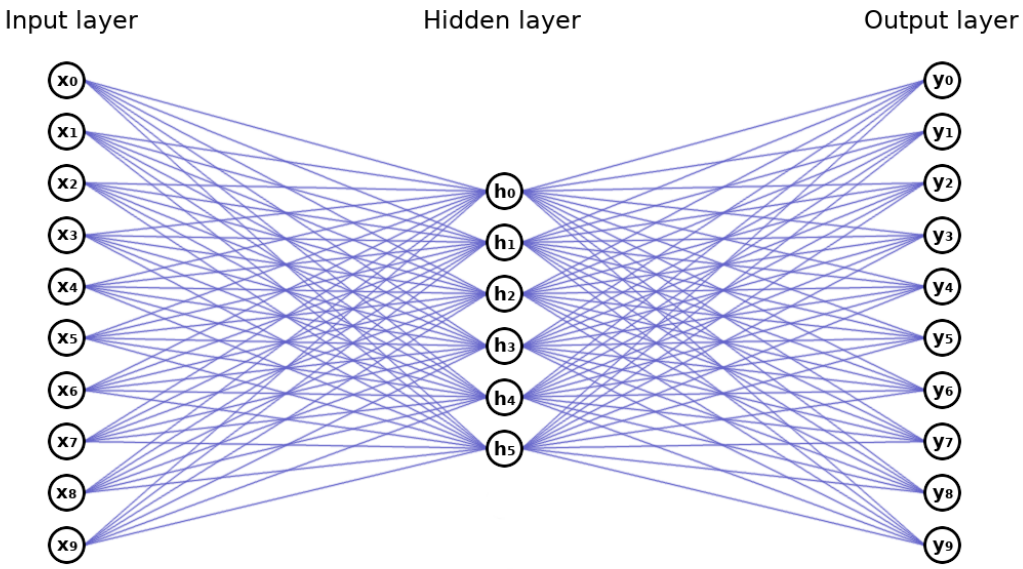


Figure 2.1: Word2Vec architecture. In addition to an input layer and an output layer, shallow architectures have a small number of hidden layers (one in this case).

Table 2.1: The categorization of shallow models by input and representation level. All listed models are unsupervised.

Model	Input	Representation
Huang et al. [61]	word	word
Word2Vec [1, 48]	word	word
Deps [62]	word	word
GloVe [3]	word	word
Doc2Vec [63]	word	sentence+
FastText [2]	n-grams	word

Word2Vec [1, 48] popularized shallow word representation using neural networks. Word2Vec has two separate autoencoding models: continuous bag-of-words (CBOW) and skip-gram. Both learn word representations through unsupervised learning. The CBOW model scans over the text with a context window and learns to predict the target word. The context window contains n preceding and n succeeding words around the target word. The skip-gram model conversely predicts the words in the context from the target word. Word2Vec neural network has only one hidden layer, and word representations are extracted from that layer. The accuracies for the skip-gram model on the SSWR dataset for semantics and syntax are 50% and 55.9%, respectively, outperforming the model by Huang et al. by 36.7% and 44.3%. The accuracy on the MSR dataset for the skip-gram model is 56%.

Whereas Word2Vec uses local neighboring words as context, its extension Deps [62] uses neighboring words in dependency parse trees to learn word representations. Deps generalizes skip-gram to include arbitrary contexts and uses dependency-based contexts derived from parse trees.

GloVe [3] directly captures global corpus statistics through unsupervised learning. It is inspired by neural models but is not a neural model itself. GloVe combines global matrix factorization and local context window methods through a bilinear regression model. By doing so, it trains by using the co-occurrence matrix and learns word representations in a way that it can predict co-occurrence probability ratios. An example was given in [3]. Let $i = ice$ and $j = steam$, if $k = solid$, we expect the ratio P_{ik}/P_{jk} to be large (P_{xy} is probability of words x and y occurring together). If $k = gas$, the ratio should be small. For words that are related to both *ice* and *steam*, or to neither, the ratio should be closer to 1. This is used instead of raw probabilities because ratios distinguish relevant words from irrelevant words better. It is shown, that mathematically GloVe is similar to Word2Vec [57, 58]. GloVe slightly outperforms Word2Vec on multiple tasks. On the SSWR dataset, GloVe has 75% total accuracy (81.9% semantics and 69.3% syntax), which is an improvement of 9.4% over Word2Vec. GloVe is evaluated on five word similarity tasks (including WordSim-353 where it achieves a 0.759 Spearman correlation), and in each evaluation, GloVe outperforms Word2Vec.

The shallow models above can learn representations for words. Doc2Vec [63] is an extension of Word2Vec that can learn representations for documents (or parts of texts). While predicting context words of a target word, Doc2Vec receives the target word and the document ID upon input. Through learning, as it learns useful relations between documents and words, it learns not only representations for words but documents as well. The accuracy on the SST dataset is 87.8% for binary classification, and 48.7% for fine-grained classification, outperforming the recursive model by Socher et al. [25] by 2.4% and 3%, respectively. Accuracy on the IMDB dataset is 92.58%.

All models described above use word-level input, but subword-level input can be beneficial. FastText [2] is an unsupervised non-neural model inspired by neural models. Similarly

to Word2Vec, it has two separate models, CBOW and skip-gram. It learns representations for character n-grams, and each word is represented as a bag-of-character n-grams. Previous models were limited to assigning a distinct vector to each word. Representations on the subword level are shown to perform better for morphologically rich languages and rare words. On the WordSim-353 dataset, FastText has a Spearman correlation of 0.71 (similar to Word2Vec's 0.72). On the SSWR dataset, it achieves 77.8% semantic and 74.9% syntactic accuracy, performing on the semantic task similarly as Word2Vec and outperforming it on the syntactic task by 4.8%.

2.1.3 Recurrent Models

Models like Word2Vec are insensitive to word order, they only capture the relations between words—if a word is in the context of another word. The ordering of words in a text is meaningful, since different orderings of words can denote different meanings. RNNs process input as a sequence and learn ordered representations from a text, hence they are well suited for learning representations of longer linguistic units like phrases, sentences, paragraphs, or documents. While tokens from the input sequence are processed, the history of all the previous tokens is preserved as a state in the neurons (Figure 2.2). The most influential RNN models are listed in Table 2.2 according to their input level, representation level, and model supervision. The majority of RNNs for text representation are autoregressive models. Output tokens are generated by taking previous output tokens as an additional input (together with the representation of the encoded sequence, currently being learned).

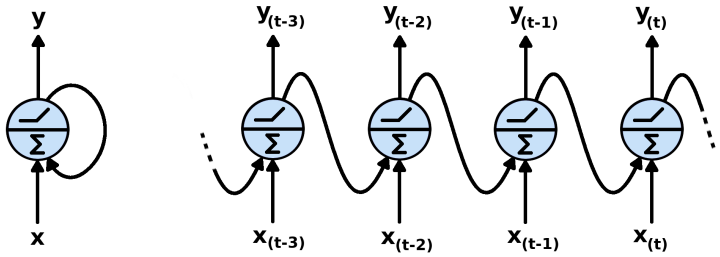


Figure 2.2: Recurrent architecture and the unfolding in time. Recurrent nodes have connections which lead to the next node, and connections which loop back to the same node. The unfolding in time shows the same node in three consecutive iterations.

In shallow feedforward models, the hidden layer and ultimately the output is affected only by the current input. In recurrent models, the hidden layer (or layers) and the output are affected not only by the current input but also by the previous inputs. That is achieved by using recurrence, where the recurrent layers propagate information not only to the next layer but back to their input as well. To train neural networks, gradients are calculated and used to update parameters in a direction that depends on the loss function. With longer texts, RNNs have a vanishing (and exploding) gradient problem as the gradient is calculated through a long chain

of recurrence. That problem is reduced by long short-term memory (LSTM) unit and its variants like gated recurrent unit (GRU), which have gates that learn what information is important and, as such, has to be propagated more strongly. LSTM's architecture is shown in Figure 2.3. It has a cell state (horizontal line on the top), which is used for memory. Gates below control information flow by selectively forgetting old and memorizing new information relevant for the current task.

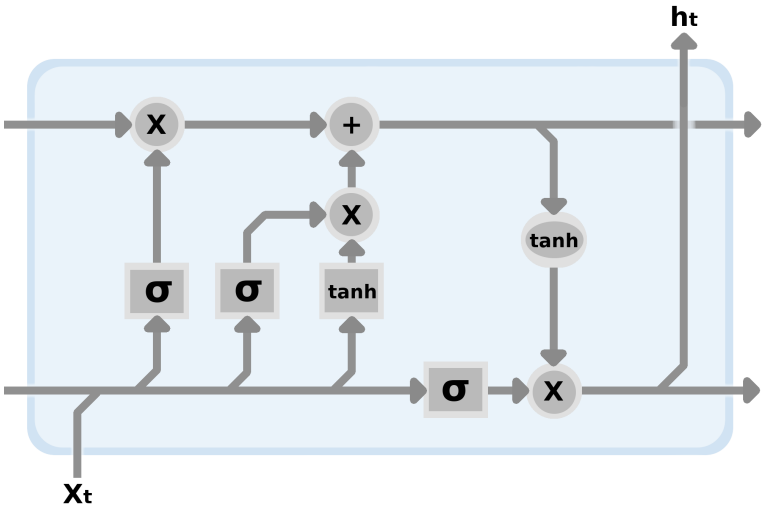


Figure 2.3: The LSTM cell. Variables x_t and h_t are input and output, respectively, at time t. Squares with "σ" or "tanh" represent layers, whereas ovals with "X", "+", or "tanh" represent pointwise operations.

A supervised encoder–decoder model introduced by Cho et al. [64] uses two RNNs. One RNN encodes a sequence of symbols into a representation vector, and the other decodes the representation into another sequence of symbols. The sequence of symbols can be a phrase, part of a sentence, a sentence, or a longer linguistic unit. The model is trained on a machine translation task. The model achieves a 34.54 BLEU score on the English–French translation task from the WMT'14 dataset, which is an improvement of 1.24 points over the baseline method (Moses).

With the addition of LSTM units, the Seq2Seq [65] model improves performance on longer sequences over basic RNN models. It is learned in a supervised way on a machine translation task. Seq2Seq uses LSTM units in recurrent layers. This model can produce representations for phrases and sentences. Seq2Seq scores 34.81 BLEU on the English–French translation task from the WMT'14 dataset, outperforming the recurrent model by Bahdanau et al. [74] by 6.36 points.

Whereas most recurrent models learn text representations by sequentially predicting words (autoregressive models), Skip-Thoughts [66] is an unsupervised encoder–decoder model that learns to reconstruct the surrounding sentences of an encoded sentence, and in that aspect, it is similar to other autoencoding models. It uses encoder with GRU activations and an RNN decoder with a conditional GRU. Skip-Thoughts learns representations for sentences and performs just as well as the LSTM approach. On the SICK semantic relatedness subtask, Skip-Thoughts

Table 2.2: The categorization of recurrent models by input level, representation level, and supervision.

Model	Input	Repres.	Supervision
Cho et al. [64]	word	sentence+	supervised
Seq2Seq [65]	word	sentence+	supervised
Skip-Thoughts [66]	word	sentence+	unsupervised
RCNN [67]	word	sentence+	supervised
Li et al. [68]	word	sentence+	unsupervised
FastSent [69]	word	sentence+	unsupervised
CoVe [70]	word	word	supervised
Akbik et al. [71]	character	word	unsupervised
ELMo [50]	character	word	unsupervised
Subramanian et al. [72]	word	sentence+	semi-supervised
LASER [73]	BPE	sentence+	supervised

performs with the Pearson correlation 0.8655 and the Spearman correlation 0.7995, which is similar to Tree-LSTM (recursive model described in Subsection 2.1.4). Accuracy on the MRPC dataset is 75.8% and the F1 score is 83.

RCNN [67] (recurrent convolutional neural network) is a supervised bidirectional RNN (BiRNN) with a pooling layer after the BiRNN layer. Sentence representations are learned through the BiRNN, which scans over texts upon input in both directions (forward and backward), whereas normal RNNs scan texts only in the forward direction. The pooling layer learns to select the most important words for a text classification task. Fine-grained classification accuracy on the SST dataset is 47.21%, similar to the accuracy achieved by Doc2Vec.

The LSTM autoencoder introduced by Li et al. ([68]) hierarchically builds embeddings for paragraphs. It is trained to hierarchically encode an embedding for a paragraph from embeddings for sentences and words and then decode this embedding back into the original paragraph text.

A simple variant of the Skip-Thoughts model FastSent [69] is an unsupervised model with GRUs. FastSent receives a bag-of-words representation of a sentence upon input and predicts adjacent sentences, which are also represented as bag-of-words representations. FastSent outperforms Skip-Thoughts on unsupervised tasks: the Spearman correlation for the SICK dataset is 0.61, which is an improvement of 0.04 over Skip-Thoughts, whereas for STS tasks the Spearman correlation is 0.63, which is an improvement of 0.36 over Skip-Thoughts. FastSent underperforms Skip-Thoughts on six supervised sentence classification tasks (on MRPC, MR, CR, SUBJ, MPQA, and TREC dataset). Overall, FastSent has an average accuracy of 77.92% on

the supervised tasks, under-performing Skip-Thoughts by 5.83%.

CoVe [70] (Context Vectors) is a supervised bidirectional LSTM (BiLSTM) encoder with an attentive LSTM decoder trained for machine translation. Word representations are extracted from the BiLSTM encoder. The attentive LSTM decoder first uses the LSTM to produce a hidden state, after which the attention mechanism computes the relevance between each encoding time-step and the current state. CoVe is evaluated on seven classification tasks (on SST-2, SST-5, IMDB, TREC-6, TREC-50, SNLI, and SQuAD dataset). Overall, its average accuracy is 84.8%, which outperforms GloVe by 1.8% and Skip-Thoughts by 2.2%.

Models mentioned previously produced one vector for each subword, word, or sentence, which cannot account for polysemy (multiple meanings of the same word). ELMo [50] (Embeddings from Language Models) is an unsupervised model that uses BiLSTM and can learn polysemy. ELMo's word representations are learned functions of the internal states of the BiLSTM. In different contexts, the same word has different representation vectors, which can account for different word meanings. ELMo achieves 54.7% fine-grained classification accuracy on the SST dataset, outperforming CoVe by 1.8%.

The model introduced by Akbik et al. [71] is an unsupervised character-level language model. The word representations are extracted from BiLSTM hidden states. As word representations are contextualized by their surrounding text, the same word can have different meanings depending on its context, which can model polysemy. Word representations are concatenations of hidden states of the forward pass through a sentence until the last character in the word, and the backward pass through a sentence until the first character in the word. This model has 93.09% accuracy on the CoNLL'03 dataset, outperforming ELMo by 0.87%.

A semi-supervised model introduced by Subramanian et al. [72] uses bidirectional GRU (BiGRU) for sentence representation multi-task learning. The tasks that the model is trained on are skip-thought, machine translation, constituency parsing, and natural language inference. The authors demonstrate that multi-task learning leads to consistent improvements over previous single-task methods (some of which are FastSent, Skip-Thoughts, and CNN-LSTM, which is described in Subsection 2.1.5).

LASER [73] (Language-Agnostic SEntence Representations) learns joint multilingual sentence representations for 93 languages through a supervised machine translation task. The learned representations are general with respect to the input language and the task. It uses the BiLSTM encoder to generate a sentence representation, and the LSTM decoder to generate a sentence in a target language. The encoder receives no information about what the input language is, due to which it learns language-independent representations. The model is trained by translating all 91 languages to both English and Spanish languages. On the XNLI (cross-lingual natural language inference) dataset [75], with 70.19% average accuracy on 15 languages, this model archives similar results as BERT (attention model described in Subsection 2.1.6).

2.1.4 Recursive Models

Recurrent models learn representations with ordered information; however, recursive neural networks go one step further and learn deeply structured information like trees. Recursive neural networks process inputs in a recursive fashion through a tree structure (example given in Figure 2.4). Each node in a tree for a subword, word, phrase, sentence, or a larger unit of text is associated with a respective representation. The most influential RvNN models are shown in Table 2.3 with their input level, model supervision, and tree source (given, learned, or latent). Most of the RvNNs for text representation are autoencoding or classification models. The representation for each node can be learned either by an autoencoding method similar to the shallow models or through a classification task.

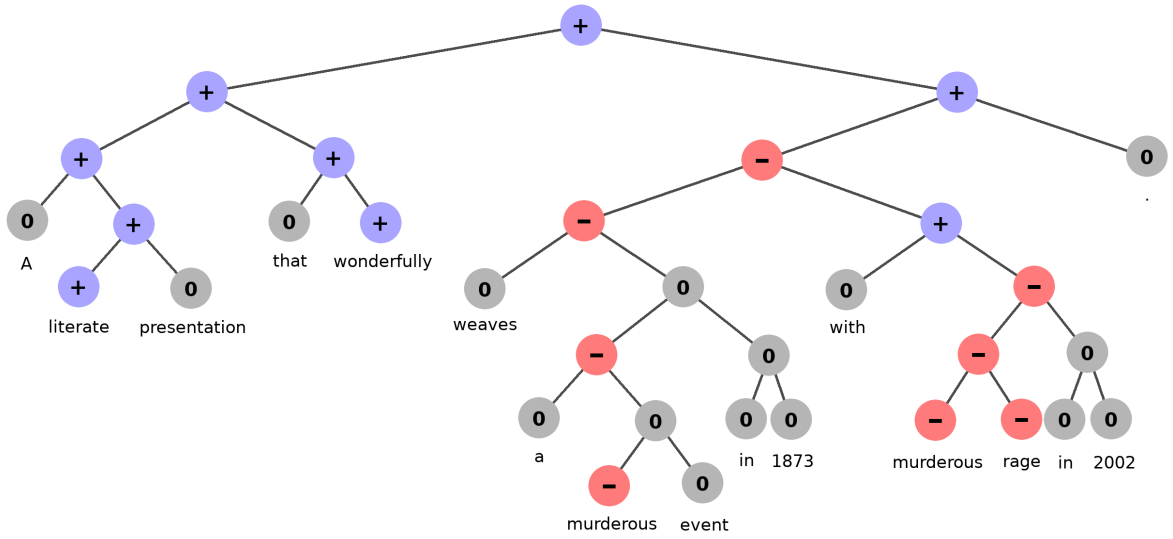


Figure 2.4: A parsing tree of a recursive neural network predicting word sentiment classes. The leaf nodes are input tokens, all the other nodes are representations of the combination of the child nodes. The root node is representation of the entire input text.

Recursive neural networks are generalized recurrent neural networks. Whereas recurrent neural networks read input in a linear chain, recursive neural networks read input in any hierarchical structure. Representations are merged recursively throughout the tree structure, where merging combines two or more representations from the lower level into one higher-level representation. Hence, learning of the recursive neural representations enables combining representations of more granular linguistic units into larger linguistic units (e.g., from characters to sentences, or from words to documents). The merging process is repeated recursively until the root node is reached. The root node represents the whole input sequence (Figure 2.4) and leaf nodes are input tokens. The tree structure can be given upon input, learned from labeled texts (texts paired with their parse trees), or implicitly generated by a neural network with no direct supervision (latent trees).

RAE [76] (Recursive AutoEncoders) introduced an architecture based on recursive au-

Table 2.3: The categorization of recursive models by input level, supervision, and parsing tree source. All the listed models learn sentence+ representations. Un/supervised supervision represents both unsupervised and supervised learning.

Model	Input	Supervision	Tree
RAE [76]	word	un/supervised	latent
MV-RNN [77]	word	un/supervised	given
morphoRNN [78]	morpheme	unsupervised	given
RNTN [25]	word	unsupervised	given
AdaSent [79]	word	supervised	latent
Tree-LSTM [80]	word	supervised	given
RL-SPINN [81]	word	reinforcement	latent
ST-Gumbel [82]	word	unsupervised	latent
DIORA [83]	word	unsupervised	latent
CRvNN [84]	token	supervised	latent
Unified TREE-LSTM [85]	word	supervised	latent
TGs [86]	WP	un/supervised	learned
Fast-R2D2 [87]	WP	un/supervised	latent
ReCAT [88]	WP	un/supervised	latent

toencoders for sentence-level prediction of sentiment label polarity. An autoencoder is a type of neural network that learns representations of input data in an unsupervised manner. The encoder part of the network encodes the input into a dense representation, while the decoder part of the network decodes that representation into output (that should be as close as possible to the input). While RAE can learn text representation through supervised learning of sentiment distribution, it can learn text representation through unsupervised learning as well. An important RAE feature is the ability to learn latent parsing trees, meaning that they are not learned or given upon input, but generated by concatenating neighboring pairs of words or phrases and combining the ones with the lowest reconstruction error (in the autoencoder) into parent nodes. On sentiment classification, RAE achieves an accuracy of 77.7% on the MR dataset and 86.4% on the MPQA opinion dataset, which is a slight improvement over Tree-CRF [89].

MV-RNN [77] (Matrix-Vector recursive neural network) learns vector and matrix representations for every node in the tree (phrases and sentences). The vector captures the meaning of the node, whereas the matrix captures how it changes the meaning of neighboring words or phrases. Supervised training for each of the tasks is done by adding a softmax classifier on top of each parent node. The model receives parse trees from a parser. It achieves 79% accuracy on the MR dataset, which is an 1.3% improvement over RAE.

Model introduced in [78], morphoRNN, focuses on constructing representations for morphologically complex words by recursively building morphological trees from the main morpheme and appended affixes in a predetermined order. Notably, morphoRNN excels in representation of rare and complex words, outperforming many contemporary models.

Another significant recursive model in this era was the Recursive Neural Tensor Network (RNTN) [25]. RNTN adopts a supervised training approach, where the representation in each node of the tree serves as a feature for a sentiment classifier. During this time, RNTN stood out as the sole model capable of effectively capturing the nuanced effects of negation across various tree levels.

Whereas other RvNNs process inputs as trees, AdaSent [79] process inputs in a hierarchical fashion (pyramid structure) which does not require parse trees upon input. The pyramid structure forms a hierarchy of representations from words to sentences. The model is trained on supervised classification tasks. Classification uses the hierarchy of representations as input. The hierarchy of representations is first summed on each level of the hierarchy, and then a gating network calculates a weight for each of the levels. AdaSent has 83.1% accuracy on the MR dataset, outperforming RAE by 5.4%. Its accuracy on the MPQA opinion dataset is 93.3%, outperforming RAE by 6.9%. It outperforms MV-RNN as well on the MR dataset by 6.9% with a 93.3% accuracy.

Tree-LSTM [80] introduced a generalization of LSTM to tree-structured network topologies. The model is trained on a supervised sentiment task and it requires parse trees upon input. LSTM architectures can process only sequential information, Tree-LSTM extends LSTM in a

way which can process structured information. Tree-LSTM performs better than an RvNN with basic RNNs because of the same reasons LSTM outperforms RNN. It has 51% fine-grained classification accuracy on the SST dataset, outperforming RAE, and MV-RNN by 7.8% and 6.6%, respectively.

RL-SPINN [81] (Reinforcement Learning SPINN) uses reinforcement learning to learn sentence representations. RL-SPINN's architecture is based on the SPINN model (Stack-augmented Parser-Interpreter Neural Network) [90]. Parse trees are latent, meaning that the tree structures are not learned from labeled data or given upon input, but are unsupervisedly optimized for the downstream task. RL-SPINN performs similarly as Tree-LSTM and DCNN (the convolutional model described in a later subsection) on the SST dataset with an 86.5% accuracy for binary classification. It has a 0.359 mean squared error on the SICK dataset, under-performing Tree-LSTM by 0.067.

ST-Gumbel [82] (Straight-Through Gumbel-softmax estimator) is a Tree-LSTM modification as well. ST-Gumbel uses latent trees that are computed with a composition query vector that measures the validity of a composition. Text representation is learned through unsupervised training. On the SST dataset, it achieves 53.7% accuracy for fine-grained classification, and 90.7% accuracy for binary classification, outperforming Tree-LSTM by 2.7% in both fine-grained and binary classification.

DIORA [83] (Deep Inside–Outside Recursive Autoencoders) is an unsupervised model that learns to predict each word in a sentence while being conditioned on the rest of the sentence. It considers all possible trees over the input sentence. The CYK (Cocke–Younger–Kasami) parsing algorithm extracts the highest-scoring parse tree (latent trees). The architecture consists of recursive autoencoders. The training is done through inside–outside passes, where the inside representations of each node are encoded by using the children of the nodes, and outside representations of each node are encoded using only the context of the node subtree. On WSJ dataset, DIORA has a 55.7 F1 score, which is an increase of 32.9 with respect to ST-Gumbel, and an increase of 42.5 with respect to RL-SPINN.

More recent advancements have focused on improving the differentiability, scalability, and integration of recursive principles with Transformer architectures. The Continuous Recursive Neural Network (CRvNN) [84] addresses limitations of traditional RvNNs by incorporating a continuous relaxation to the induced latent tree structure, making it backpropagation-friendly. CRvNN demonstrated strong performance on tasks like sentiment analysis (SST2 binary accuracy: 88.36%) and natural language inference (SNLI accuracy: 85.12%), performing comparably to or better than prior latent structure models like ST-Gumbel on SNLI.

Simoulin and Crabbé [85] proposed a Unified TREE-LSTM that jointly learns its composition function and its structure using a Biaffine dependency parser to induce trees. This model achieved a Pearson correlation of 87.0 on the SICK-R dataset, comparable to a BERT base model (87.3), and 85.0% accuracy on SNLI. These results highlight the potential of jointly

learned parsing and composition.

Transformer Grammars (TGs) [86] augment Transformer language models with recursive syntactic compositions via special attention masks based on linearized phrase-structure trees. TGs achieved a F1 score of 93.7 on PTB, slightly outperforming an RNNG (93.3), and demonstrated strong syntactic generalization (SG score of 82.5 on BLLIP sentence-level).

The Fast-R2D2 model [87] introduced a pretrained recursive neural network using a pruned CKY algorithm and a fast top-down unsupervised parser. It showed strong results in unsupervised grammar induction, achieving an F1 of 53.88 on PTB WSJ (word-piece level), outperforming DIORA (51.4 at word-level). On the GLUE SST-2 task, Fast-R2D2* (using the parser's tree) achieved 90.71% accuracy, comparable to ST-Gumbel (90.7%).

ReCAT [88] is a recursive composition augmented Transformer that uses Contextual Inside-Outside layers to learn multi-grained span representations without gold parses. For unsupervised grammar induction on PTB, ReCAT achieved an F1 of 65.00, surpassing Fast-R2D2* (53.88) and DIORA (55.7). On the GLUE SST-2 task, ReCAT achieved 88.65% accuracy.

2.1.5 Convolutional Models

Recurrent and recursive neural networks are a good fit for textual modality (as they are appropriate for sequential data), whereas convolutional neural networks (CNN) are originally used for 2-D data, and as such had to be modified to fit textual modality. CNNs proved successful with visual data (images and videos); however, they can also be used for text representation learning (the architecture is shown in Figure 2.5). CNNs by their nature learn to abstract input data through multiple convolutional levels and detect specific patterns on each level (e.g., textures or borders of objects in images, syllables, or word n-grams in text). The most representative CNN models used for learning text representations are shown in Table 2.4 with their input level, representation level, and model supervision. CNNs for text representation were in the beginning mostly classification models, but later CNN models were combined with other architectures, typically with autoencoding or autoregressive models.

Convolutional models for images learn to recognize patterns from the lowest level (e.g., edges, corners) to the highest level (e.g., cat, dog, house). Similarly, convolutional models for text learn patterns like syllables, syntax, and phrases. The notable difference between convolutions for images and text is the dimensionality of the convolutions. Convolutions for images are two-dimensional, and convolutions for text are mostly one-dimensional. In Figure 2.5, input is connected to the convolution layer. The convolution layer in this example has four filters. Each filter has a kernel (a matrix of weights) trained to detect patterns that are of importance to the task at hand. Kernel slides over the values from the previous layer (in this example, input values) and outputs an activation that corresponds to how much that region of image or text fits the pattern. Next, the convolution layer is connected to the fully connected layer. Whereas recurrent and recursive models can easily read inputs of undefined length, convolutional models

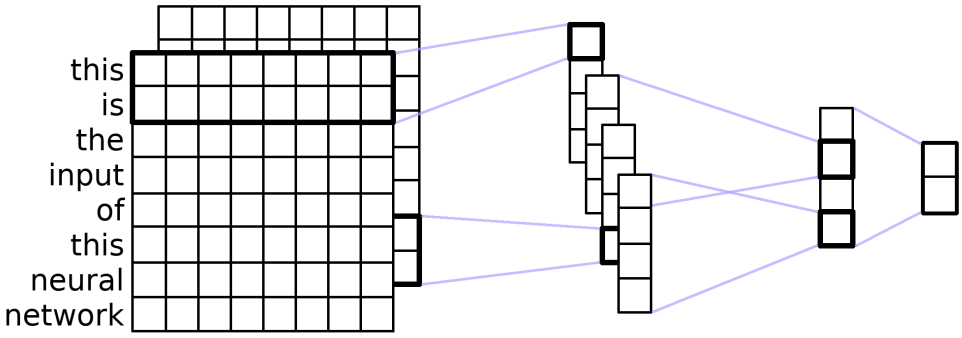


Figure 2.5: Convolutional architecture. A convolution has multiple filters, and each filter has a kernel (a matrix of weights) that is being trained. The kernel slides over the values from the previous layer, producing values that are sent to the next layer. Each filter learns to recognize a different pattern.

need pooling layers to process inputs of different sizes. The pooling layer lowers the previous layer's dimensionality by forwarding (pooling) the maximum or average of each region.

The model introduced by C&W [91] (Collobert and Weston) uses a single convolutional neural network architecture that, given a sentence upon input, outputs part-of-speech tags, chunks, named entity tags, semantic roles, semantically similar words, and the likelihood that the sentence makes sense. All of these tasks are trained jointly, resulting in an implicitly trained language model. While the language model is learned unsupervisedly, downstream tasks are learned supervisedly. During learning, all the tasks share weights, resulting in better language representation. Note that individual word representations can be extracted from the trained network.

Table 2.4: The categorization of convolutional models by input level, representation level, and supervision. Un/supervised supervision represents both unsupervised and supervised learning.

Model	Input	Repres.	Supervision
C&W [91]	word	word	semi-supervised
DCNN [92]	word	sentence+	supervised
CharCNN [49]	character	word	unsupervised
ByteNet [93]	character	sentence+	un/supervised
CNN-LSTM [94]	word	sentence+	unsupervised
CDWE [95]	word	word	supervised

DCNN [92] (dynamic convolutional neural network) is a convolutional model supervised for each downstream task separately. The architecture consists of multiple one-dimensional convolutional layers and dynamic k-max pooling layers, which induce a latent tree over the input sentence. The last layer, which is fully connected, contains the representation of the

input sentence. This model achieves an 48.5% and 86.8% accuracy for fine-grained and binary classification on the SST dataset, outperforming the baseline method (Naive Bayes) by 7.5% and 5%, respectively. On the SST dataset, its performance is similar to the recursive model RL-SPINN described in Subsection 2.1.4.

CNNs are a good fit for character-level input, which is why CharCNN [49] (character-level convolutional neural network) is an unsupervised model that relies only on character-level inputs to learn word representations. In CharCNN architecture, single-layer character-level CNN reads the input, and LSTM generates the output. Each of the filters in the CNN detects a single character n-gram. The perplexity of this model on the English Penn Treebank test set [35] is 78.9, which is similar to the perplexity of the recurrent model by Zaremba et al. [96] that contains a double number of parameters.

Similarly to CharCNN, ByteNet [93] relies only on character-level inputs. It is unsupervised for the language modeling task and supervised for the machine translation task. ByteNet implements two mechanisms: the first mechanism allows the preservation of the temporal resolution of the sequences, and the second mechanism allows the network to process source and target sequences of different lengths. The model runs in a time that is linear to the length of the sequence. On the WMT'14 and WMT'15 dataset, it achieves a 23.75 and 26.26 BLEU score, respectively, outperforming the recurrent model by Chung et al. [97] by 2.42 and 2.81 points, respectively.

CNN-LSTM [94] is an unsupervised encoder–decoder architecture with a CNN encoder and LSTM decoder. It has two modes of training: autoencoder and future predictor. The autoencoder reconstructs the input sentence, whereas the future predictor predicts the rest of the sentence. The average accuracy for five-sentence classification tasks (on MR, CR, SUBJ, MPQA, and TREC dataset) is 87.1%, which is an improvement of 8% over FastSent.

By using deconvolution (transposed convolution) CDWE [95] (Convolution–Deconvolution Word Embedding) can learn multiple vectors for each word. CDWE is supervised for each downstream task. It generates multiple prototypes in the deconvolution layer for each word, which can model polysemy. After that, according to the context, a proper prototype for a word is selected. The average accuracy for three sentence classification tasks (on TREC, AG News, and MR dataset) is 90.9%, which is an improvement of 13% over CharCNN.

2.1.6 Attention Models

The architectures from the previous subsections (shallow, recurrent, recursive, and convolutional) are either very general or are originally developed for a different modality of input data. The neural attention mechanism was created to capture long-range dependencies and was inspired by how humans read and understand longer texts [98]. Long-range dependencies exceed the dependencies captured within the limited boundaries of context windows as used in shallow models. RNNs are better at capturing long-range dependencies but are not as good as attention

models since they are focused equally on every word in the input. Conversely, neural networks with attention can focus on parts of a text that are more important for a current task, and thus perform better with long texts (an example is shown in Figure 2.6). The most influential attention models are shown in Table 2.5 with their input level and model supervision. As attention mechanisms can be implemented in a wide range of architectures, the models using them can be autoencoding, autoregressive, or classification models. Models that greatly gain upon the attention mechanism (e.g., Transformer, which is described later) are mostly autoencoding or autoregressive models.

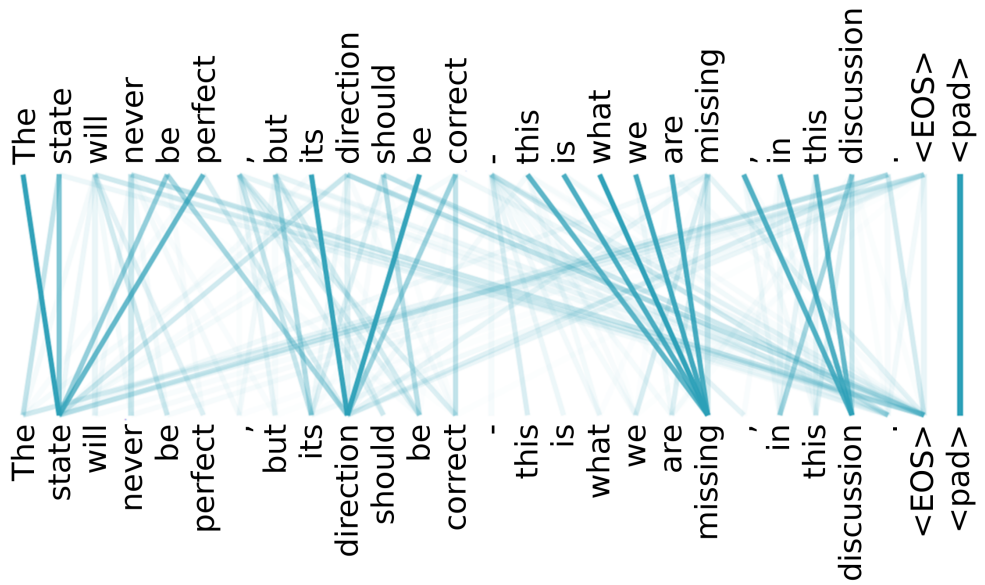


Figure 2.6: A visualization of a learned self-attention head on a sentence. The visualization shows learned relations between the words this self-attention head has learned. Each head learns a different kind of relations between the words.

Attention mechanism learns which parts of the input vector are important for each part of the output vector (e.g., while translating a sentence from one language to another, each output word depends more on some parts of the input than the other parts, so it makes sense to pay more attention to those parts when deciding what the next word in the output is). Self-attention is similar, and it learns dependency as well, but between words in the input rather than between the input and output (e.g., when predicting the next word of the sentence "I am eating a green", it makes sense to pay more attention to words "eating" and "green" to predict the next word "apple"). The attention mechanism has multiple heads, which are similar to filters in convolutions. Each head learns to pay attention in different ways. One head can learn to attend to words that help decide the tense of the next word, whereas another head can learn to attend to entity names.

HAN [99] (Hierarchical Attention Network) is a supervised hierarchical attention network for a document classification task. HAN architecture is based on BiGRUs and attention mechanisms on the word and sentence level. First HAN learns sentence representations and then

aggregates those representations into a document representation. The attention mechanism enables it to attend to different words when building sentence representations, and to different sentences when building document representations. On the IMDB dataset, HAN achieves 49.4% accuracy, outperforming Doc2Vec by 15.3%.

Table 2.5: The categorization of attention models by input level and supervision. All listed models learn sentence+ representations. Un/supervised supervision represents both unsupervised and supervised learning.

Model	Input	Supervision
HAN [99]	word	supervised
Kim et al. [100]	word	unsupervised
Lin et al. [101]	word	supervised
Transformer [52]	BPE	semi-supervised
GPT [5]	BPE	semi-supervised
DiSAN [102]	word	supervised
Bi-BloSAN [103]	word	supervised
ReSAN [104]	word	supervised
BERT [4]	WP	unsupervised
Liu et al. [105]	word	supervised
GPT-2 [106]	BPE	unsupervised
XLM [107]	BPE	un/supervised
Star-Transformer [108]	word	supervised
Transformer-XL [109]	char. or word	unsupervised
MASS [110]	BPE	unsupervised
SBERT [111]	word	un/supervised
XLNet [55]	SP	unsupervised
ALBERT [112]	WP	unsupervised
SpanBERT [113]	WP, word, span	unsupervised
REALM [114]	WP	semi-supervised
ELECTRA [115]	WP	unsupervised

Authors Kim et al. [100] introduce structured attention networks (a generalization of the basic attention procedure), which can attend to partial segments or subtrees while learning representations. This approach does not consider non-projective dependency structures and its

inside–outside algorithm is difficult to parallelize. To overcome the parallelization problem, Liu et al. [105] implicitly consider non-projective dependency trees and make each step of the learning process differentiable. Evaluated on the SNLI dataset, this model achieves 86.5% accuracy, which outperforms the recursive model SPINN [90] by 3.3%.

The supervised model introduced by Lin et al. [101] instead of a vector to represent text uses a 2-D matrix. Each row in that matrix representation is attending to a different part of the sentence. The model has two parts, the first part is a BiLSTM, and the second part is a self-attention mechanism. Each LSTM hidden state provides only a short-term context information around each word, whereas self-attention captures longer-range dependencies by summing the weighted LSTM hidden states. Evaluated on the SNLI dataset, this model achieves 84.4% accuracy slightly under-performing the model by Kim et al.

Most of the sequence translation models use recurrent or convolutional neural networks with an encoder and a decoder. Some also connect the encoder and decoder through an attention mechanism. Transformer [52] is a sequence translation model that is used by many models created subsequently. Transformer is a neural network architecture based exclusively on attention mechanisms, dispensing recurrent or convolutional layers. Self-attention mechanisms in Transformer show the ability to correctly resolve anaphora. The goal of some of the convolutional neural network models is to reduce sequential computation by relating signals between two positions in a sequence. Hence, in CNN models the number of computations grows linearly or logarithmically with the distance between those positions. In Transformer architecture, the number of computations is constant, which makes it easier to learn dependencies between distant positions. On the WMT'14 dataset (English–German translation task), Transformer achieves a 28.4 BLEU score, outperforming ByteNet by 4.65 points.

Similarly to Transformer, DiSAN [102] (Directional Self-Attention Network) and ReSAN [104] (Reinforced Self-Attention Network) are based only on attention mechanisms without any RNN or CNN structure. DiSAN is composed of a directional self-attention, followed by a multi-dimensional attention that creates a vector representation. It achieves 85.62% accuracy on the SNLI dataset. It has a 90.8% average accuracy on four-sentence classification datasets (on CR, MPQA, SUBJ, and TREC dataset), outperforming Skip-Thoughts by 2.2%. ReSAN integrates both soft and hard attention into one model, outperforming DiSAN on the SNLI dataset by 0.7%. On the SICK dataset it achieves a Spearman correlation of 0.8163, close to the DiSAN's correlation of 0.8139. Star-Transformer [108] is a lightweight alternative to Transformer which reduces model complexity by sparsification. Sparsification replaces the fully connected structure with a star-shaped structure, which reduces the number of connections from n^2 to $2n$. On the SNLI dataset, it performs with 86% accuracy, outperforming Transformer by 3.8%. Whereas Transformer takes 49.31 ms on test time, Star-Transformer takes 10.94 ms.

BERT [4] (Bidirectional Encoder Representations from Transformers) is a deep bidirectional Transformer. Deep bidirectional means that it is conditioned on every word in the left

and right contexts at the same time. It does so by masking some percentage of the input tokens at random and then predicts those masked tokens. BERT is an unsupervised model that learns sentence representations. It achieves an 82.1% average accuracy on a subset of GLUE datasets, which is a 7% improvement over GPT (a model mentioned below). ALBERT [112] (A Lite BERT) optimizes BERT by lowering memory consumption and increasing the training speed. Evaluated on five datasets (on SquAD1.1, SQuAD2.0, MultiNLI, SST-2, and RACE dataset), ALBERT performs with an 88.7% average accuracy, outperforming BERT by 3.5%. SpanBERT [113] extends BERT by masking random spans (sections of text) whereas BERT is masking random tokens, and by training the representations to predict the entire content of the masked span. It has an 82.8% average accuracy on a subset of GLUE datasets, which is an improvement of 2.4% over BERT.

GPT [5] (Generative pre-trained Transformer) is a semi-supervised model based on Transformer. Its training is unsupervised for pre-training of the language model and supervised for fine-tuning to a downstream task. To avoid interventions into the architecture for each of the downstream tasks, they convert structured inputs into an ordered sequence that GPT can process. Largely following GPT architecture, GPT-2 [106] learns byte sequence representations through unsupervised training. The main task of this model is language modeling. GPT-2 performs well with anaphora. It is the state-of-the-art at the time of the publishing of the GPT-2 paper, outperforming other models in seven out of eight tested language modeling datasets.

Authors in XLM [107] (cross-lingual Language Model) introduced two methods to learn cross-lingual language models based on Transformer. One method is supervised and uses monolingual data, the other method is unsupervised and uses parallel texts in each of the languages. XLM has a 0.69 Pearson correlation on the SemEval'17 dataset, outperforming the model by Conneau et al. [116] by 0.04.

Standard Transformer learns short dependencies, and context becomes fragmented because of the segmentation of input contexts while training. Transformer-XL [109] is based on Transformer architecture with an added segment-level recurrence mechanism. Whereas the basic Transformer can learn dependencies of length only equal to the segment length, Transformer-XL can learn long-range dependencies by using its recurrence mechanism. It can learn dependencies that are 80% longer than dependencies in RNNs and 450% longer than dependencies in basic Transformers. Transformer-XL also solves the problem of context fragmentation, which appears in Transformer because of its segmentation. Similarly to Transformer-XL, Bi-BloSAN [103] (Bi-directional Block Self-Attention Network) captures long-range dependency. It does so with a segment-level self-attention mechanism. When evaluated on datasets CR, MPQA, SUBJ, TREC, and SST, Bi-BloSAN performs similarly to DiSAN.

MASS [110] (MAsked Sequence to Sequence pre-training) is a semi-supervised model (unsupervised pre-training and supervised fine-tuning to a specific task) based on Transformer. The encoder receives a sentence with a randomly masked fragment, while the decoder predicts

that masked fragment. MASS is a generalization of GPT and BERT. It has a hyperparameter that defines the length of the masked fragment, which when set to 1 makes MASS equivalent to BERT, and when set to the number of tokens in a sentence makes MASS equivalent to GPT. For the machine translation task evaluated on NewsTest'14 and NewsTest'16 from the WMT dataset, MASS scores a 34 average BLEU score, which is an improvement of 1.85 points over XLM.

A modification of BERT, SBERT [111] (Sentence-BERT) is a supervised model trained on pairs of sentences. SBERT uses Siamese and triplet network structures. The Siamese structure consists of two BERT networks with tied weights, which are then fed into one layer that calculates the similarity between the inputs of the two BERTs. The Triplet structure receives upon input an anchor sentence, a positive sentence, and a negative sentence. The triplet network is then trained in a way to make the distance between the anchor sentence and the positive sentence smaller while making the distance between the anchor sentence and the negative sentence bigger. BERT is slow on large-scale tasks like semantic similarity comparison, information retrieval, and clustering. It takes BERT 65 hours to find the most similar sentence pair in a collection of 10,000 sentences, whereas SBERT takes 5 seconds to compute 10,000 sentence representations and 0.01 seconds to compute cosine similarity.

Generally, autoencoding models (e.g., BERT) perform better than autoregressive models (e.g., Transformer-XL), but BERT is not optimal because it neglects the dependency between the masked positions and suffers from a pre-train/finetune discrepancy [55]. Autoencoding models perform better because of their ability to model bidirectional contexts. XLNet [55] is an unsupervised model that combines BERT and Transformer-XL. XLNet integrates ideas from Transformer-XL while learning bidirectional contexts by maximizing the expected likelihood over all permutations of the factorization order. As a result of its autoregressive formulation, XLNet overcomes the limitations of BERT. XLNet is able to model anaphora. It outperforms BERT on 20 evaluated datasets [55].

Models similar to BERT (that use masked language modeling training methods) corrupt the input by replacing some tokens with a mask and then train a model to reconstruct those tokens. ELECTRA [115] (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) has a more efficient training method. It replaces selected tokens with alternatives produced by a generator network and then trains a discriminative model to predict for each token if it was replaced by another plausible token or not. The architecture has two components: the generator, which is typically a small masked language model, and the discriminator—the ELECTRA model. The generator and the discriminator are trained jointly: the generator of the output text with tokens replaced with plausible alternatives, and the discriminator to detect which tokens are replaced. The ELECTRA model is similar to a generative adversarial network (GAN), but the generator is not trained to generate text that would deceive the discriminator, instead it produces tokens that have the highest likelihood. On the GLUE datasets, ELECTRA

performs similarly as XLNet with an average accuracy of 89.5%, outperforming BERT by 5.5%.

REALM [114] (Retrieval-Augmented Language Model) is a semi-supervised masked language model. Language models typically capture world knowledge implicitly, REALM does it in a more modular and interpretable way by incorporating knowledge retriever in the learning procedure. Two main components of REALM are: the neural knowledge retriever, which finds a document containing the answer to the question, and the knowledge-augmented encoder, which outputs the answer to the question with the help of the retrieved document. The knowledge retriever uses a BERT-style architecture, and the knowledge-augmented encoder uses a vanilla Transformer architecture. After fine-tuning on the Open-domain Question Answering task, REALM outperforms BERT on the NQ and WQ datasets by 13.9% and 23%, achieving 40.4% and 40.7% accuracy, respectively.

Following the initial versions of attention-based models, transformer architectures have continued to develop. Current leading LLMs often achieve enhanced capabilities primarily through substantial scaling of parameters and training data. This scaling is frequently accompanied by architectural innovations, such as Mixture-of-Experts (MoE) for managing larger models, components for multimodal processing, refined attention mechanisms, or explorations into recurrent structures. Some of the most noteworthy models are Mistral [117], Google's Gemini [118], Anthropic's Claude 3 [119], Meta's Llama 3 [120], OpenAI's GPT-4o [121], Alibaba's Qwen3 [122], DeepSeek-AI's DeepSeek-V3 [123], and DeepSeek-AI's DeepSeek-R1 [124].

2.1.7 Analysis Conclusions

In this review, we have systematized the findings according to the defined criteria: the representation level (the level of linguistic units for which representations are being learned); the input level (the granularity of data upon input); the model type (the general strategy for representation learning); the model architecture (neural network architecture); and the model supervision (an indication of the need for labeled training data). Still, there are additional insights into the criteria for comparing, selecting, and evaluating the representation models, as well as insights into which model properties are important to consider when building or choosing an architecture for a specific downstream task and language.

Comparison. The majority of shallow models are unsupervised, learn word-level representations, and take word-level input. They perform well on simple tasks like measuring word similarity and are the easiest architecture to train.

Word-level input was popular among recurrent models, but recently subword-level input has been more frequently used. In this study, we noticed that recurrent models do not prefer one type of supervision or representation level more frequently than other types.

The majority of recursive models work with word or subword-level input and learn sentence+ representations for larger chunks of text like phrases, sentences, paragraphs, or entire

documents. Recently, recursive models that generate latent trees are preferable due to the low data preparation requirement. They are a good fit for semantic and sentiment tasks [80] where the tree structure of a text is important. Recursive models are currently underexplored, and some of the popular neural network libraries (e.g., TensorFlow) do not support them well (with some exceptions like DyNet).

Convolutional and attention models mainly use subword or word-level input. Attention models most frequently learn sentence+ representations. Unsupervised models are preferred again because of large amounts of data usable without any manual labeling. For tasks that process long texts with long-range dependencies, attention models outperform all the other models. Lately, attention models are increasing in size to improve the performance but at the cost of increased training time and higher memory requirements.

Shallow models are effective when dealing with word-level units, but if the downstream task depends on higher-level units (e.g., phrases, sentences, etc.), recursive, convolutional, or attention models are a better choice. Currently, a significant drawback with recursive models is a complicated training procedure that is hard to implement and leads to slower training times. Convolutional models can learn local patterns and are a good fit for computer vision tasks. It has been shown that convolutions can be used for text, but currently, attention models outperform them on text-related tasks as attention mechanisms can learn long-range dependencies.

Subword input provides the most flexibility, as unseen and rare words are easily dealt with, and input vectors are not as large. The subword-level facilitates the learning of neural models for morphologically rich languages. Complex downstream tasks benefit from sentence+ representations, but word representations and shallow models still work well for simple downstream tasks.

Computational Complexity. Model's computational complexity is an important factor when deciding which model or architecture to use, or which one is superior. As such, when reporting results for a model, information about computational complexity should be a priority. Some of the models in this survey use the number of trainable parameters as a proxy for computational complexity.

Word2Vec's skip-gram model has the complexity $C \times (D + D \times log_2(V))$ (where C is window size, V is size of the vocabulary, and D is dimensionality of the representations) [1]. If vocabulary contains 30k words, representations' dimensionality is 300, and window size is 10, Word2Vec's computational complexity would be 47618. If accounting for the number of training steps, the complexity is equal to 47,618 $\times$ $E \times T$, where E is the number of the training epochs and T is the number of the words in the training set. On the other end of computational complexity, GPT-2 has 1.5B trainable parameters [106], which if used as a proxy for complexity is approximately 3,000,000% higher complexity than Word2Vec.

Challenges. Large industry-based research groups are developing extremely large models with billions of parameters that require powerful hardware and custom systems for training. Even the inference of test cases becomes problematic with the largest models [125]. For smoother progress of large text representation models, new solutions that can support large models' training on clusters are expected.

Further, to be able to compare representation models, a standardized evaluation strategy will have to emerge including definitions for standard datasets, tasks, and measures. There are several obstacles for research teams with limited computational resources when studying the comparative performance of their models against large models. Namely, contrasting the models of different magnitudes of parameters and training time will lead to inconsistent and possibly misleading results. To obtain comparable results, the large models would have to be evaluated in several versions, ranging from the smallest to the biggest (measured by the number of trainable parameters). Accordingly, low-resourced research teams would be able to compare models fairly by evaluating the model of the same magnitude as an adequate version of the large model.

Evaluating a model on the same downstream task used for training does not assess its generality. One solution for addressing the generality evaluation is to use cross-validation across tasks. A model will be trained on one of the tasks and evaluated on the rest, repeating that process for every task in the benchmark. Hence, there is a need to define a better evaluation measure that will quantify the quality of the neural text representation model's generality across tasks.

2.2 Desirable Model Properties

The main properties we aim for in our text representation model include representation compositionality, hierarchical representation, representation decodability, and self-supervised learning.

2.2.1 Representation Compositionality

Composing word embeddings to generate meaningful representations of larger text units remains a critical area of study in NLP (explored in [126], [127], and [128]). Approaches to this problem span from simple aggregation techniques that compose word embeddings to more complex neural architectures that embed entire sentences, each offering unique advantages in different contexts.

A common approach is to average word embeddings to generate a single vector representing a phrase or sentence. Joulin et al. [129] introduced this idea in the context of fastText, where word embeddings are averaged and subsequently used for efficient text classification. This technique, inspired by the CBOW model [1], offers a computationally lightweight solution

that performs competitively with deeper models in various NLP tasks.

Building upon this, Arora et al. [130] proposed an enhanced version where word embeddings are combined using weighted averages, followed by post-processing through principal component analysis (PCA) or singular value decomposition (SVD). The weighting scheme they propose significantly improves performance on textual similarity tasks. This method demonstrates that simple compositional techniques can rival more complex architectures, especially in unsupervised settings.

Wieting et al. [131] conducted a comparative study that highlighted the trade-offs between simple word averaging and more complex models like LSTMs for sentence embedding. Their findings showed that while LSTMs perform well on in-domain data, simple word averaging techniques tend to outperform LSTMs in out-of-domain tasks. This suggests that straightforward compositional methods, despite their simplicity, are robust and generalizable across diverse datasets.

Recursive models have also been explored for word compositionality, often by mirroring linguistic hierarchies. Some recursive models explicitly rely on syntactic parse trees. Socher et al. [25] proposed a Recursive Neural Tensor Network (RNTN) that captures compositional semantic effects by recursively computing vectors for higher nodes in a parse tree, using a tensor-based composition function. Simoulin and Crabbé [85] introduced a model that learns its composition function, a Tree-LSTM, and its dependency structure, via a biaffine parser, jointly. The parser's edge weights then guide the weighted sum of child representations within the Tree-LSTM during composition.

Other recursive approaches aim for compositionality without strict reliance on predefined syntax or learn latent syntactic structures. Zhao et al. [79] introduced a Self-Adaptive Hierarchical Sentence Model, using recursive structures learned through supervision to capture compositional semantics in a non-syntactic hierarchy. The Continuous Recursive Neural Network (CRvNN) [84] incorporates a continuous relaxation to RvNNs, enabling end-to-end differentiability and automatic, task-specific latent tree induction, with a gated recursive cell performing the composition.

Transformer [52] architectures use attention mechanism to compose embeddings. Bahdanau et al. [98] introduced the attention mechanism in neural machine translation, allowing models to dynamically focus on different parts of the input sequence during decoding. The introduction of attention helped relieve the encoder from compressing all information into a single fixed-length vector, thus enabling a more flexible and effective composition of representations over sequential data.

Building on Transformer principles, some recursive models integrate attention or Transformer-style encoders for composition. Transformer Grammars [86] augment Transformers with syntactic inductive biases, using special attention masks and transformations of linearized trees to achieve recursive syntactic compositions and create explicit composed representations for each

constituent. Hu et al. [87] presented Fast-R2D2, a pretrained recursive neural network based on a pruned CKY mechanism, where an unsupervised top-down parser guides the pruning of the CKY chart, and an n-layer Transformer encoder serves as the composition function for combining cell representations. Similarly, Hu et al. [88] proposed ReCAT, which augments Transformers with Contextual Inside-Outside (CIO) layers that perform iterative bottom-up composition and top-down contextualization, with a single-layered Transformer also used as the core composition function within these layers.

In this work, we build on these approaches by applying the Pyramidal Recursive learning (PyRv) method to recursively combine word embeddings into more abstract representations. Unlike the averaging techniques of Joulin et al. [129] and Arora et al. [130], PyRv enables hierarchical composition, capturing both word-level and higher-level semantic structures in a more structured way.

PyRv differs from recursive models reliant on explicit syntactic parse trees, such as RNTNs and the model by Simoulin and Crabbé, as PyRv operates without requiring such predefined linguistic structures. Its hierarchy is formed by a fixed, symmetric, and exhaustive pairwise composition strategy. This also distinguishes it from supervised recursive models like that of Zhao et al., as PyRv is fully unsupervised. Compared to other unsupervised or latent-syntax recursive models like CRvNN, PyRv does not infer a single optimal tree structure. Instead, its compositional steps are determined by its inherent pyramidal architecture and guided by local autoencoding and autoregressive objectives.

Additionally, by recursively merging embeddings in a localized manner, PyRv offers an alternative to the global attention mechanisms in Transformers [52] and Transformer-based recursive models like Transformer Grammars, Fast-R2D2, and ReCAT. While these models leverage the power of Transformer encoders for complex compositional functions or global context integration, PyRv employs a simpler encoding function recursively.

2.2.2 Hierarchical Representation

The second property, hierarchical representation, enables text to be represented at various levels of granularity, from individual characters or subwords to complete phrases or sentences. This property is supported by PyRv through pyramidal recursion, wherein neighboring pairs of embeddings are recursively composed into higher-level embeddings. Hierarchical representation is particularly beneficial for languages with complex morphology, variable-length phrases, and compound words. Furthermore, it facilitates fine-grained analysis of text by capturing meaningful structures across different abstraction levels.

Deep learning models often demonstrate a general progression of feature complexity, with lower layers learning basic features and higher layers capturing more abstract patterns. For example, CNNs identify simple edges in lower layers and increasingly complex patterns in higher layers [132]. Recursive neural networks, however, reuse the same layers and weights recursively

across abstraction levels, making them inherently well-suited for hierarchical representations.

Recursive models explicitly employ recursive operations to build hierarchical embeddings. In contrast, many non-recursive language models lack such explicit structural composition.

Recursive Neural Tensor Network (RNTN) [25] captures compositional semantic effects in text by recursively computing vectors for higher nodes in a parse tree.

Deep Inside-Outside Recursive Autoencoder (DIORA) [83] employs the inside-outside algorithm within a latent tree chart parser to induce syntax trees and learn hierarchical representations. The inside pass compresses the sentence recursively, while the outside pass incorporates external context.

Transformer Grammars [86] explicitly model hierarchical syntactic structures by creating composed representations for each constituent, with the hierarchy being defined by an input linearized tree.

The model by Simoulin and Crabbé [85] uses a dependency tree, parsed by a biaffine parser, to dictate the hierarchical structure, with a Tree-LSTM composing embeddings recursively from leaves to the root.

Other recursive models induce latent tree structures to form hierarchies. The Continuous Recursive Neural Network (CRvNN) [84] induces such a structure, and the depth of recursion, defining the hierarchy, depends on this induced tree.

Fast-R2D2 [87] uses a CKY-style chart where cells represent constituents at different spans, forming levels of a hierarchy. A top-down parser then determines the specific tree structure for recursive composition.

ReCAT [88] employs stacked Contextual Inside-Outside layers to explicitly emulate a hierarchical composition process, with an inside pass composing lower-level spans into higher-level ones and an outside pass enabling information flow across these hierarchical levels.

Dynamic Convolutional Neural Network (DCNN) [92] adapts CNN architectures for sentence modeling using one-dimensional convolution operations. Dynamic k-max pooling, a key feature of DCNNs, enables the network to handle sentences of varying lengths while inducing hierarchical representations of words and phrases.

Graph Convolutional Neural Network (Graph-CNN) [133] transforms text into a graph-of-words representation to capture non-consecutive and long-distance semantic relationships. Convolutional layers operate on this graph to learn hierarchical features, representing varying levels of semantics.

PyRv achieves hierarchical representation through its systematic pyramidal recursion. This contrasts with parse-tree-based recursive models like RNTNs, Transformer Grammars, and the model by Simoulin and Crabbé, because PyRv's hierarchy is not tied to linguistic syntax (constituency or dependency trees). Its hierarchy is a fixed, layered structure based on systematic pairwise merging from tokens upwards, providing a different kind of hierarchical abstraction.

Compared to latent-tree models such as CRvNN, Fast-R2D2, ReCAT, and DIORA which infer or operate on an induced tree structure to guide hierarchical composition, PyRv imposes a consistent pyramidal structure. It does not search for an optimal parse tree, its pyramidal levels themselves constitute the hierarchical representations, making its hierarchical nature explicit and deterministic given the input sequence length.

Unlike models such as DCNNs or Graph-CNNs that induce hierarchy through operations like pooling or graph convolutions, PyRv's hierarchy is formed by the direct, recursive application of the same encoding function to build distinct representations at each level of the pyramid.

A key aspect of PyRv is that it explicitly generates representations at each level of its pyramid – capturing subword compositions, word compositions, and multi-word phrase compositions. This makes these intermediate hierarchical states directly accessible, which can be contrasted with models where hierarchy might be more implicitly encoded in the network's depth or learned attention patterns.

2.2.3 Representation Decodability

Representation decodability refers to the ability of a model to map a learned representation back to the original input space. This property is fundamental in evaluating and utilizing learned representations effectively. PyRv achieves this capability through its autoencoder head, which is explicitly trained to perform input reconstruction.

Autoencoders are a prime example of models that leverage decodability [134, 135]. Autoencoders are neural networks designed to encode input data into a compressed latent representation and then decode it to reconstruct the original input. The encoding process forces the hidden representation to capture meaningful features about the data, rather than simply memorizing the input-output mapping. The ability to decode provides several advantages:

- Evaluating representation quality: The fidelity of the reconstructed input compared to the original can be used to assess the quality of the learned representation. High-quality reconstructions indicate that the latent representation captures critical information about the data.
- Data generation: The decoder component of an autoencoder can be used as a generative model. By sampling points from a prior distribution in the latent space, novel data samples can be generated.
- Interpretability: Decoding representations can reveal the features learned by the model, aiding in the interpretation of what aspects of the data are being captured in the latent space.

The hierarchical LSTM model in [68] extends autoencoder decodability to paragraphs and documents by creating hierarchical embeddings from words to sentences to paragraphs, enabling the reconstruction of coherent long texts. This approach is useful for tasks like summa-

rization and text generation.

The variational autoencoder (VAE) in [136] utilizes LSTM-based encoders and decoders to capture high-level sentence features such as style and syntax. The decoding process allows for the generation of sentences conditioned on the latent representations, enabling both data generation and interpolation in the latent space.

Recursive autoencoders (RAEs) from [76] build hierarchical representations for phrases and sentences using tree structures, with each node holding a "parent" vector for reconstruction. This method supports interpretability and excels in sentiment prediction without requiring sentiment lexicons.

BERT [4] indirectly demonstrates decodability via its masked language model (MLM) objective, predicting masked tokens from context. This approach maps learned representations back to vocabulary space, achieving state-of-the-art results in NLP tasks.

Similarly, Fast-R2D2 [87] employs a pretraining objective to reconstruct tokens based on context representations derived from its tree encoder. ReCAT [88] also uses MLM.

An alternative to decoder-based reconstruction is nearest-neighbor search in embedding spaces. Methods like fastText [2] use vector representations to find the most similar data points to a given query by searching for the closest vectors in the latent space. This approach does not rely on explicit decoding but leverages the geometry of the latent space to retrieve similar items efficiently. Nearest-neighbor search has been widely applied in tasks like word similarity, image retrieval, and recommendation systems.

Both decodability approaches, explicit reconstruction and nearest-neighbor retrieval, highlight the utility of well-structured latent representations. These methods demonstrate how decodability supports a range of applications, from feature learning and interpretability to practical deployment in search and generation tasks.

PyRv's autoencoding head provides explicit reconstruction capabilities at each level of its pyramidal hierarchy. This offers a direct and granular form of decodability, allowing, for example, a phrase-level representation to be decoded back into its constituent word representations, or a word representation back into its subword components.

This explicit reconstruction of constituent parts distinguishes PyRv from models that primarily rely on MLM for decodability, such as BERT, Fast-R2D2, and ReCAT. While MLM effectively decodes by predicting missing tokens from context, PyRv's autoencoder aims to reconstruct the exact input representations of the combined elements, potentially leading to different learned properties in the embeddings focused on faithfulness to the original components.

2.2.4 Self-supervised Learning

Self-supervised learning has emerged as a powerful paradigm in machine learning, allowing models to leverage large amounts of unlabeled data for representation learning. In contrast to

traditional supervised learning, which requires labeled datasets, self-supervised methods generate pseudo-labels from the data itself. This approach enables models to learn meaningful representations that can be transferred to downstream tasks.

One of the strengths of PyRv lies in its support for self-supervised learning, as it does not rely on labeled data. PyRv features two distinct heads: an autoencoding head and an autoregressive head. The autoencoding head focuses on reconstructing input data, capturing its underlying structure, while the autoregressive head models sequential dependencies. Both mechanisms facilitate the learning of robust representations in an unsupervised manner.

An important aspect of many recursive models is their reliance on parse trees, which represent hierarchical syntactic structures. These models either require explicit parse trees as input or learn to infer them during training.

PyRv adopts a pyramidal hierarchy, eliminating the need for parse trees. This design choice simplifies the training process and allows the model to directly exploit the hierarchical nature of textual data. By utilizing self-supervised learning, PyRv is capable of training on large-scale unlabeled datasets.

2.3 Identified Gaps and Challenges

Many models fail to capture all four properties essential for robust text representation: hierarchical representation, compositionality, decodability, and self-supervised learning. Each property contributes uniquely to the quality and utility of learned embeddings, so it is beneficial for a text representation model to support all of them.

Many models achieving robust compositionality and explicit hierarchy, such as Recursive Neural Tensor Networks (RNTNs) [25], Transformer Grammars [86], and the unified parsing model by Simoulin and Crabbé [85], typically rely on syntactic parse trees. This dependence on external linguistic annotations or pre-parsed data limits their applicability to richly annotated languages and domains and hinders their ability to learn purely from raw text.

Conversely, large-scale self-supervised models, particularly those based on the Transformer architecture like BERT [4], excel at learning powerful contextual representations from vast quantities of unlabeled text. However, the hierarchical structures learned by these models are often implicit within their deep layers, and their summary representations (e.g., the [CLS] token) may not offer clear decomposability into constituent parts. While their decodability is evident through objectives like Masked Language Modeling (MLM), this typically involves predicting missing tokens rather than reconstructing the structured build-up of the representation itself. Integrating explicit hierarchical processing and granular, multi-level decodability into these highly effective self-supervised frameworks without sacrificing their training efficiency or scalability remains a considerable hurdle.

Effectively integrating all four properties presents significant challenges. Some recent self-

supervised recursive models like DIORA [83], CRvNN [84], Fast-R2D2 [87], and ReCAT [88] make major strides by inducing latent tree structures from unlabeled data. However, the induced hierarchies might be optimized for specific downstream signals or internal objectives, and the decodability might still primarily be at the token level (e.g., via MLM or token reconstruction objectives within the induced structure), rather than a systematic decoding of the compositional steps that formed the hierarchy. Thus, achieving robust compositionality and an explicit, multi-level decodable hierarchy through purely self-supervised mechanisms remains an active area of research.

Chapter 3

Theoretical Background

The success of modern machine learning, particularly in domains like natural language processing (NLP), computer vision, and speech recognition, owes much to advancements in representation learning and neural architectures. These fields enable models to capture intricate patterns and relationships in data, often with minimal reliance on manual feature engineering. This chapter lays the foundational concepts that underpin the proposed Pyramidal Recursive learning (PyRv) method. It explores the theoretical pillars of neural network representations, autoencoding, autoregressive learning, and recursive neural networks, providing a coherent context for PyRv's design.

We begin by discussing neural network representations (3.1), emphasizing the evolution from handcrafted features to learned embeddings and hierarchical representations, particularly in text processing. Next, we delve into the principles of autoencoding and autoregressive learning (3.2), detailing how these paradigms contribute to robust, self-supervised representation learning. Following this, recursive neural networks (3.3) are introduced as powerful architectures for handling hierarchical data, setting the stage for PyRv's recursive embedding approach. Finally, we explain how these ideas converge in the context of Pyramidal Recursive learning (3.4).

3.1 Neural Network Representations

Neural networks fundamentally aim to learn representations of data that facilitate the extraction of meaningful information. This capability is at the heart of representation learning, where the learned features are optimized for downstream tasks, reducing the reliance on manual feature engineering.

3.1.1 The Role of Representations in Machine Learning

The performance of machine learning models is highly sensitive to the choice of data representation [137, Chapter 15]. Historically, the creation of effective representations relied on feature engineering, where domain experts manually designed transformations and preprocessing steps. However, this process is both time-consuming and prone to limitations when dealing with high-dimensional or complex data. Representation learning automates this process, enabling models to learn to disentangle the underlying explanatory factors of the data autonomously. This paradigm shift has significantly accelerated progress in areas such as NLP, computer vision, and speech recognition.

A representation is considered good if it provides a compact and informative encoding of the input data that is effective for downstream tasks, typically in a supervised learning context [138]. Given a dataset $\mathscr{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}^d$ are input samples and y_i are corresponding outputs, a learned representation $z = f_\theta(x)$, where $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ is a transformation parameterized by θ, should satisfy:

$$\mathscr{L}(y, g(z)) \ll \mathscr{L}(y, h(x)), \tag{3.1}$$

where $\mathscr{L}$ is a loss function, $g : \mathbb{R}^k \to \mathbb{R}^m$ is a simpler supervised model (e.g., linear regression), and h is a more complex function relying on raw features. The representation z thus encodes the essential information from x relevant for predicting y.

Deep learning introduces the concept of deep representations, where data undergoes multiple layers of transformations. Each layer captures progressively more abstract features, enabling neural networks to model complex relationships and patterns. In a deep network, a sequence of nonlinear transformations is applied as:

$$z^{(l+1)} = \sigma(W^{(l)} z^{(l)} + b^{(l)}), \quad l = 1, \ldots, L, \tag{3.2}$$

where $z^{(l)}$ represents the activations at layer l, $W^{(l)}$ and $b^{(l)}$ are learnable weights and biases, and σ is a nonlinearity such as the ReLU or sigmoid function. The final layer $z^{(L)}$ is used as the output or the representation for downstream tasks.

3.1.2 Representations in Text

In the domain of text, neural networks have played a pivotal role in advancing representation learning. Early efforts in distributed representations were introduced by Hinton [139], where symbolic data was encoded into dense, continuous vectors. These efforts were later formalized in statistical language modeling by Bengio et al. [140], introducing embeddings that capture semantic and syntactic properties.

For a vocabulary of size V, each word with index $i \in \{1, 2, \ldots, V\}$ can be represented as

a dense vector $\mathbf{e}_{w_i} \in \mathbb{R}^d$, where w_i is the word corresponding to index i. A common objective for learning these embeddings is to maximize the likelihood of context words given a target word. For example, in the skip-gram model, the optimization objective minimizes the negative log-likelihood:

$$\mathscr{L} = -\sum_{t=1}^{T} \sum_{c \in \text{context}(t)} \log P(w_c \mid w_t), \tag{3.3}$$

where w_t is the target word, w_c are context words, and $P(w_c \mid w_t)$ is modeled using a softmax over the embeddings.

The effectiveness of these embeddings is illustrated in Figure 3.1, which demonstrates how Skip-gram embeddings can organize semantic concepts and capture relationships, such as those between countries and their capitals. Such embeddings serve as input to more complex neural architectures, which further refine and leverage these representations for tasks like text classification, machine translation, and sentiment analysis.

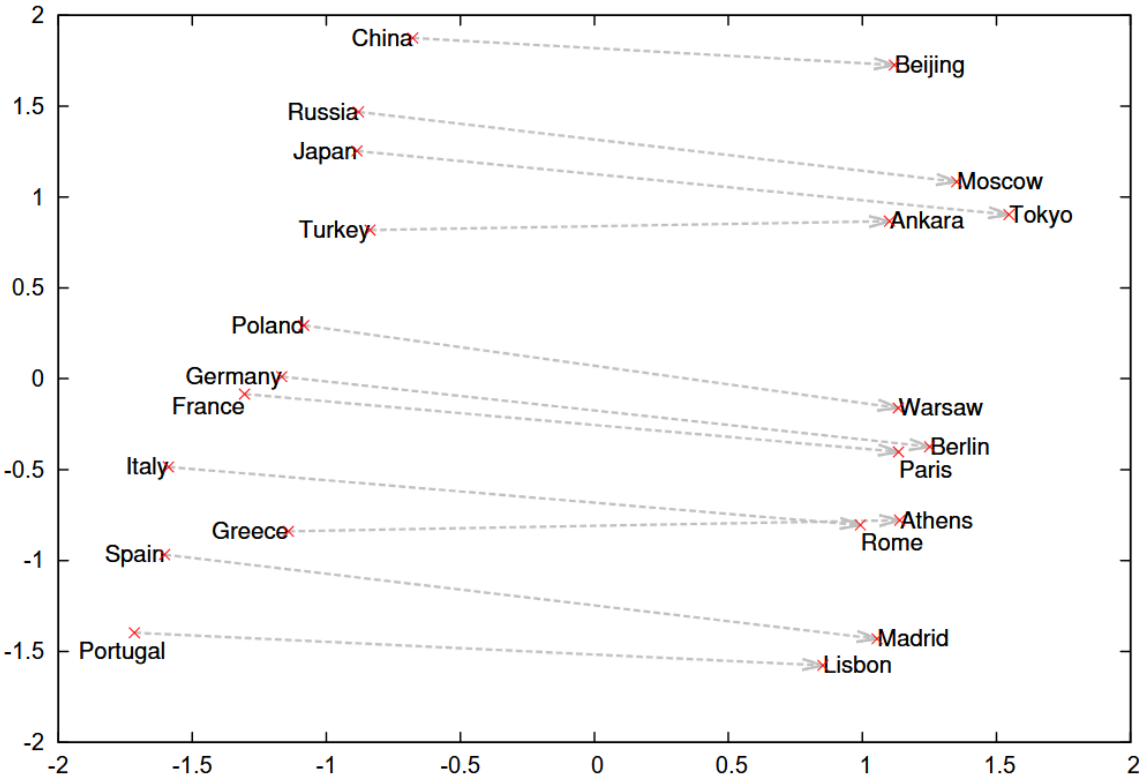


Figure 3.1: Two-dimensional principal component analysis (PCA) projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities [48]. This visualization highlights the ability of Skip-gram embeddings to organize semantic concepts and capture implicit relationships, such as the association between countries and their capitals, without explicit supervision.

Deep representations. A hierarchical organization of explanatory factors underpins the effectiveness of deep representations. In this hierarchy, abstract concepts are defined in terms of simpler, less abstract ones, enabling a structured understanding of complex relationships.

Deep representations exploit this assumption to model data with increasing levels of abstraction, which is particularly relevant in NLP.

In NLP, this hierarchical structure is reflected in the composition of linguistic elements. Subwords or words can be combined to form phrases, and these phrases can further combine to represent sentences. Deep neural networks leverage this compositional nature, learning representations at multiple levels. For example, an embedding layer might learn dense representations for individual words, which are then passed through subsequent layers to capture the semantics of phrases and sentences.

3.2 Autoencoders and Autoregressive Learning

Neural network models often leverage principles of autoencoding and autoregressive learning to develop representations that capture essential characteristics of the data. In this section, we explore these two paradigms, emphasizing their formalizations and significance in learning compact, expressive, and generative representations.

3.2.1 Autoencoders

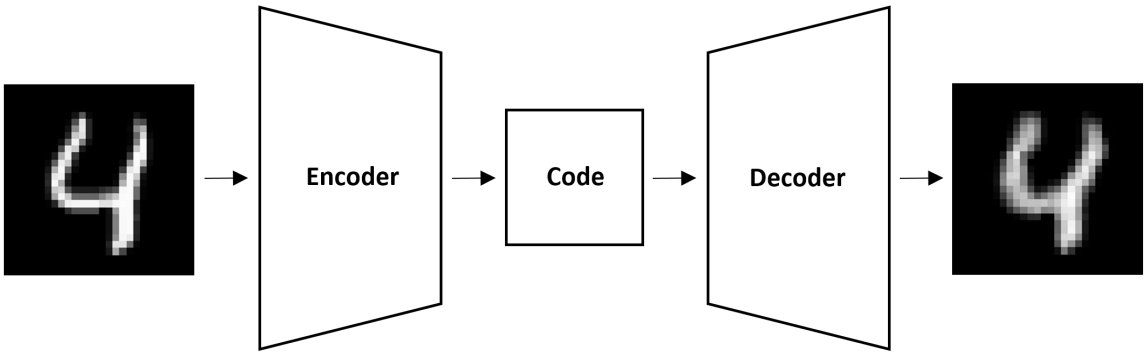


Figure 3.2: Visualization of an autoencoder architecture, consisting of an encoder that maps the input (e.g., a picture of a handwritten digit) to a lower-dimensional hidden representation (code), and a decoder that reconstructs the input from the hidden representation.

An autoencoder is a neural network architecture designed to reconstruct its input as output, learning an efficient representation (code) in the process. An autoencoder consists of two primary components (visualized in Figure 3.2):

- An **encoder function** $h = f_\theta(x)$, which maps the input $x \in \mathbb{R}^d$ into a hidden representation $h \in \mathbb{R}^k$, where $k < d$.
- A **decoder function** $r = g_\phi(h)$, which reconstructs the input from the hidden representation.

The training objective of an autoencoder minimizes a reconstruction loss between the input

x and its reconstruction r, expressed as:

$$\mathscr{L}(x,r) = \mathscr{L}(x, g_\phi(f_\theta(x))), \tag{3.4}$$

where $\mathscr{L}$ is often chosen as the mean squared error:

$$\mathscr{L}(x,r) = \|x-r\|_2^2. \tag{3.5}$$

Autoencoders are deliberately restricted to prevent perfect copying of input data, encouraging the network to capture the most salient features of the data [137, Chapter 14]. This restriction can take various forms, leading to different types of autoencoders:

- Undercomplete Autoencoders: Employ a bottleneck structure where the hidden layer h has fewer dimensions (k) than the input (d), enforcing a compressed representation. The compression prioritizes learning essential features, particularly those relevant to the training data distribution.
- Regularized Autoencoders: Incorporate a regularization term into the loss function to induce specific properties in the learned representations.
- Sparse Autoencoders: Impose sparsity on the hidden layer h by penalizing activations, encouraging a representation where few neurons are active.
- Denoising Autoencoders: Train the model to reconstruct clean input x from corrupted input $\tilde{x}$, enforcing robustness to noise.
- Contractive Autoencoders: Regularize the derivative of the encoder function $f_\theta(x)$ to encourage invariance to small input perturbations.
- Variational Autoencoders (VAEs): Learn probabilistic representations by maximizing a variational lower bound on the data log-likelihood.
- Recursive Autoencoders (RAEs): Extend autoencoders for hierarchical data, such as sentences, using recursive structures to aggregate representations.

3.2.2 Autoregressive Learning

Autoregressive models are generative models that predict the probability of a sequence by modeling the conditional probability of each element in the sequence given the preceding elements [137, Chapter 20]. Figure 3.3 shows how such models generate a sentence step by step by considering a fixed window of preceding elements. For a sequence of observed variables $x = (x_1, x_2, \ldots, x_d)$, the joint probability is decomposed as:

$$P(x) = \prod_{i=1}^{d} P(x_i \mid x_{i-1}, \ldots, x_1). \tag{3.6}$$

In this framework, each term $P(x_i \mid x_{i-1}, \ldots, x_1)$ represents the probability of x_i conditioned

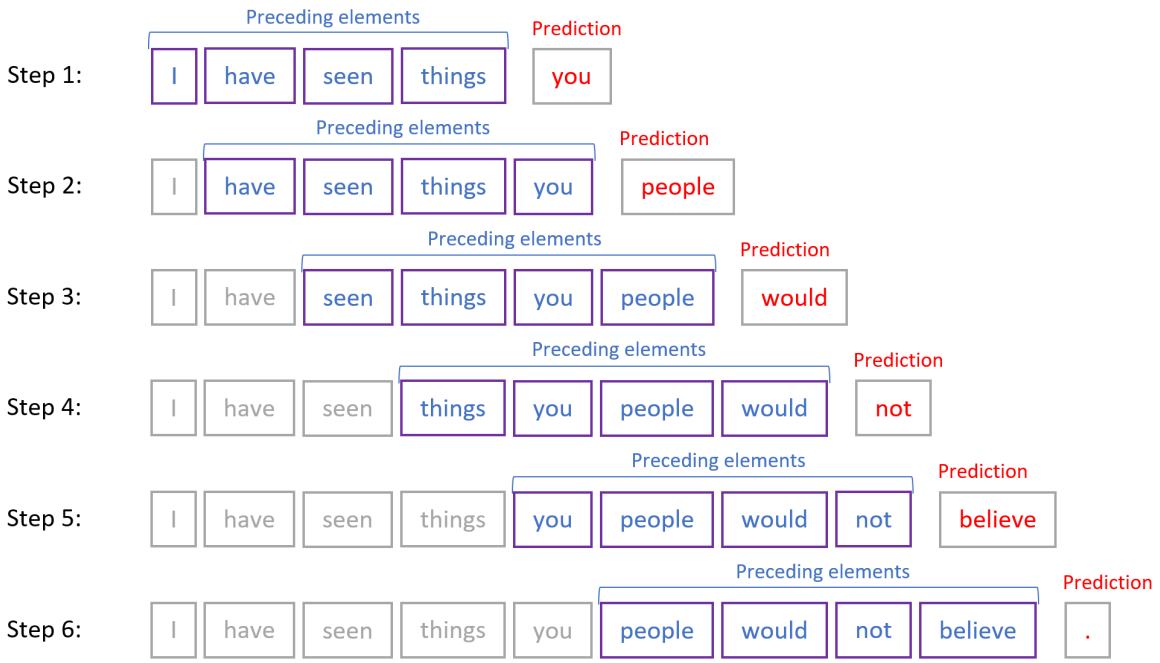


Figure 3.3: Illustration of the autoregressive modeling process for sentence generation. The model predicts the next word in a sequence step by step, based on a fixed window of four preceding words.

on all preceding elements, and is typically modeled using a neural network. This approach is referred to as neural autoregressive modeling.

Neural autoregressive models leverage neural networks to parameterize the conditional distributions. The graph structure of these models is often a complete directed graph, with each node corresponding to a variable in the sequence. For instance:

$$P(x_i \mid x_{i-1}, \ldots, x_1) = \text{Softmax}(f_\theta(x_{<i})), \tag{3.7}$$

where $f_\theta(x_{<i})$ represents a neural network function parameterized by θ, and $x_{<i}$ denotes the set of preceding variables.

By leveraging the chain rule of probability and neural networks, autoregressive models can capture intricate dependencies within sequences, making them powerful tools for generative modeling.

3.2.3 Integration of Autoencoding and Autoregression

The integration of autoencoding and autoregressive learning provides a robust framework for recursive embedding, combining the strengths of both paradigms to model text representations effectively. This subsection explores the theoretical unification of these approaches and their complementary roles in capturing semantic and structural information.

Theoretical Unification. Autoencoding focuses on learning a compact, decodable representation of the input data, while autoregressive learning emphasizes modeling the sequential de-

pendencies between elements. In a recursive framework, these principles can be unified by encoding complex structures into latent embeddings that are both informative and predictive. The integration involves two objectives:

- Autoencoding objective: Ensure the embedding can decode back into the original structure.
- Autoregressive objective: Predict the conditional distributions of structural components.

By optimizing both objectives, the model learns embeddings that are decodable (via autoencoding) and semantically meaningful (via autoregression).

Autoregressive Learning for Recursive Embeddings. Recursive embeddings represent hierarchical or sequential structures by encoding pairs or groups of units recursively. For example, in the context of text, the encoded pair may combine subwords, words, or phrases into a single latent representation, which is then recursively combined with other components. Autoregressive learning complements this by ensuring that these embeddings capture dependencies between the elements, enabling the model to account for semantic relationships.

Our method leverages this integration by predicting the immediate left and right nodes of the encoded pair. Through recursive composition, however, both the neighboring nodes and the encoded pairs usually represent multiple text units. This recursive composition allows the model to maintain larger context while predicting neighboring nodes.

Advantages of Integration. The unification of autoencoding and autoregression offers several unique advantages:

- Decodability: Autoencoding ensures that the learned embeddings are not just abstract representations but can be reconstructed back into text or other input forms, enabling interpretability and reconstruction-based tasks.
- Semantic Learning: Autoregressive learning aids in capturing fine-grained sequential dependencies, enriching the embeddings with semantic information essential for downstream tasks like language modeling or sentiment analysis.
- Hierarchical Modeling: Recursive embedding, when combined with these approaches, allows for the compact representation of complex structures, balancing global context and local relationships.

3.3 Recursive Neural Networks

Recursive Neural Networks (RvNNs) are a class of neural networks designed to operate on hierarchical or tree-structured data. Unlike traditional feedforward or recurrent neural networks, RvNNs recursively apply the same set of weights at each node of a tree structure to compute parent node representations from their child nodes. Formally, given a tree structure with a parent

node p and child nodes $\{c_1, c_2, \ldots, c_k\}$, the representation of the parent node h_p is computed as:

$$h_p = f_\theta(h_{c_1}, h_{c_2}, \ldots, h_{c_k}), \tag{3.8}$$

where f_θ is a learnable function parameterized by θ, often implemented as a neural network layer. Recursive networks enable effective modeling of data with hierarchical relationships, such as syntactic trees in NLP. Figure 3.4 illustrates this process applied to a morphological tree, where word representations are constructed recursively from morphemic vectors.

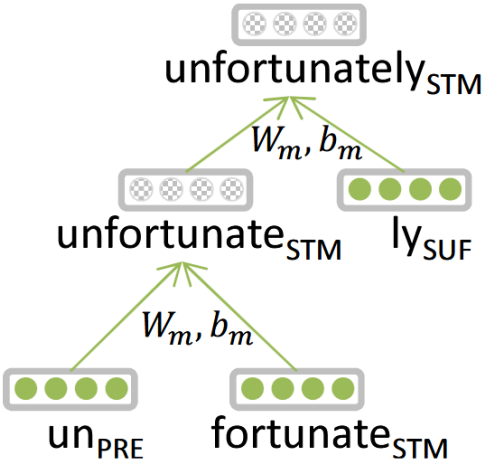


Figure 3.4: Morphological Recursive Neural Network [78]. A vector representation for the word "unfortunately" is constructed from morphemic vectors: $\text{un}^{\text{pre}}, \text{fortunate}^{\text{stm}}, \text{ly}^{\text{suf}}$.

3.3.1 Backpropagation Through Structure

Backpropagation through structure (BPTS) [141] is an extension of the traditional backpropagation algorithm, specifically designed for training recursive neural networks on tree-structured data. While traditional backpropagation computes gradients layer by layer in feedforward networks, and backpropagation through time (BPTT) extends this process to sequential recurrent structures, BPTS adapts the gradient computation to hierarchical tree structures.

The adaptation in BPTS arises from the recursive dependency in tree-structured data, where the parent node's representation is computed as a function of its children. During backpropagation, this hierarchical relationship requires the error signal to be distributed appropriately among child nodes.

Forward Propagation. For a parent node p, its representation h_p is computed from the representations of its child nodes $\{h_{c_1}, h_{c_2}\}$ as:

$$h_p = g\left(W \begin{bmatrix} h_{c_1} \\ h_{c_2} \end{bmatrix} + b\right), \tag{3.9}$$

where W is a weight matrix applied to the concatenated child representations (h_{c_1} and h_{c_2}), b is a bias vector, and g is the activation function (e.g., tanh, ReLU). This formulation establishes a dependency between h_p and the outputs of its child nodes, which must be accounted for in the backward pass.

Backward Propagation. During backpropagation, the error signal $\delta_p = \frac{\partial \mathscr{L}}{\partial h_p}$, which represents the gradient of the loss $\mathscr{L}$ with respect to h_p, is propagated to the child nodes. This is achieved using the chain rule of differentiation.

To simplify the gradient computation, the tree structure is virtually unfolded into a sequence of layers, analogous to how BPTT unrolls recurrent networks in time. Each node in the tree corresponds to a "layer" in this virtual structure. The error signals are propagated layer by layer, ensuring that the exact gradient of the loss is computed.

Challenges. BPTS enables recursive neural networks to learn effectively from hierarchical data. However, it introduces challenges such as:

- Computational complexity: Traversing tree structures and computing gradients for all nodes can be computationally expensive.
- Gradient flow in deep trees: As trees grow deeper, gradients may vanish or explode, especially in regions where node representations depend on long hierarchical paths.

3.4 Pyramidal Recursive Learning

Our proposed method, Pyramidal Recursive learning PyRv, unifies autoencoding, autoregression, and recursive neural networks into a hierarchical framework for text representation. It addresses the challenges of learning structured, hierarchical representations of text through a novel recursive encoding mechanism, as described in the next chapter.

PyRv operates by constructing a pyramidal hierarchy of representations, starting from low-level units (e.g., characters, subwords, or words) and progressing to higher levels (e.g., words, phrases, sentences). This approach leverages:

- **Autoencoding:** To enforce decodability, each level of the pyramid is encoded and decoded with an autoencoding head, enabling the reconstruction of lower-level representations from higher-level embeddings.
- **Autoregression:** To ensure embeddings are semantically and syntactically meaningful, an autoregressive head predicts neighboring representations at each level.
- **Recursion:** The encoding of pairs of representations into higher-level embeddings mimics the recursive operations in tree structures, facilitating compositional representation learning.

3.4.1 Training Challenges and Solutions

Training PyRv introduces challenges primarily due to the hierarchical and recursive nature of the architecture, as well as constraints imposed by computational complexity. To address these issues, the following strategies are employed:

Small Batch Sizes. To manage computational demands, PyRv uses small batch sizes, typically comprising one paragraph of text per batch. While this minimizes memory usage, it can lead to unstable training dynamics due to limited gradient diversity within each batch.

Multi-Level Loss Aggregation. To stabilize training despite small batch sizes, losses are computed at every level of the pyramid. Each node on each level contributes an output and a corresponding loss, ensuring that gradient updates capture information from all levels of recursion. This design is conceptually similar to residual or skip connections, providing direct gradient pathways that mitigate issues with vanishing gradients and improve training stability.

Gradient Normalization. To prevent gradient vanishing or explosion in deep hierarchies, PyRv normalizes losses using a custom normalization technique, detailed in the next chapter (Equation 4.8). By consistently scaling losses across all levels, this method ensures numerically stable gradient updates, enabling the model to converge reliably.

Chapter 4

Methodology

Large language models with Transformer architecture [52] currently dominate natural language processing (NLP) space, but other paths towards representation of text should be investigated as well, especially smaller models that are easier to train and less demanding to use. Recursive neural networks (RvNNs), such as those proposed by Luong et al. [78] and Socher et al. [25], generalize recurrent neural networks (RNNs). While RNNs have a linear structure, RvNNs have a hierarchical structure. RNNs are well suited for sequential data (e.g., time series data, text, etc.), but suffer from slow training due to their sequential nature (parallel processing is difficult), and difficult processing of long-term dependencies. Language is sequential but has an implicit hierarchical structure (letters, syllables, morphemes, words, phrases, sentences, etc. [142]), so processing text in a hierarchical structure could provide benefits over processing it linearly.

Besides language being hierarchical in its nature, possibly making RvNNs a good option for text representation, other reasons to use an RvNN for text are the variable length of the input token sequence and reduction of the required neural network depth due to the recursive application of the same weights.

Furthermore, the proposed approach based on Pyramidal Recursive learning enables the incorporation of certain desirable properties. The primary properties we aim for in our text representation model include representation compositionality, hierarchical representation, representation decodability, and self-supervised learning. Part of this chapter, which describes the PyRv method and its properties, has been published in our previous work ([143]).

The first property, representation compositionality (explored in [126], [127], and [128]), entails that multiple representations can be combined into a single, unified representation. A model with such a property could capture the meaning of a complex text by combining the meanings of its constituent parts in a structured and semantically meaningful way. When a model can combine representations, it can learn more efficiently from limited data.

The second property, hierarchical representation, would allow text to be represented at var-

ious levels of detail, from individual characters or subwords to complete phrases or sentences, producing multilevel representations. This property could be useful for languages with complex morphology, variable-length phrases, and compound words. Hierarchical representations enable fine-grained analysis of text. Most recursive models share this property (e.g., RAE [76] and DIORA [83]), while most non-recursive language models do not.

The third property, representation decodability, simply means that a representation can be decoded back into the text that it represents. Decodability can serve as a useful metric for evaluating the quality of text representations and making it easier to understand and interpret what the model has learned. All autoencoders share this property (e.g., [68]).

The fourth property, self-supervised learning, means the model does not need a labeled dataset for training as the model trains by predicting neighboring representations and decoding encoded representations, and it does not need a parse tree for each sentence due to the model's pyramidal nature. Most other recursive models for text require parse trees for each sentence (e.g., RNTN [25] and Tree-LSTM [80]).

Utilizing the proposed method, we introduce a novel model for text representation, the Pyramidal Recursive Neural Network (PyRvNN). This model achieves the desired properties by adopting a recursive autoencoding and autoregressive approach, embedding text in a recursive pyramidal fashion. At the zeroth pyramid level, the model takes as input a pair of one-hot encoded tokens. As the pyramid levels increase, the model processes pairs of dense representations corresponding to sub-words, words, or phrases. The output includes these representations and their neighboring counterparts to the left and right of the input pair.

The chapter is organized as follows:

- Pyramidal Recursive Learning (4.1): This section explains the PyRv method and its computational complexity.
- Pyramidal Recursive Neural Network Architecture (4.2): This section outlines the design of PyRvNN, including input representations.
- Training Procedure (4.3): This section discusses the self-supervised training process, covering training data, pyramidal training, loss computation, and the training setup.
- Evaluations (4.4): The section details intrinsic and extrinsic evaluations, downstream tasks, and model variations.
- Model Limitations and Constraints (4.5): This section provides a critical assessment of PyRvNN's limitations and practical considerations.

4.1 Pyramidal Recursive Learning Method

Pyramidal Recursive learning (PyRv) is a method specifically designed for training neural networks to represent text. It operates by constructing a pyramidal hierarchy of representations, as illustrated in Figure 4.1. In this hierarchy, each level encapsulates higher-level representations

of the input text, progressively abstracting it from characters or subwords to words, phrases, sentences, and potentially paragraphs.

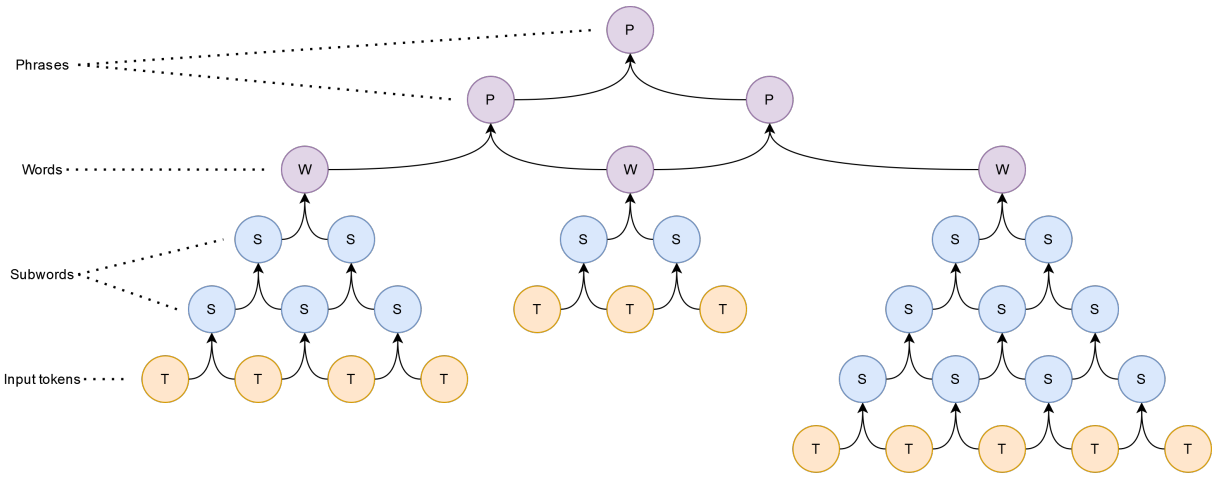


Figure 4.1: A visualized example of a pyramidal recursion in the PyRv method. The lowest-level nodes correspond to input tokens. Moving upward, the nodes within the three pyramids represent combined subword embeddings. At the pyramid peaks, nodes represent word embeddings, and higher nodes signify combined word embeddings, representing phrases.

At each level of the pyramid, the method encodes each successive pair of representations into a higher-level representation. This representation is then used for two tasks: decoding the representation into the pair of representations it was encoded with (autoencoding head) and predicting the representations to the left and right of the encoded pair of representations (autoregressive head).

The recursive encoding can start with characters or subwords (or with words in a simpler version of this method). Each successive pair of representations is encoded into one representation, building a pyramid of subword representations for each word. After reaching the word level the model starts pairing and encoding those representations into a single pyramid of phrase representations starting with words until it reaches the tip of the pyramid or a defined maximum level.

Due to this recursive encoding process, the trained model possesses the capability to embed text hierarchically, generating representations on each level within the pyramidal hierarchy, thus exhibiting a hierarchical representation property. Furthermore, neighboring representation pairs can be jointly encoded into a single embedding, showcasing the property of representation compositionality.

The autoencoding head can be used to decode the entire pyramid from the top-level representation to the bottom-level representations. The autoregressive head is used to predict the neighboring characters, subwords, words, etc., and learn a representation space in which the relationship between the text representations is semantically/lexically meaningful.

The representation decodability property is facilitated by the autoencoder, as the trained model possesses the ability to decode each representation down to the lowest level of the pyra-

mid (tokens). The autoencoding and autoregressive nature of the model make the training process self-supervised, eliminating the necessity for labeled data.

4.1.1 Complexity

The computational complexity of the PyRv method can be expressed as the number of node pairs the neural network embeds into higher nodes. This count equals the number of nodes in a pyramid, excluding the bottom-level nodes, which represent the input tokens. The total number of nodes in a pyramid with $n+1$ input nodes can be calculated as the sum of nodes across each level, except the bottom one (base level).

To find this sum of nodes, we employ the arithmetic series formula:

$$S_n = \frac{n(a_1 + a_n)}{2} \tag{4.1}$$

where S_n represents the sum of the series, n is the number of terms, a_1 is the first term, and a_n denotes the n-th term.

In our context, n corresponds to the number of levels above the pyramid's base. This value also equals the number of nodes in the level directly above the base. The first term $a_1 = 1$ represents the apex node of the pyramid, while the n-th term $a_n = n$ reflects the n nodes in the level immediately above the base. Substituting these into the arithmetic series formula simplifies the sum to:

$$S_n = \frac{n(1+n)}{2} = \frac{n(n+1)}{2}. \tag{4.2}$$

This closed-form expression, $S_n = \frac{n^2+n}{2}$, efficiently calculates the cumulative count of nodes across all levels above the base.

Considering only the dominant term, the computational complexity of the PyRv method can be expressed as $O(n^2)$ for a single pyramid with $n+1$ input nodes. However, this analysis does not account for the specifics of the neural network architecture employed.

By leveraging parallel processing of nodes at the same pyramid level on a graphics processing unit (GPU), the computational complexity of the PyRv method can be reduced to $O(n)$, where n denotes the number of pyramid levels (excluding the bottom one) and the bottom pyramid level contains $n+1$ nodes.

4.2 Pyramidal Recursive Neural Network Architecture

Following the proposed method, we implement a neural network model, PyRvNN. The model's layers are fully connected, but it has two paths of fully connected layers leading from the two inputs to the one central layer that then branches towards two output layers for each head, autoregressive and autoencoding, which is four output layers in total. The architecture is shown

in Figure 4.2.

In the main architecture version, we use fully connected layers with the Leaky ReLU activation function, defined as:

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x \geq 0, \\ \alpha x & \text{if } x < 0, \end{cases} \tag{4.3}$$

where α is a small positive constant (in our case, $\alpha = 0.01$). For the one-hot output layers, we employ the softmax activation function, given by:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{4.4}$$

where z_i represents the input to the i-th output neuron and K is the total number of output classes. Additionally, L2 regularization is applied to each layer, which adds a penalty term proportional to the squared magnitude of the weights:

$$L_{\text{reg}} = \lambda \sum_{i=1}^{n} w_i^2, \tag{4.5}$$

where λ is the regularization strength (in our case, $\lambda = 0.001$) and w_i are the weights of the layer.

The same neural network (with same weights and biases) is used recursively on each pyramid level for every pair of nodes.

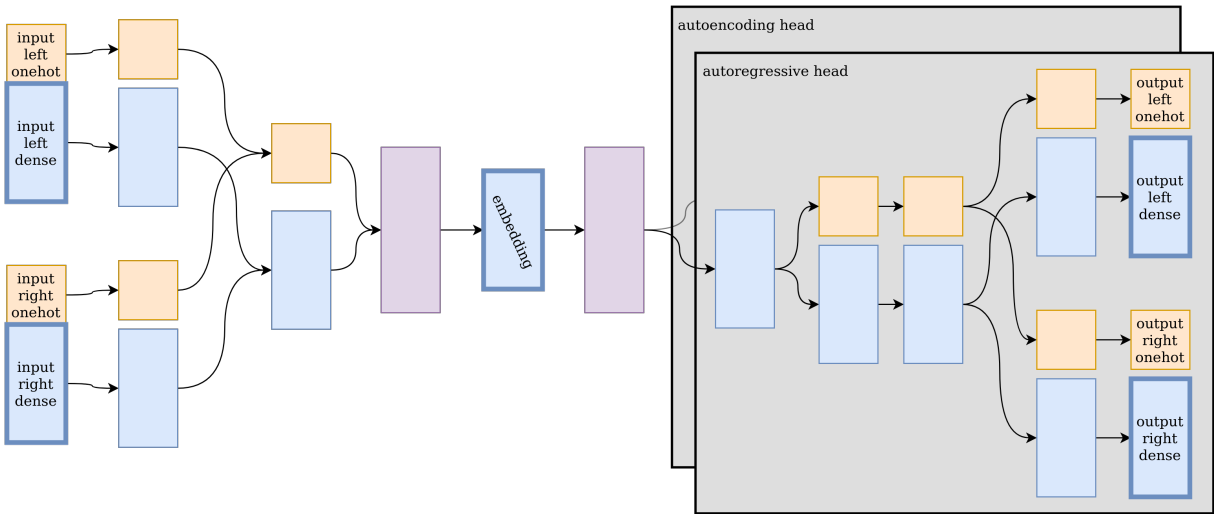


Figure 4.2: PyRvNN model architecture. The inputs are representations of adjacent nodes (one-hot or dense embeddings). Boxes connected with an arrow are fully connected layers. Autoregressive and autoencoding heads are identical, only the output labels they recieve differ.

4.2.1 Input Representations

At each pyramid level, the input comprises two types of vector pairs: one-hot vectors and dense vectors. In the initial pyramid level (zeroth), one-hot vector pairs represent adjacent input tokens, which are subwords in the current implementation. As we ascend to higher pyramid levels, these one-hot vectors instead signify the current recursion level (either a subword pyramid or the phrase pyramid).

At the zeroth pyramid level, the dense vectors are initialized with zeros. In subsequent pyramid levels, these dense vectors encapsulate representations obtained from activations in the middle "embedding" layer. These representations encode information about two neighboring nodes from the pyramid level immediately below.

The autoencoding head's outputs reconstruct the input vectors, while the autoregressive head's outputs predict vectors on both the left and right from the encoded pair of vectors from the input.

4.2.2 Embedding Dimension

The embedding dimension is a critical hyperparameter in the PyRvNN model. It directly dictates the dimensionality of the main embedding generated by the PyRvNN encoder. This dense representation is subsequently used by the decoding heads or as input for higher pyramid levels. Furthermore, the sizes of other layers within the PyRvNN model that process dense representations, such as those receiving embeddings from child nodes, are also determined by this embedding dimension. Consequently, the embedding dimension significantly impacts both the model's representational capacity and its computational demands. To investigate these effects, we conducted a series of ablation studies.

In these studies, we varied the embedding dimension, examining values of 1000, 1200, 1400, 1600, 1800, 2000, 2200, and 2400, an increment of 200 for each step. We evaluated the impact of these changes on decoding accuracy, processing time, and GPU memory consumption.

The effect of embedding dimension on model performance is presented in Figure 4.3. This figure illustrates the decoding accuracies achieved for different embedding dimensions when decoding is initiated from various pyramid levels. For this experiment, the PyRvNN model was trained on a corpus of 200,000 Wikipedia paragraphs for each of the tested embedding dimensions, with the reported accuracy representing the average performance on the last 200 paragraphs of this dataset. The figure indicates that while larger embedding dimensions generally correlate with higher decoding accuracy at the word pyramid level, the gains tend to diminish beyond a certain point. Phrase pyramid levels reach a plateau much earlier, which we attribute to the limited amount of training data used in this ablation study.

The choice of embedding dimension also has a substantial impact on computational re-

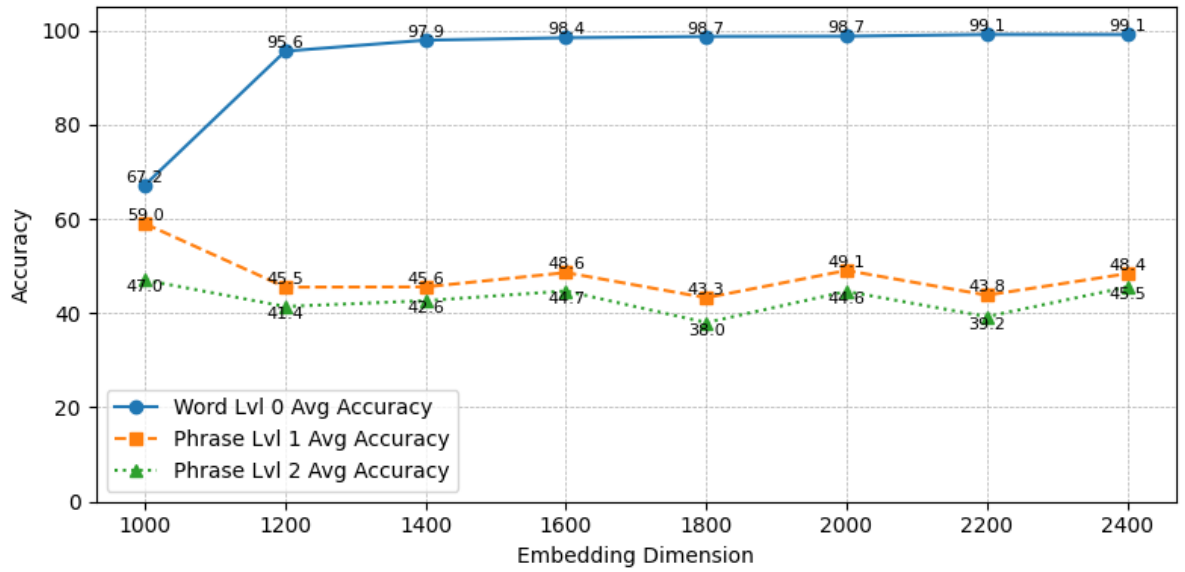


Figure 4.3: Decoding accuracy versus embedding dimension for different pyramid levels.

sources. Figure 4.4 presents a bar plot visualizing the average time taken for a single pyramid level to process its input, across different embedding sizes. The data suggests that the time complexity scales approximately linearly with the embedding dimension. This linear trend is further explored in Figure 4.5, which plots the average processing time per pyramid level against the input size (i.e., the number of pairs being embedded and processed concurrently) for each tested embedding dimension. Trend lines, computed using an averaging sliding window of size 50, confirm the linear relationship with input size. Notably, the figure illustrates that larger embedding dimensions lead to steeper trend lines, indicating a proportionally greater increase in processing time as more pairs are processed simultaneously.

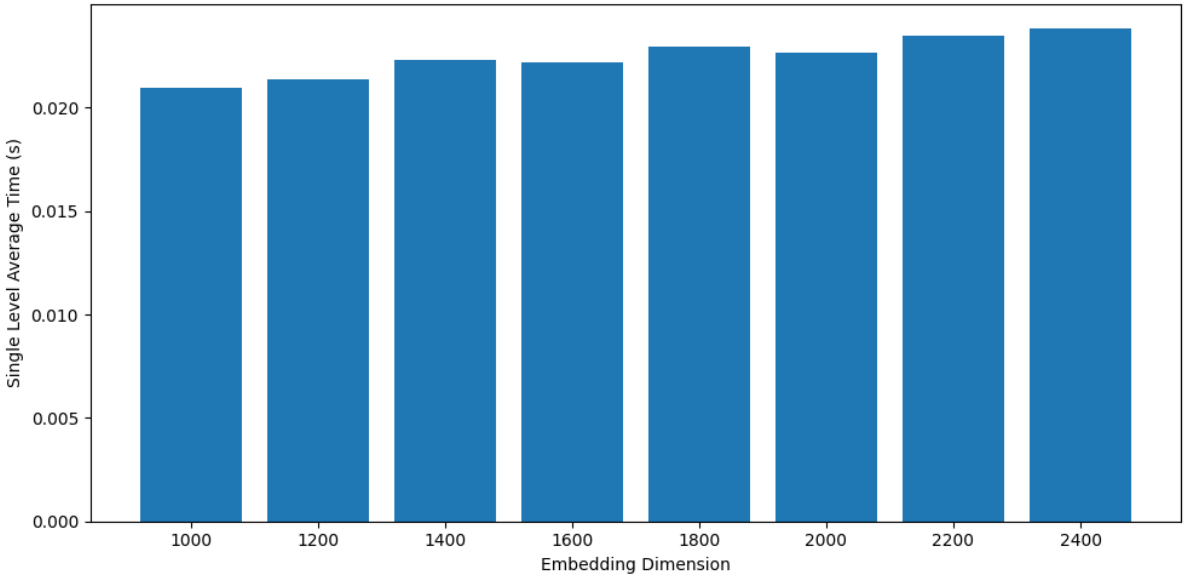


Figure 4.4: Average processing time per pyramid level versus embedding dimension.

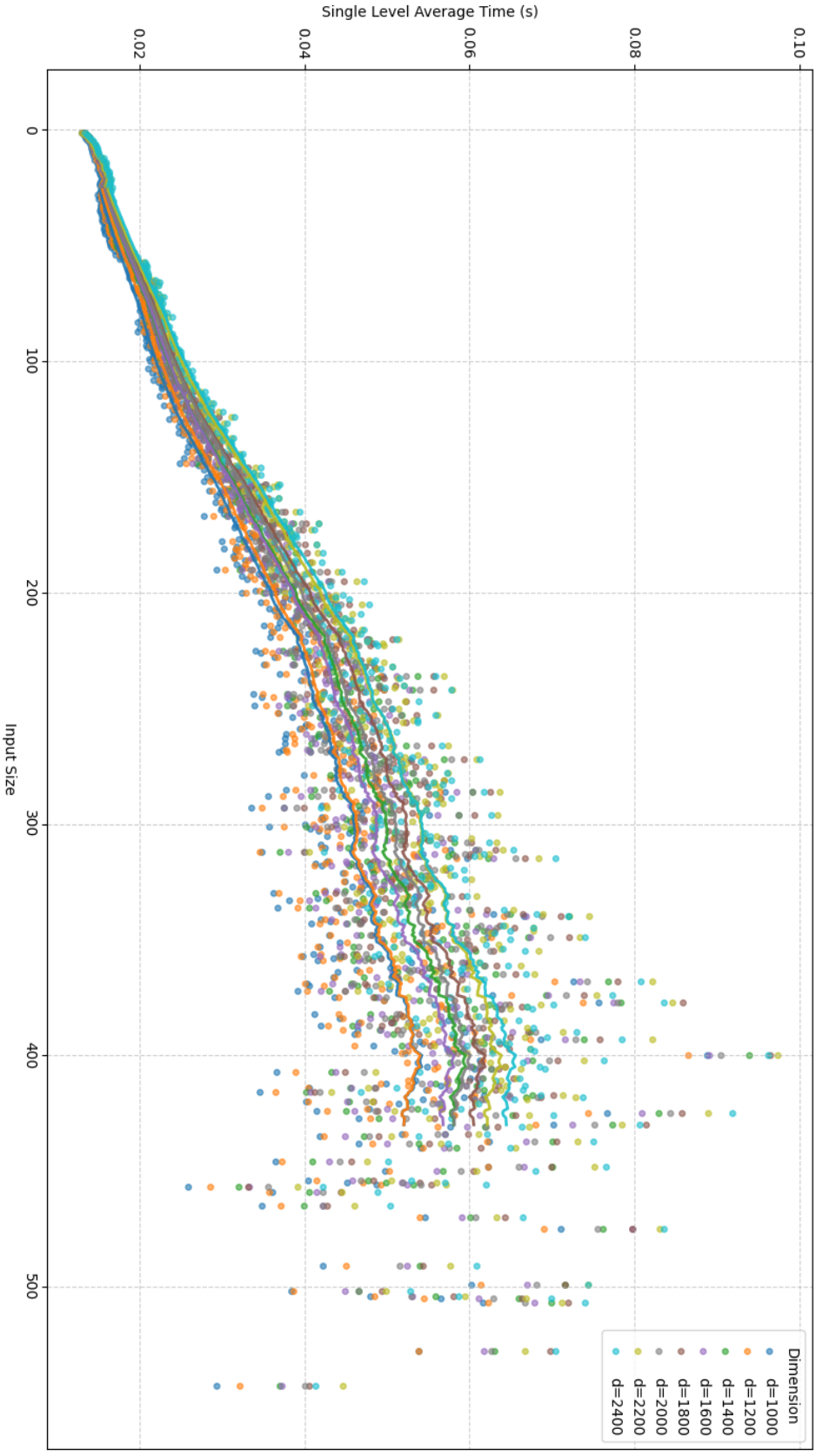


Figure 4.5: Average processing time per pyramid level versus input size for different embedding dimensions.

GPU memory consumption is another critical factor influenced by the embedding dimension. Figure 4.6 shows average GPU memory usage per pyramid level for each embedding dimension. The figure suggests that GPU memory consumption increases substantially, potentially exponentially, with the embedding dimension. Figure 4.7 details GPU memory usage as a function of input size across the different embedding dimensions. In contrast to the time measurements where larger dimensions resulted in steeper slopes (Figure 4.5), this plot shows that lines corresponding to larger embedding dimensions are primarily vertically offset, maintaining a roughly parallel increase with input size. This implies that while the incremental memory cost per additional input pair is relatively consistent across dimensions, the baseline memory footprint is significantly elevated for larger embedding dimensions from the outset.

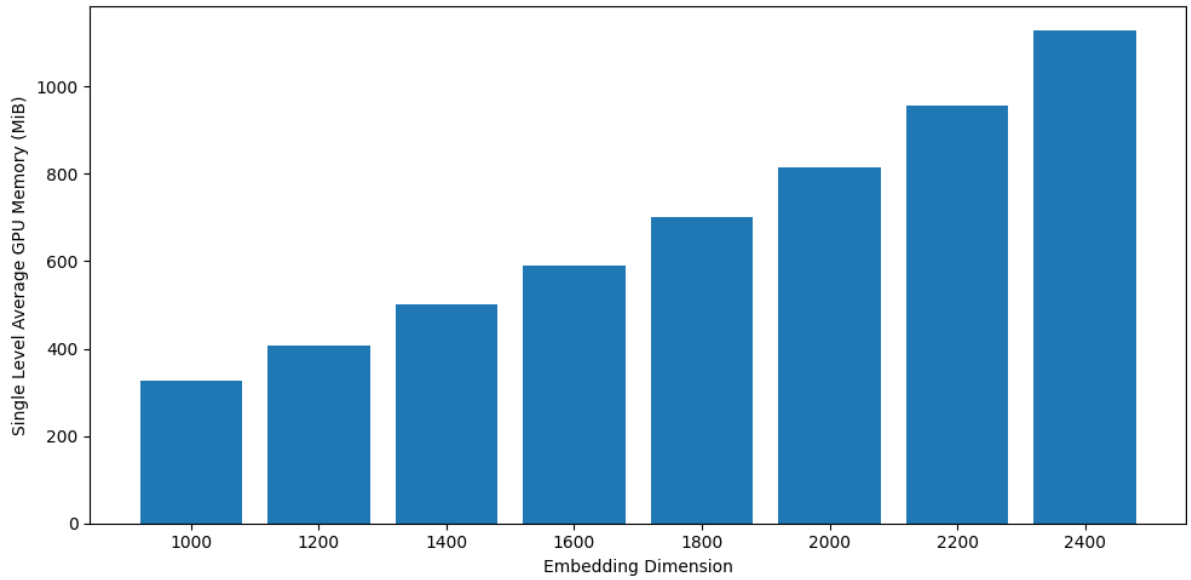


Figure 4.6: Average GPU memory usage per pyramid level versus embedding dimension.

4.3 Training Procedure

The training procedure is unsupervised, as it does not need labeled data. It is trained to decode the encoded inputs (with the autoencoding head) and to predict the neighboring nodes (with the autoregressive head).

The pyramidal recursive nature of PyRv method is visible in the way the model is trained. The method recursively embeds inputs following the pyramidal hierarchy (as is visualized in Figure 4.1 in Section 4.1), and on each level of recursion both autoencoding and autoregressive heads produce outputs.

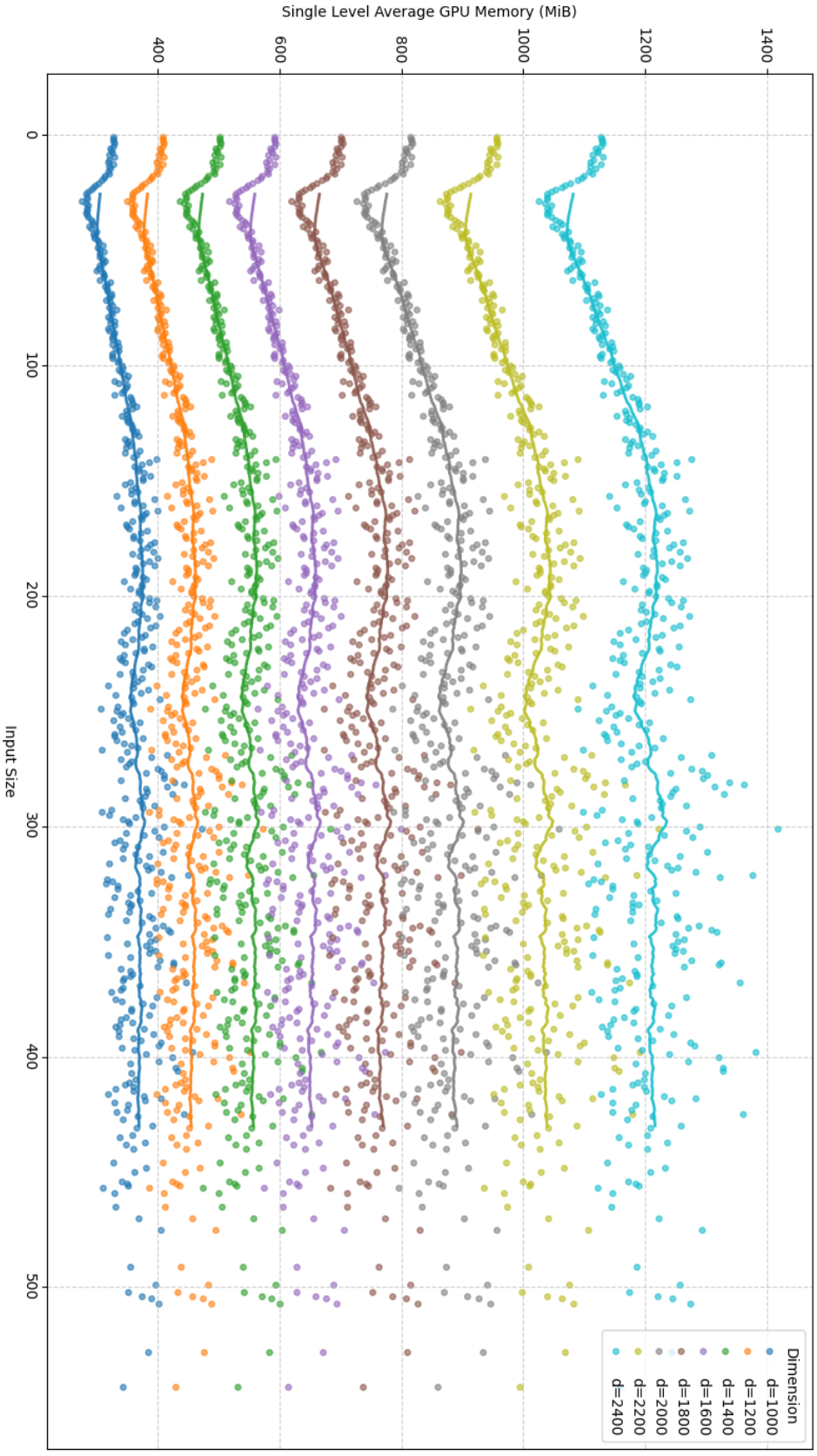


Figure 4.7: Average GPU memory usage per pyramid level versus input size for different embedding dimensions.

4.3.1 Training Data

We use English Wikipedia articles (text only) from Wikimedia public dump (11. January 2020) as the main training data. After preprocessing, it contains over 29,000,000 paragraphs (more statistics about the dataset are shown in Table 4.1 and Figure 4.8 that shows density plot of word counts per paragraph).

The entire text is lowercased and cleaned. We clean it by:

- removing external, file, and image links,
- replacing HTML special characters with Unicode characters,
- replacing different types of quotation marks and dashes with a singular and consistent quotation mark and dash,
- adding spaces around words and other elements (in preparation for tokenization by words),
- removing miscellaneous junk,
- removing paragraphs that are short (less than 128 characters) or too long (more than 2000 characters).

Examples of preprocessed texts from Wikipedia:

```
recursion ( adjective : recursive ) occurs when a thing is defined
    in terms of itself or of its type . recursion is used in a
   variety of disciplines ranging from linguistics to logic . the
   most common application of recursion is in mathematics and
   computer science , where a function being defined is applied
   within its own definition . while this apparently defines an
   infinite number of instances ( function values ) , it is often
   done in such a way that no infinite loop or infinite chain of
   references can occur .
for classification , output will be a vector of class
   probabilities ( e.g. , ( 0.1 , 0.7 , 0.2 ) , and target output
   is a specific class , encoded by the one-hot / dummy variable (
    e.g. , ( 0 , 1 , 0 ) ) .
w^l = ( w^l_ { jk } ) : the weights between layer l - 1 and l ,
   where w^l_ { jk } is the weight between the k-th node in layer
   l - 1 and the j-th node in layer l
```

After the texts have been cleaned and spaces have been added between words (and between other text units), we train a byte pair encoding (BPE) tokenizer. We set its vocabulary size to 500, alphabet limit to 50, and define a list of special tokens:

- `<subwordlvl>`: used as a flag after the initial recursion, signifying that the current recursion is computing embeddings on the subword level (nodes below word level - combinations of subwords).
- `<phraselvl>`: used as a flag after the initial recursion, signifying that the current

Table 4.1: Wikipedia dataset statistics

Statistic	Value
Paragraph count	29,402,923
Average words per paragraph	78
Standard deviation of words per paragraph	51
Minimum words in a paragraph	1
25th percentile words per paragraph	41
Median words per paragraph	64
75th percentile words per paragraph	100
Maximum words in a paragraph	653
Total word count	2,295,263,882

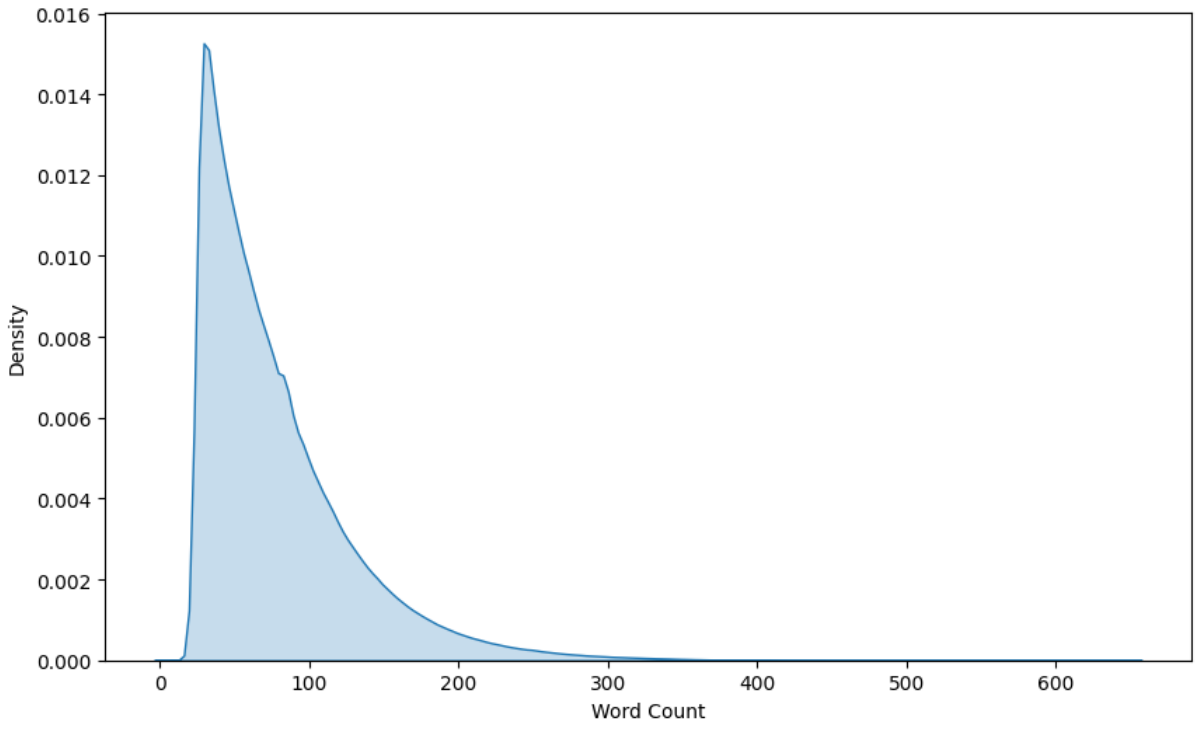


Figure 4.8: Density plot of word counts per paragraph.

recursion is computing embeddings on the word level and above (nodes above word level - nodes representing words and their combinations).

- `<word.beg>`: signifies the beginning of a word, and is placed before each word (a sequence of subwords of that word).
- `<word.end>`: signifies the ending of a word, and is placed after each word (a sequence of subwords of that word).
- `<->`: represents a dash surrounded by other characters (as in "`j-th`"), which differs from a dash that is surrounded by spaces (as in "`1 - 1`").
- `<unk>`: special token reserved for all characters that are not in the tokenizers alphabet (limited to 50 characters).

The ordering of paragraphs is shuffled and fed into the training method. Each paragraph is converted into one batch of input and output pairs used for training.

4.3.2 Pyramidal Training

The PyRvNN expects a pair of inputs (a neighboring pair of nodes) and two pairs of outputs (one for the autoencoding head, and the other for the autoregressive head). The nodes on input are one-hot representations of subwords when in the initial recursion (the zeroth subwords pyramid level), and dense embeddings (produced by the middle PyRvNN layer) when on any pyramid level above the zeroth.

The input and the autoencoder head's output are identical - a neighboring pair of nodes from the level below (nodes "`A B C D`" would form pairs "`AB BC CD`"). The autoregressive head's output pair contains the input node pair's left and right neighbors (the same nodes would form pairs "`_C AD B_`", where "_" symbolizes a null vector).

On each level of recursion, all the inputs and outputs are formed at once (by sliding windows) and fed into the PyRvNN. First, as each word has its own sub-pyramid (containing subword representations), each of those pyramids is processed before continuing with phrases. Once we reach the top node of every sub-pyramid (representing entire words), the same process continues for the pyramid above (which contains phrase representations) starting with word representations. Figure 4.9 shows an example of a phrase being recursively encoded.

Loss, as is computed on individual pyramid levels, is described in the next subsection. The total loss is the sum of those losses, divided by the number of pyramid levels.

4.3.3 Loss Computation

Loss is computed for each pyramid level as a combination of binary cross-entropy for one-hot vectors and Huber loss for dense vectors. Pseudo-code for loss calculation is provided in Algorithm 6 (in Appendix A).

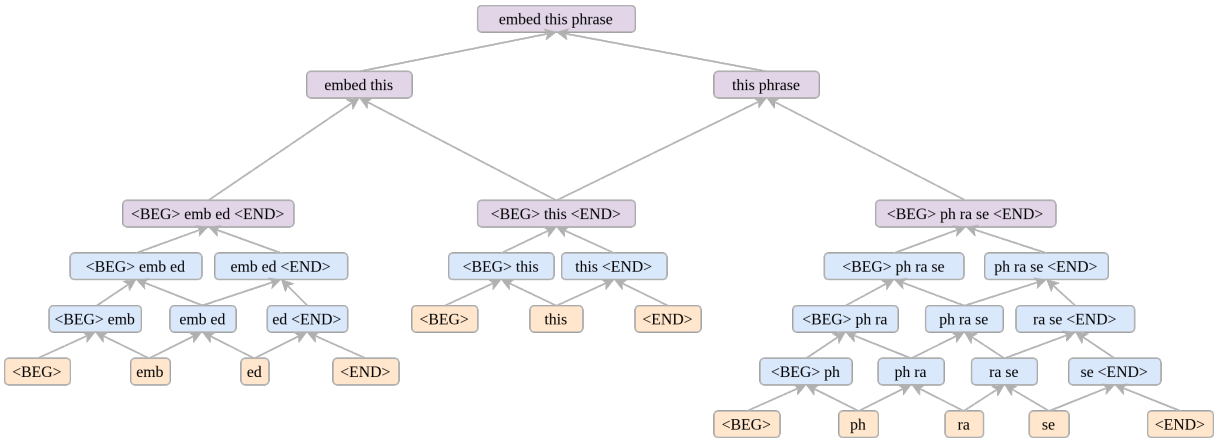


Figure 4.9: Utilizing the PyRv approach, the phrase "embed this phrase" undergoes recursive embedding. Tokenized words (subwords) are received at the bottom pyramid level, with each node in the hierarchy representing all the tokens below it, illustrating the recursive embedding process.

The binary cross-entropy loss function used for one-hot vectors is given by:

$$\text{BCE}(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{N}\sum_{i=1}^{N}(\mathbf{y}_i \cdot \log(\hat{\mathbf{y}}_i) + (1-\mathbf{y}_i)\cdot \log(1-\hat{\mathbf{y}}_i)) \tag{4.6}$$

Here, $\mathbf{y}$ represents the true binary label, and $\hat{\mathbf{y}}$ denotes the predicted probability that the sample belongs to a class.

The Huber loss function used for dense vectors is defined as:

$$\text{HuberLoss}(\mathbf{y}, \hat{\mathbf{y}}) = \begin{cases} \frac{1}{2}(\mathbf{y}-\hat{\mathbf{y}})^2 & \text{for } |\mathbf{y}-\hat{\mathbf{y}}| \leq \delta \\ \delta(|\mathbf{y}-\hat{\mathbf{y}}| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{4.7}$$

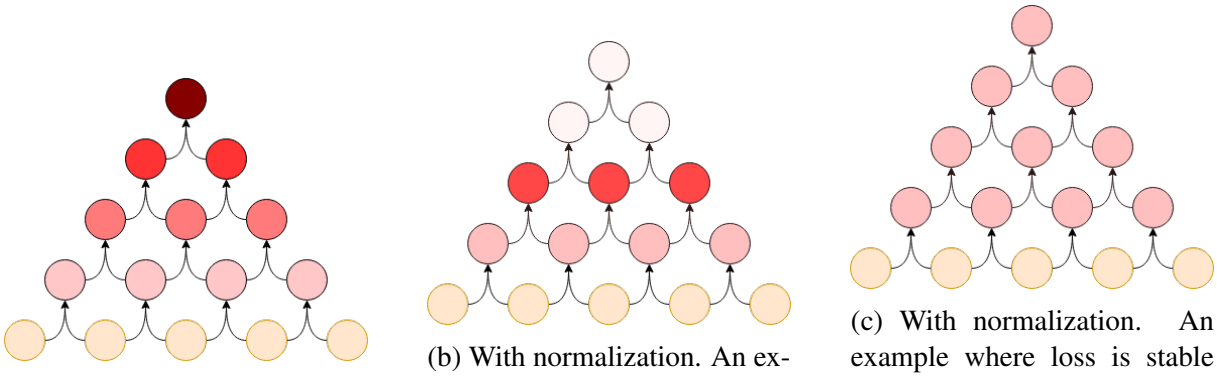
In this function, $\mathbf{y}$ represents the true value, $\hat{\mathbf{y}}$ denotes the predicted value, and δ serves as a threshold parameter.

Loss Normalization Based on Previous Level Loss Value. The model's ability to reconstruct the input is contingent upon the quality of input representations. If a pyramid level exhibits a high loss, the representations it generates will be of low quality. Subsequently, since these representations serve as input for the next pyramid level, that level will incur an even higher loss, as illustrated in Figure 4.10a.

To address this issue, we employ normalization of the loss on each level, resulting in stabilized losses, as visualized in Figures 4.10b and 4.10c. The normalization factor (NF) is inversely proportional to the total loss of levels below the current one (LB):

$$\text{NF} = \frac{1}{\text{LB}^a \cdot b + 1}, \tag{4.8}$$

where a and b are constants.



(a) Without normalization. Example of loss explosion.

(b) With normalization. An example where loss spiked on the middle level.

(c) With normalization. An example where loss is stable across all levels after some training.

Figure 4.10: Visualization of loss normalization by level.

Figure 4.11 shows loss by pyramid level for the initial untrained model, with and without normalization. Figure 4.12 shows the same but for the trained model.

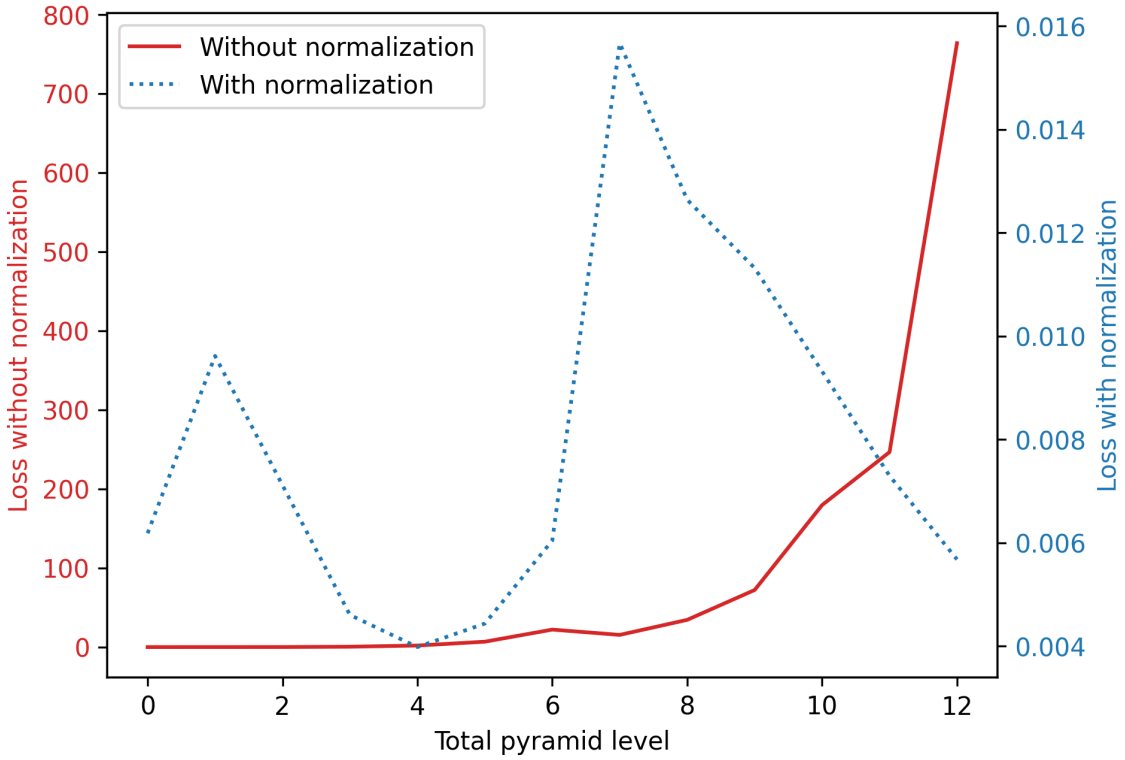


Figure 4.11: Loss by pyramid level for the initial untrained model, with and without normalization.

Loss Normalization Based on Representation Magnitude. The magnitude of PyRvNN representations varies depending on the pyramid level at which they are computed (illustrated in Figure 4.13). Since the loss function's value is influenced by these magnitudes, discrepancies in magnitude may lead to uneven learning speeds across pyramid levels. To address this, we normalize the loss function by the average magnitude of all output vectors.

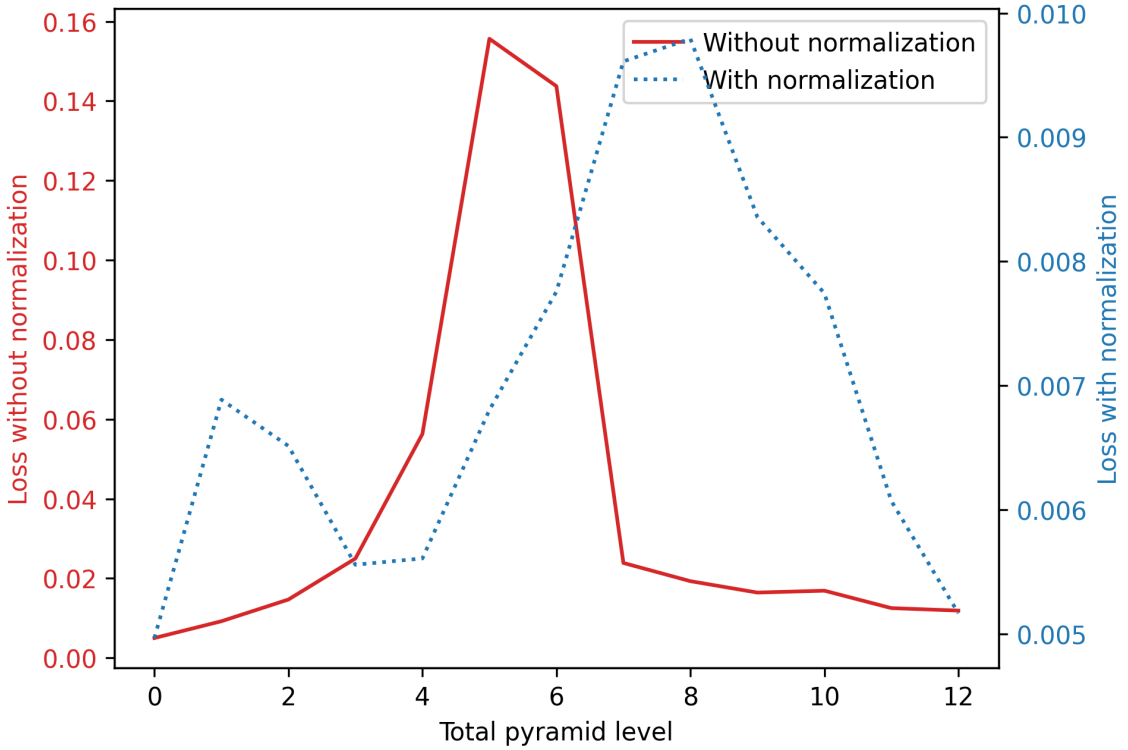


Figure 4.12: Loss by pyramid level for the trained model, with and without normalization. The loss was normalized while training the model.

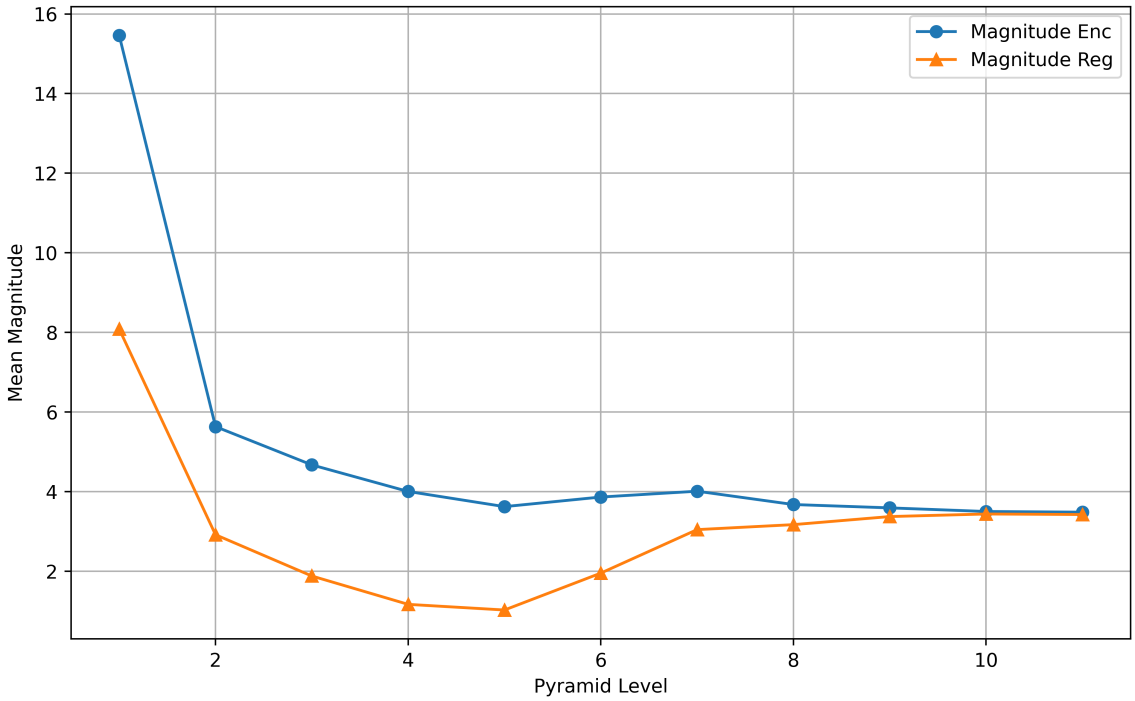


Figure 4.13: Average magnitude of PyRvNN representations across different pyramid levels. The figure shows Magnitude Enc and Magnitude Reg, corresponding to the autoencoding and autoregressive head output magnitudes, respectively. Left and right representations are averaged into these two plots due to their very similar magnitudes.

Formally, let $y_{\text{enc,left}}$, $y_{\text{enc,right}}$, $y_{\text{reg,left}}$, and $y_{\text{reg,right}}$ denote the dense output vectors from the autoencoding and autoregressive heads for the left and right outputs, respectively. The magnitude M is computed as:

$$M = \begin{cases} 1, & \text{if one-hot} \\ \frac{1}{4}\left(\|y_{\text{enc,left}}\| + \|y_{\text{enc,right}}\| + \|y_{\text{reg,left}}\| + \|y_{\text{reg,right}}\|\right), & \text{otherwise.} \end{cases} \tag{4.9}$$

where $\|\cdot\|$ represents the ℓ_2 norm. The final magnitude M is averaged across all instances in the batch.

To ensure consistent learning across pyramid levels, the loss function $\mathscr{L}$ is normalized as:

$$\mathscr{L} \leftarrow \frac{\mathscr{L}}{M}. \tag{4.10}$$

This approach guarantees that pyramid levels with larger representation magnitudes do not dominate the learning process.

4.3.4 Training Setup

For optimizing the model, we employ the Adam optimizer from the tf.keras.optimizers module with its parameter configuration: beta_1=0.9, beta_2=0.999, epsilon=1e-08, and amsgrad=False. Adam (Adaptive Moment Estimation) combines the advantages of both the AdaGrad and RMSProp algorithms, making it highly effective for problems with sparse gradients and noisy data. The parameters beta_1 and beta_2 control the exponential decay rates for the first and second moment estimates of the gradients, respectively, which helps stabilize the optimization process. The epsilon term ensures numerical stability during division by very small gradient values, while the amsgrad parameter, when set to True, improves convergence in certain cases by preventing the moment estimates from growing too large, though we opt to keep it disabled in this setup.

To control overfitting and enhance generalization, we apply L2 regularization (with lambda set to 0.001) to the model's layers. L2 regularization works by adding a penalty proportional to the square of the magnitude of the weights to the loss function, effectively discouraging excessively large weight values. This regularization helps prevent the model from becoming too complex and reduces the risk of overfitting the training data, especially in deep learning models with many parameters.

The training schedule is designed to progressively expose the model to pyramid levels. Initially, the model is trained on only subword-level pyramid levels, starting with the 0th level and incrementally moving up to higher levels (1st, 2nd, and so on). After mastering these subword levels, the training extends to include phrase-level pyramids, again starting at the 0th level and progressively adding more phrase levels. Throughout this process, the model is trained

on a maximum of 10 subword pyramid levels and up to 3 phrase pyramid levels.

To further support effective learning, the learning rate begins at 0.001 and is decreased to 0.0005 as the model encounters more pyramid levels. This progressive decrease allows for more refined adjustments to the model's weights as it is exposed to higher pyramid levels, ensuring better convergence without overshooting or causing instability in the learning process.

In Figures 4.14 and 4.15, we visualize the loss values at the beginning of the training process. Figure 4.15 presents the same loss curves as Figure 4.14 but scaled up to highlight smaller loss variations. The plots display four types of losses: Onehot Enc, Onehot Reg, Dense Enc, and Dense Reg, corresponding to the one-hot and dense representations in the autoencoding and autoregressive heads, respectively. The total loss is computed as the mean of these four loss types.

The effect of the progressive training schedule is evident in the spikes observed across different loss components. The training schedule follows these stages:

- From iteration 0 to 5000, the model is trained solely on the 0th subword pyramid level.
- From iteration 5000 to 9000, training extends to subword levels 1 through 9, with each level trained for 500 iterations.
- From iteration 9000 to 15000, the model is trained on all nine subword levels as well as phrase levels 1 and 2.

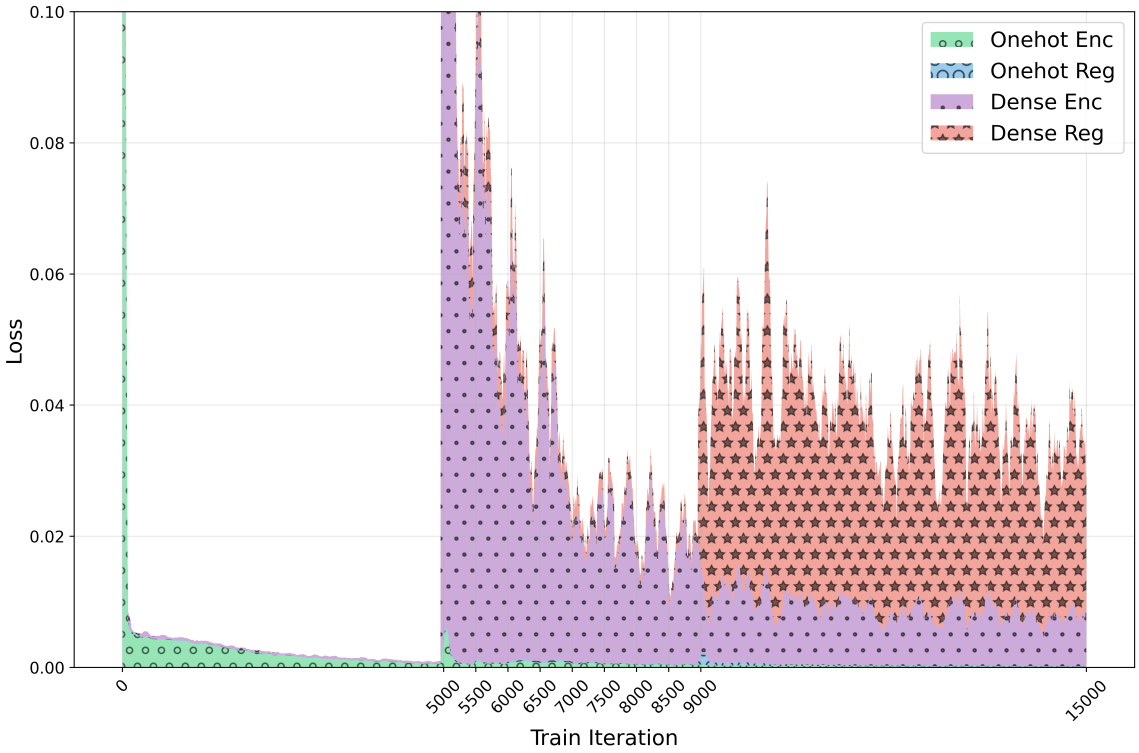


Figure 4.14: Loss curves during the initial training phase, showing the impact of the progressive training schedule across different loss components.

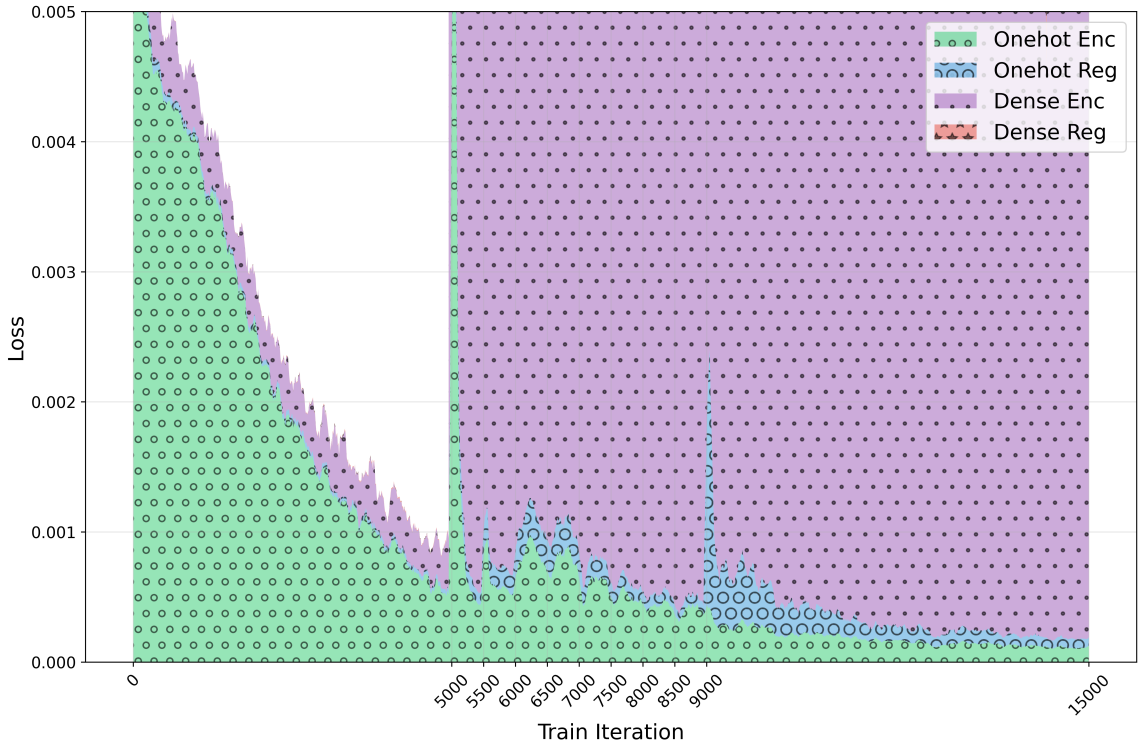


Figure 4.15: Same loss curves as in Figure 4.14, but scaled up to emphasize smaller loss variations.

4.4 Evaluations

To evaluate PyRvNN, we employ a combination of intrinsic and extrinsic evaluation tasks that measure different aspects of the model's performance and the quality of the learned text representations. These tasks provide insight into both the model's internal capability (intrinsic evaluation), such as decodability and compositionality, and its effectiveness on downstream tasks (extrinsic evaluation), where text representations are used for more complex NLP tasks.

The first and most direct evaluation of PyRvNN is decodability. Decodability measures how well PyRvNN can decode a subword, word, or phrase from its internal representation. PyRvNN recursively embeds subwords pair-wise level-by-level into a word representation and continues this process to form phrase-level representations. It can then decode these representations back into subwords, demonstrating its ability to capture hierarchical structures. We monitor decodability throughout the training process, as it serves as a core performance indicator. A higher decodability score implies that the model's internal representations preserve essential linguistic information during both the encoding and decoding phases.

While decodability is a direct reflection of the model's performance, additional evaluation through downstream tasks is crucial to assess the usefulness of PyRvNN's learned representations in real-world scenarios.

Downstream tasks evaluate how well the text embeddings produced by PyRvNN perform on external tasks. These tasks assess whether the learned representations are generalizable and

useful for various linguistic applications. Downstream evaluations are essential for text embedding models because they reveal how well the internal representations encode information that is beneficial for other NLP tasks, beyond just the original structure that the model was trained on.

The downstream tasks used to evaluate PyRvNN are:

- **Universal part-of-speech tagging (UPOS)** and **dependency relation labeling (DEPREL)**. Using these tasks we evaluate the effectiveness of PyRv for embedding composition (compositionality property of PyRv) using fastText word embeddings (as input for PyRvNN). UPOS tagging assigns core grammatical categories (e.g., noun, verb, adjective) to each token in a sentence, while DEPREL labeling identifies syntactic dependencies between words, such as subject-object relationships.
- **Memorization**. While PyRv is primarily designed to produce structured embeddings by focusing on properties like compositionality, hierarchy, and decodability, rather than for direct text generation like Large Language Models (LLMs), we evaluate its embeddings' utility in a sequential prediction context. Perplexity, a standard metric for generative language models, is less suitable for PyRv given that its autoregressive component primarily serves to enhance embedding quality rather than to enable fluent text generation. Instead, in this task, we test the model's ability to memorize a small dataset by training a simple RNN language model to predict the next token in a sequence, using the embeddings as input. The RNN model consists of fully connected layers at the input and output and multiple long short-term memory (LSTM) layers in between. We train the language model on 500 random Wikipedia paragraphs, using token embeddings produced by PyRvNN (and FastText for comparison). This task measures how well the PyRvNN-generated representations support next-token prediction when utilized by a downstream sequential model.
- **Plagiarism Detection**. For evaluating PyRvNN on paraphrased plagiarism detection, we use a dataset of 200,767 paragraphs from Wikipedia, with 50% of them paraphrased by the SpinBot API. We evaluate embeddings by training a support vector machine (SVM) classifier to distinguish between original and paraphrased text. Each paragraph is represented by averaging the token embeddings (produced by PyRvNN or baseline models like GloVe, Word2Vec, and FastText).
- **Readability Classification**. In this task, we assess the model's ability to classify texts based on their readability. Using the CLEAR dataset, which categorizes texts by readability level, we train a Naive Bayes classifier on embeddings generated by PyRvNN and compare them with BERT (base model, uncased), FastText, and Term Frequency-Inverse Document Frequency (TFIDF) embeddings. This task evaluates how well the embeddings capture text complexity and readability.

4.5 Model Limitations and Constraints

While the PyRvNN demonstrates promising capabilities in text representation learning, several limitations and constraints emerged during its design, training, and evaluation. This section highlights these challenges and proposes potential avenues for future research and improvement.

Decoding Accuracy and Recursion Depth. As observed in the decodability evaluation (5.1), the accuracy of decoding representations back into text diminishes at higher pyramid levels, especially for longer phrases. This indicates that the model's ability to effectively capture and preserve information is adversely affected by increasing recursion depth. While increasing the dimensionality of representations may improve performance at deeper levels, it also significantly raises computational complexity and memory requirements. Future research could explore alternative strategies, such as hierarchical attention or dynamic weighting mechanisms, to maintain high decoding accuracy without excessive computational overhead.

Memory Constraints and Batch Size. The hierarchical representations in PyRvNN impose high memory demands, restricting the model to training on batches containing a single paragraph of text. While this approach manages memory usage, it limits the diversity of training examples in each batch, potentially affecting learning efficiency and generalization. Techniques such as gradient checkpointing, which trades off computation for reduced memory usage by selectively storing and recomputing activations, could enable larger batch sizes and improve both training speed and stability. This approach, commonly used in large language model training, represents a promising direction for addressing memory constraints in PyRvNN.

Vanishing/Exploding Gradients and Training Instability. The recursive nature of PyRvNN makes it vulnerable to vanishing and exploding gradient issues, leading to instability during training. To mitigate this, the training procedure employs multi-level loss aggregation and gradient normalization, where losses are computed and normalized at each pyramid level. This stabilizes training, as visualized in Figures 4.10, 4.11, and 4.12. However, advanced optimization techniques, such as gradient clipping, alternative activation functions, or adaptive learning rate strategies, warrant further investigation to enhance training stability and convergence.

Computational Cost and Slow Training. The recursive and hierarchical nature of PyRvNN necessitates the generation of input-output pairs for both the autoencoder and autoregression heads at each pyramid level. This significantly increases computational cost and slows down training.

Limitations in Handling Long-Range Dependencies. Although PyRvNN effectively captures hierarchical relationships within text, it struggles with long-range dependencies, a challenge common to many recursive models. Relationships between distant words in a sentence,

which are better modeled by attention mechanisms, remain difficult for PyRvNN. Integrating attention mechanisms into the PyRvNN architecture could enhance its ability to process longer sequences and capture more nuanced relationships, addressing this limitation.

Chapter 5

Experiments and Results

To assess the effectiveness of the proposed Pyramidal Recursive Neural Network (PyRvNN) and the quality of its learned text representations, this chapter presents a series of experiments designed to evaluate both the intrinsic and extrinsic properties of the model. These experiments provide a comprehensive view of PyRvNN's capabilities, focusing on its internal mechanisms, such as representation decodability and compositionality, and its performance on downstream natural language processing (NLP) tasks. PyRvNN is designed primarily to investigate hierarchical and compositional representation learning from subword units in a self-supervised manner, rather than to directly compete with large-scale pretrained contextual embedding models on all downstream tasks. Some of the results presented in this chapter have been published in our previous work ([143]).

The chapter begins with an evaluation of the representation **decodability** property (5.1), assessing how well PyRvNN can decode subwords, words, and phrases from their internal representations. This property serves as a crucial measure of the interpretability and reconstruction quality of the model's learned representations.

Next, the chapter explores the representation **compositionality** property (5.2). This evaluation specifically isolates PyRvNN's ability to compose multiple pre-existing word embeddings (fastText embeddings, which are non-contextual) into meaningful representations that effectively capture the structure and semantics of text. This setup allows for a focused assessment of PyRv's compositional mechanism itself. The compositionality experiments include tasks such as universal part-of-speech (UPOS) tagging and Universal dependency relation (DEPREL) labeling, highlighting the model's proficiency in leveraging hierarchical representation learning.

Beyond intrinsic evaluations, the chapter examines the practical utility of PyRvNN's text representations through several extrinsic tasks. In these tasks, PyRvNN is used to embed text into representations that are subsequently fed into simpler models to perform the following:

- **Memorization** Task (5.3): Representations generated by PyRvNN are used by a sequence prediction model to memorize a small dataset of Wikipedia paragraphs and predict the

next token in a sequence. FastText, another subword-based non-contextual embedding method, serves as a suitable baseline here.

- **Plagiarism** Task (5.4): Following the methodology of Wahle et al. [144], which uses averaged single-word embeddings for classification, PyRvNN embeddings (representing single words composed from subwords) are input to a classification model. This evaluates PyRvNN's single-word representation quality in a setting directly comparable to their word-level baseline evaluations.
- **Readability** Task (5.5): Representations from PyRvNN are utilized by a classifier to determine the readability levels of texts in the CLEAR dataset.

5.1 Representation Decodability Property

Representation decodability property simply means that a representation can be decoded back into the text that it represents. Decodability can serve as a useful metric for evaluating the quality of text representations and making it easier to understand and interpret what the model has learned.

The first task we evaluate PyRvNN on is decodability; how well PyRvNN decodes a subword/word/phrase from a single representation. Decoding accuracy was the main measure used for tuning PyRvNN hyperparameters during its training on the Wikipedia dataset. PyRvNN is able to recursively embed subwords pair-wise level-by-level into a single word representation, and then recursively embed word representations pair-wise level-by-level into a single representation of a phrase up to three words long. It can then decode that representation down to subwords again.

In this evaluation, we demonstrate the model's capacity to decode representations into the original input text from any level of the generated pyramid.

5.1.1 Results

We measure decoding accuracy by converting representations from various pyramid levels back to text and comparing it with the original encoded text. Specifically, accuracy is measured as the percentage of subword tokens that are correctly decoded. This evaluation was performed on 10,000 paragraphs randomly sampled from Wikipedia. Figure 5.1 depicts accuracy for different word lengths and three phrase pyramid levels, excluding subword levels. Table 5.1 shows the same.

This evaluation reveals the effective learning of three crucial properties by the model: hierarchical representation, representation compositionality, and representation decodability. Representation decodability is directly evaluated with decoding accuracy. Representation compositionality is implicitly proven to work because representations are composed together on each

Table 5.1: Decodability accuracy for each word length (token count) and three phrase pyramid levels, tested on Wikipedia texts.

	3 Tokens	4 Tokens	5 Tokens	6 Tokens	7 Tokens	8 Tokens
Phrase level 1	0.9994	0.9995	0.9992	0.9938	0.6253	0.0326
Phrase level 2	0.9674	0.9870	0.9645	0.5073	0.0192	0.0125
Phrase level 3	0.9479	0.6003	0.0955	0.0144	0.0108	0.0064

level to produce representations for the next level. The representations are composed hierarchically, so the accuracy of decoded text proves hierarchical representation works well.

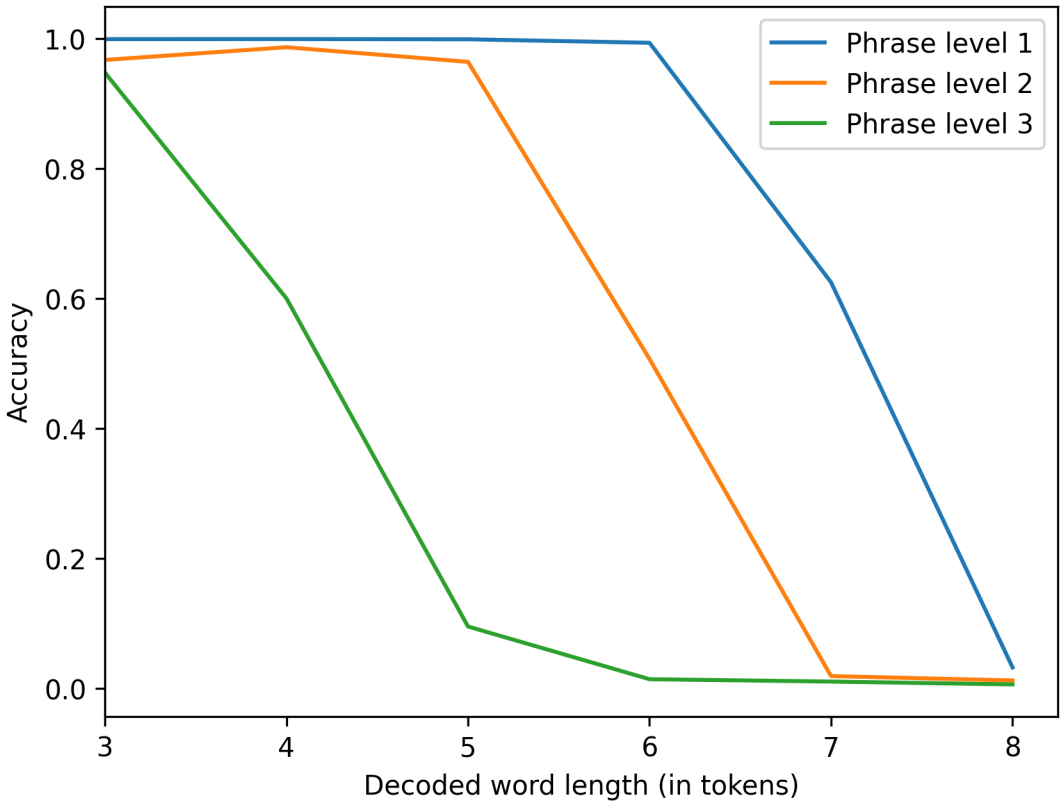


Figure 5.1: Decodability accuracy for each word length (token count) and three phrase pyramid levels, tested on Wikipedia texts.

To assess how decodability performance generalizes to texts different from the training data (Wikipedia), we also evaluated it on texts from Amazon reviews. Specifically, we used a random sample of 10,000 reviews from the test set of the dataset by McAuley and Leskovec [145]. As shown in Table 5.2 and Figure 5.2, the performance is practically identical to that observed on Wikipedia texts.

We observed in Figures 5.1 and 5.2 that the decodability accuracy experiences a sharp drop when words are composed of approximately six or seven tokens. We hypothesize that the

Table 5.2: Decodability accuracy for each word length (token count) and three phrase pyramid levels, tested on texts from Amazon reviews.

	3 Tokens	4 Tokens	5 Tokens	6 Tokens	7 Tokens	8 Tokens
Phrase level 1	0.9991	0.9993	0.9989	0.9929	0.6095	0.0117
Phrase level 2	0.9864	0.9926	0.9580	0.4660	0.0220	0.0065
Phrase level 3	0.9724	0.5219	0.0814	0.0145	0.0102	0.0080

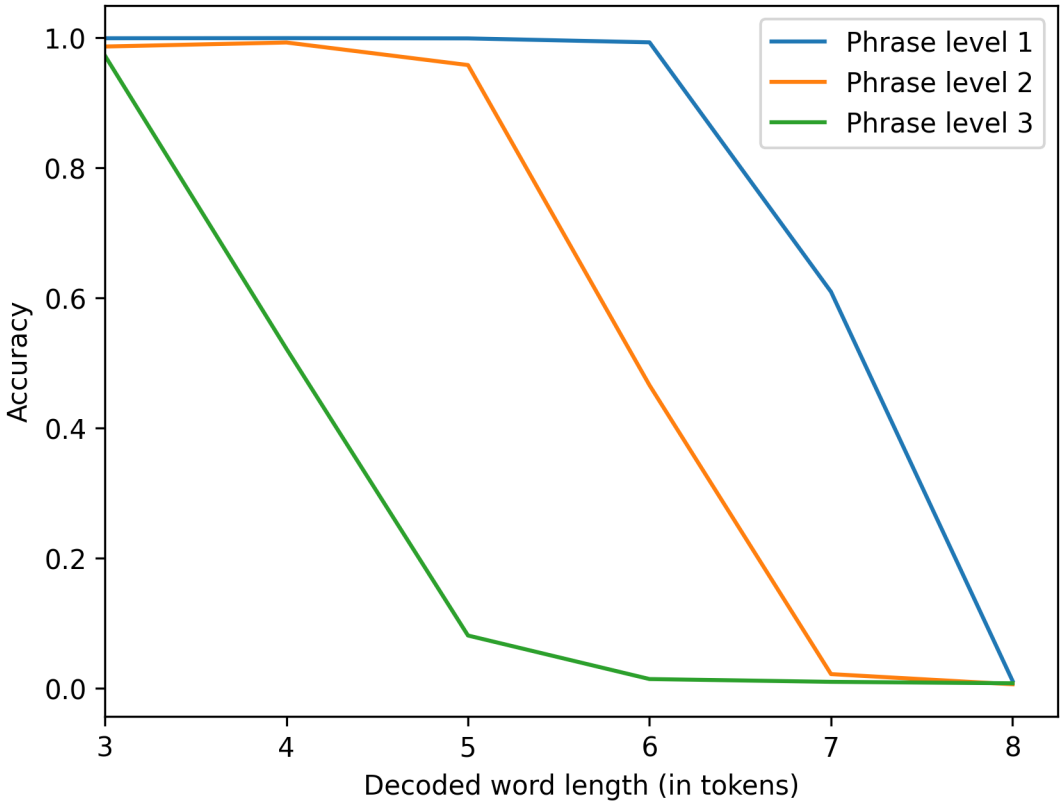


Figure 5.2: Decodability accuracy for each word length (token count) and three phrase pyramid levels, tested on texts from Amazon reviews.

primary reason for this phenomenon is the relative scarcity of words with such lengths in the training corpus. Consequently, the PyRvNN model receives less training exposure specifically at its higher pyramid levels, as these levels are reached less often since they primarily process longer words.

To investigate this hypothesis, we analyzed the distribution of word lengths in terms of token counts. Figure 5.3 displays a histogram illustrating this distribution, calculated from a sample of 10,000 paragraphs from the Wikipedia dataset. As the figure shows, the frequency of words decreases as the number of tokens per word increases, lending support to our explanation for the observed accuracy drop.

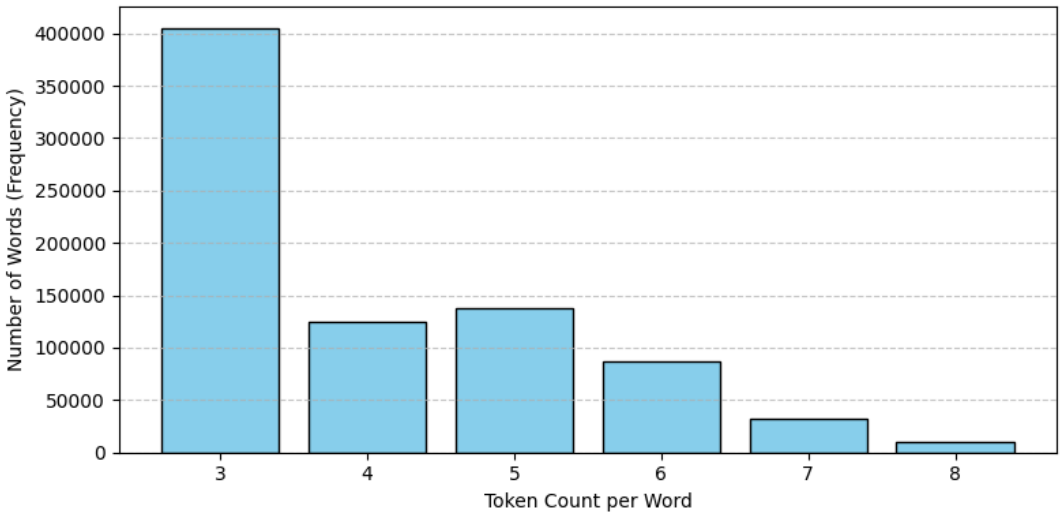


Figure 5.3: Histogram illustrating the frequency distribution of the number of tokens per word. Data was computed using 10,000 paragraphs from the Wikipedia dataset. The histogram is truncated at 8 tokens per word. The plot shows that longer words (e.g., 6+ tokens) are significantly less frequent, supporting the hypothesis that reduced training data for these lengths contributes to the accuracy drop seen in Figure 5.1.

5.2 Representation Compositionality Property

One of the primary properties of the PyRv method is representation compositionality, which enables the composition of multiple embeddings into a coherent, semantically rich representation. A model with such a property could capture the meaning of a complex text by combining the meanings of its constituent parts in a structured and semantically meaningful way.

In this section, we explore the composition of word embeddings to create richer, more meaningful representations of multi-word units. Existing methods, such as averaging word embeddings, provide simple and efficient approaches. However, they often fail to capture the complexity of multi-word interactions. To address this, we employ the Pyramidal Recursive learning (PyRv) method, which recursively combines word embeddings into unified representations. Originally developed for constructing representations hierarchically from subwords to

phrases, PyRv is well-suited for progressively merging individual word vectors into phrase vectors. We evaluate the effectiveness of PyRv specifically for its embedding composition mechanism using pre-trained fastText embeddings as input. This experimental design allows us to isolate and assess PyRv's ability to combine existing non-contextual word embeddings, rather than comparing its end-to-end sentence representation capabilities against state-of-the-art contextual sentence encoders. The focus here is on the compositional transformation PyRv applies. The evaluation is performed on the UPOS tagging task and DEPREL labeling task.

In many real-world applications understanding text at the word level alone is insufficient. The need to represent larger linguistic units, such as phrases or sentences, necessitates techniques for combining word embeddings into more complex structures. Word composition, which involves aggregating individual word embeddings to represent multi-word expressions or entire sentences, serves this purpose. By integrating information from multiple word vectors, compositional methods aim to capture both the meanings of individual words and the syntactic and semantic interactions between them, particularly in morphologically complex languages, such as Croatian, which was used for evaluation in this section.

There are several established methods for combining word embeddings. Simple techniques include element-wise operations such as addition, averaging, or multiplication, which produce a composite vector by leveraging individual word vectors (such as [129] and [130]). These methods, while computationally efficient, may fail to fully capture the complexity of phrase or sentence meaning. More sophisticated approaches employ weighted combinations, context-aware methods, or syntactic structures to improve the expressiveness of the resultant embeddings (such as [25] and [98]).

While transformer-based models, such as BERT [4] and GPT [125], offer robust pre-trained sentence embeddings by learning deep contextual representations, they are often computationally expensive and may not always align with the specific needs of certain tasks. Word composition methods provide a more lightweight and flexible alternative, especially in cases where transparency and control over the aggregation process are crucial. Additionally, word-level composition techniques can better retain the granularity of individual word meanings, which is sometimes diluted in sentence-level embeddings produced by transformer models.

Word composition provides a valuable approach for constructing meaningful representations of multi-word units, balancing computational efficiency and interpretability. These methods remain relevant, particularly in domains where sentence embedding techniques may obscure important details or where domain-specific customization of embedding composition is required.

In this work, we leverage the PyRv method to compose multiple word embeddings into unified representations. PyRv facilitates structured composition through its hierarchical learning approach, recursively combining representations at each level of abstraction. Initially developed for constructing representations from tokens (subwords) up to phrases (as illustrated

in Figure 4.1), PyRv is well-suited for combining word embeddings by progressively merging individual word vectors into phrase vectors (as is shown in Figure 5.4).

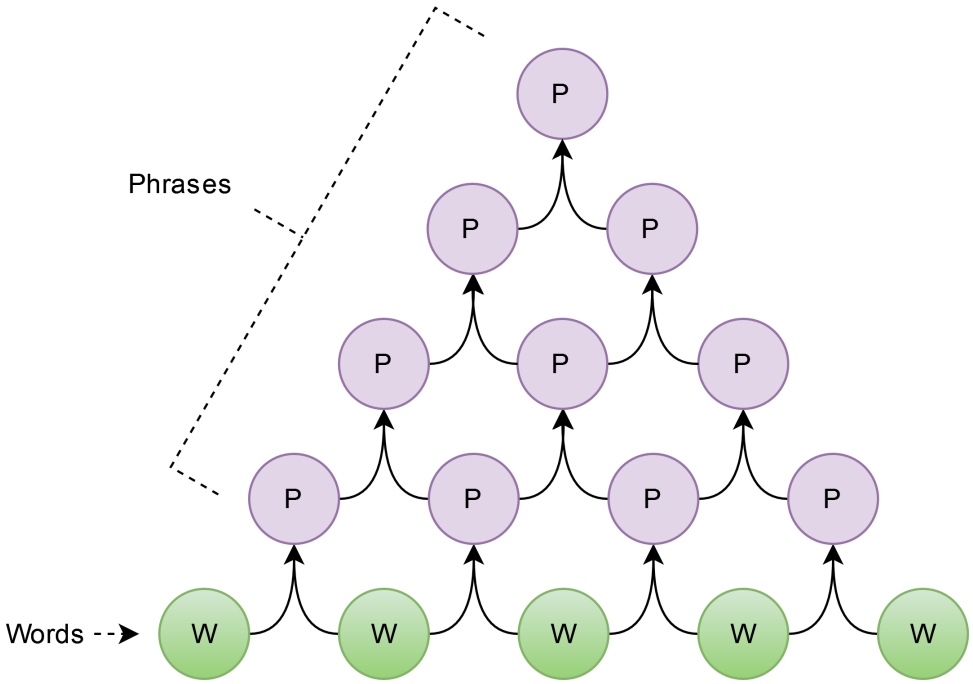


Figure 5.4: A visual representation of pyramidal recursion in the PyRv+FT method. The lowest-level nodes correspond to fastText-embedded words. As we move upward, the nodes represent combined word embeddings, capturing phrase-level meanings.

5.2.1 Embedding method

In this subsection, we introduce word embeddings and common composition techniques, followed by a detailed explanation of how the PyRv method is applied to compose word embeddings.

Word embedding. Word embeddings, such as those produced by fastText [2], are commonly used to represent individual words. To represent multiple words, basic techniques like averaging or concatenation can be applied. However, both approaches come with notable limitations.

When averaging embeddings (e.g., taking the mean of multiple word vectors), important information, particularly word order, is lost. Concatenation preserves all information but presents two significant challenges.

First, concatenated embeddings vary in dimensionality depending on the number of words, which complicates their use as input for models that require a fixed input size. Second, the dimensionality of concatenated representations can grow excessively large. For instance, a 5-word phrase embedded with 300-dimensional fastText results in a 1500-dimensional vector (5x300).

In our evaluation, we use averaging to compose Croatian fastText embeddings [146] to maintain a consistent dimensionality across all methods, ensuring that the results are comparable.

Composing fastText word embeddings with PyRv. PyRv is a method designed to construct hierarchical representations of text, moving progressively from low-level units such as characters or subwords to higher-level representations such as words, phrases, sentences, and even paragraphs. PyRv combines representations recursively, forming increasingly abstract and semantically rich embeddings at each level of the hierarchy.

To address the limitations of averaging and concatenation, we explore the use of PyRv for composing multiple word embeddings into a single, unified representation. Unlike our previous work on PyRv [143], where the recursion starts from subwords or tokens, in this study we begin with word embeddings produced by fastText. This hybrid approach is referred to as PyRv+FT.

For this experiment, we use pre-trained Croatian fastText word vectors [146] to embed words, which are then recursively combined into phrase embeddings via the PyRvNN model. The PyRv+FT embeddings are trained on Croatian Wikipedia texts (Wikimedia public dump, January 11, 2020) for 10 epochs.

We evaluate PyRv+FT on two downstream tasks, described in detail in the next subsection. Through this process, we investigate how PyRv improves the compositionality of word embeddings while maintaining manageable dimensionality and enhancing representational quality.

5.2.2 Evaluation

In this subsection, we describe the evaluation of fastText and PyRv+FT embeddings on two NLP tasks: UPOS tagging and DEPREL labeling. UPOS tags represent core grammatical categories such as nouns, verbs, and adjectives, while DEPREL captures the syntactic relationships between words in a sentence, indicating dependencies like subjects, objects, and modifiers.

For this evaluation, we use the hr500k 2.0 dataset [147], a Croatian corpus with labeled data for both UPOS and DEPREL tasks (amongst others). This dataset contains 901 texts, 24,763 sentences, and a total of 499,635 tokens.

Additionally, we perform a qualitative analysis of PyRv+FT embeddings by visualizing the representation space to gain deeper insights into its structure and characteristics.

Experiment setup. To assess the performance of different embedding methods on the downstream tasks of UPOS and DEPREL, we use a multi-layer perceptron (MLP) model with one hidden layer. The hidden layer consists of 1,000 neurons and uses the ReLU activation function. The input to the model is a 300-dimensional vector (the size of both fastText and PyRv+FT embeddings). The output layer, with softmax activation, adjusts to the number of classes in each task: 17 classes for UPOS and 37 classes for DEPREL. The MLP is trained on the training set

of the hr500k dataset and evaluated on its test set. Each evaluation is conducted by training the MLP for one epoch.

The embedding procedure remains consistent across both UPOS and DEPREL tasks, differing only in the MLP output labels.

For strategies that incorporate context, a window of N words centered around the target word is considered. We select N as an odd number to ensure the target word is central. Based on the nature of the tasks and preliminary observations, N=3 (target word + 1 neighbor on each side) is used for the UPOS task, as wider contexts tend to degrade performance for this relatively local task. For the more complex DEPREL task, which benefits more from surrounding syntactic cues, N=5 (target word + 2 neighbors on each side) is employed, as this window size was found to be beneficial without introducing excessive noise from overly broad contexts.

The method of embedding a word from a sentence depends on the embedding strategy employed:

- **fastText 1 word**: Embeds only the target word, ignoring its surrounding context.
- **mean fastText N words**: Represents the target word by embedding all N words (the target word and its N-1 neighboring context words) separately and averaging the embeddings to obtain a final representation.
- **weighted average fastText N words**: Represents the target word by computing a weighted average of the embeddings of the N-word window centered around the target. The weights are derived from a symmetric Gaussian distribution centered at the middle word. The standard deviation of the Gaussian is defined as $\sigma = N \cdot \lambda$, where N is the window size and λ is a tunable hyperparameter, referred to as the Gaussian STD factor. Smaller values of λ result in the middle word receiving most of the weight, reducing the influence of surrounding context.
- **PyRv+FT N words**: Embeds each word using fastText, but instead of averaging the N embeddings, it recursively combines them using the PyRvNN model to generate a single, unified embedding.

Quantitative results: UPOS. Part-of-speech tagging is a relatively simple task where the surrounding word context does not provide significant benefits for classification. We include UPOS evaluation primarily to demonstrate how averaging fastText word embeddings can degrade downstream performance, while combining fastText word embeddings using PyRvNN preserves much of the embedding quality.

When using fastText to embed a single word without considering its context, we achieve an accuracy of 95%. However, averaging fastText embeddings over three words leads to a substantial drop in performance, with accuracy falling to 61%. Weighted averaging of three word embeddings using a Gaussian kernel achieves the highest accuracy of 95% when the Gaussian STD factor is set to 0.15. In this configuration, the middle word dominates the representation,

receiving approximately 86% of the total weight. This setting results in performance equivalent to embedding the target word alone. As shown in Figure 5.5, increasing the STD factor distributes the weights more evenly across the context window, degrading performance toward the level of simple mean averaging (61% accuracy). In contrast, when combining fastText embeddings for three words using PyRvNN, the performance degradation is minimized, yielding an accuracy of 93%. These results highlight how PyRvNN can effectively mitigate the loss of information that occurs when averaging word embeddings. Detailed results are shown in Table 5.3.

Table 5.3: UPOS results, Macro and Weighted averages.

	Accuracy	**Precision**		**Recall**		**F1 score**	
		M. avg	W. avg	M. avg	W. avg	M. avg	W. avg
fastText 1 word	**0.95**	0.91	**0.95**	**0.89**	**0.95**	0.89	**0.95**
mean fastText 3 words	0.61	0.57	0.61	0.59	0.61	0.57	0.61
w. avg. fastText 3 words	**0.95**	**0.93**	**0.95**	**0.89**	**0.95**	**0.91**	**0.95**
PyRv+FT 3 words	0.93	0.9	0.93	**0.89**	0.93	0.9	0.93

Quantitative results: DEPREL. The dependency relation task is more complex than UPOS, as it requires understanding the syntactic relationships between words. In this case, enriching word embeddings with surrounding context can significantly improve classification performance. Our primary objective here is to assess PyRvNN's effectiveness as a method for composing pre-existing word embeddings (fastText) for this context-sensitive task, rather than to build a state-of-the-art dependency labeler.

When embedding a single word using fastText (without its context), the model achieves an accuracy of 71%. However, averaging five fastText word embeddings, including the target word and its four neighbors, results in a sharp decline in performance, with accuracy dropping to 34%. This reduction in accuracy reflects how averaging word embeddings leads to the loss of important information, including word order and syntactic structure. Weighted averaging of five fastText word embeddings using a Gaussian kernel achieves a maximum accuracy of 74% when the STD factor is 0.1, corresponding to a central weight of 79%. This surpasses the performance of using only the target word embedding (71%) and demonstrates that, for DEPREL, limited context improves classification. However, performance declines as the STD factor increases and the weights become more uniform, reflecting the same degradation trend observed with mean averaging (34%). These results are visualized in Figure 5.6. By contrast, composing five fastText word embeddings using PyRvNN boosts accuracy to 77%, demonstrating the method's ability to capture more nuanced relationships between words.

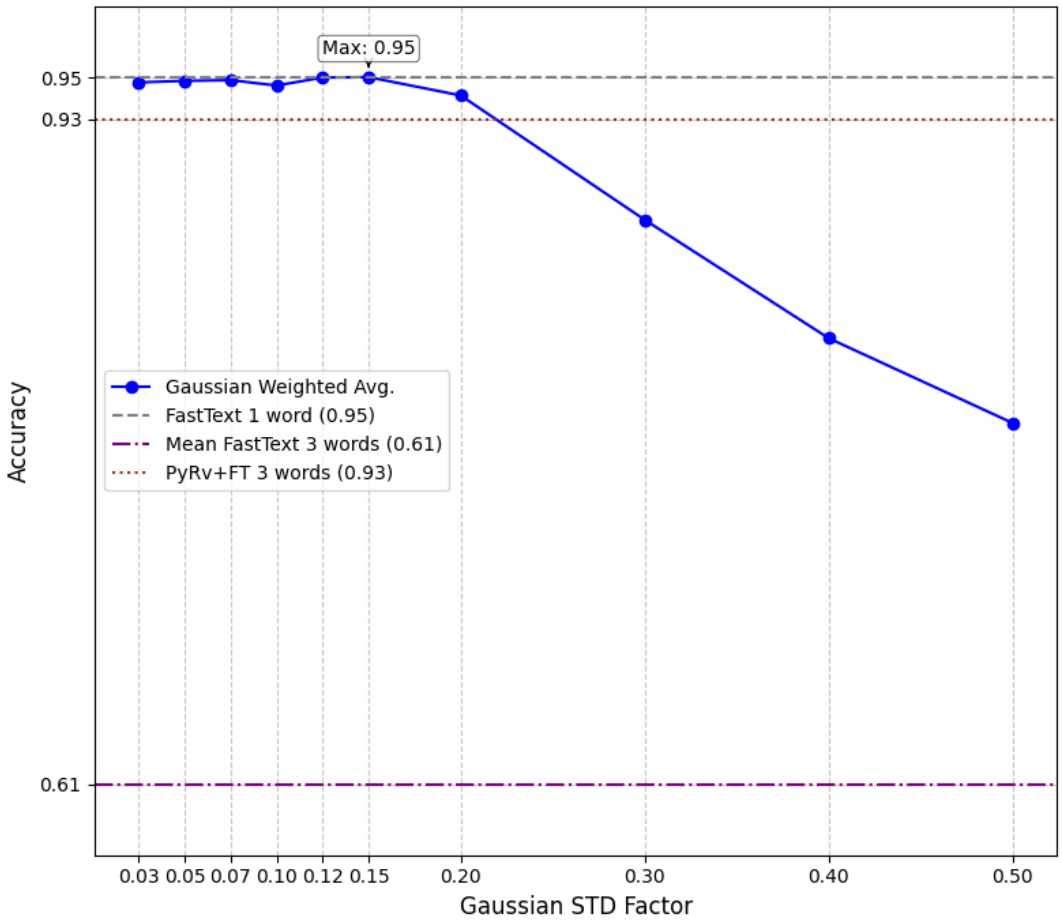


Figure 5.5: UPOS task, weighted average: accuracy as a function of the Gaussian STD factor. As the STD factor increases, the weights assigned to context words become more uniform, degrading performance. The best result (95%) is obtained with STD factor 0.15, where the middle word dominates the representation (86% of total weight). Plots for F1 score are shown in Appendix B: Figures B.1 and B.2.

Evaluation results are presented in Table 5.4 with more detailed results (by class) available in Appendix B: Tables B.1, B.2, and B.3.

Table 5.4: DEPREL results, Macro and Weighted averages.

	Accuracy	**Precision**		**Recall**		**F1 score**	
		M. avg	W. avg	M. avg	W. avg	M. avg	W. avg
fastText 1 word	0.71	0.52	0.68	0.48	0.71	0.47	0.68
mean fastText 5 words	0.34	0.25	0.34	0.19	0.34	0.19	0.31
w. avg. fastText 5 words	0.74	0.49	0.73	0.46	0.74	0.46	0.72
PyRv+FT 5 words	**0.77**	**0.58**	**0.77**	**0.55**	**0.77**	**0.56**	**0.76**

Figure 5.7 presents bar plots comparing F1 scores between two composition methods, mean averaging and PyRv, relative to single word fastText embeddings' performance. The left plot 5.7a illustrates the F1 score ratio between "mean fastText 5 words" and "fastText 1 word", while the right plot 5.7b contrasts "PyRv+FT 5 words" with "fastText 1 word".

In the following analysis, we focus on three notable classes: punctuation, conjuncts, and adnominal clauses. These were selected because punctuation classification does not rely on context (isolated punctuation tokens in this dataset are generally unambiguous), classifying conjuncts without context is nearly impossible, and for adnominal clauses, context is helpful but averaging tends to degrade performance.

Punctuation (punct) refers to punctuation marks such as ".", "?", "!", and ",". Since punctuation is straightforward to classify without context (within the hr500k dataset), a single fastText embedding for the target word alone achieves a perfect F1 score of 1. Averaging five fastText embeddings (the target word and its four neighboring words) significantly degrades performance, reducing the F1 score to 0.61. However, using PyRv to compose these embeddings preserves the high performance, maintaining an F1 score of 1. Weighted averaging with Gaussian weighting also preserves high performance for the punctuation class, achieving an F1 score of 1 when the context is strongly downweighted.

Conjunct (conj) denotes a relation between elements connected by coordinating conjunctions like "and," "or," or ",". In coordinate structures, the first element is conventionally treated as the head, with subsequent elements connected through the conj relation. For example, in the sentence "Bill is *big* and *honest*," the word "honest" is labeled as conj (connected to "big"). Similarly, in "He *came* home, *took* a shower and immediately *went* to bed," the words "took" and "went" are both labeled as conj (connected to "came"). Classifying conjuncts accurately requires contextual information. A single fastText word embedding, without any context, yields a poor F1 score of 0.03. Averaging the embeddings of the target word and its four neighbors improves performance significantly, achieving an F1 score of 0.32 (a 10.8x improvement over

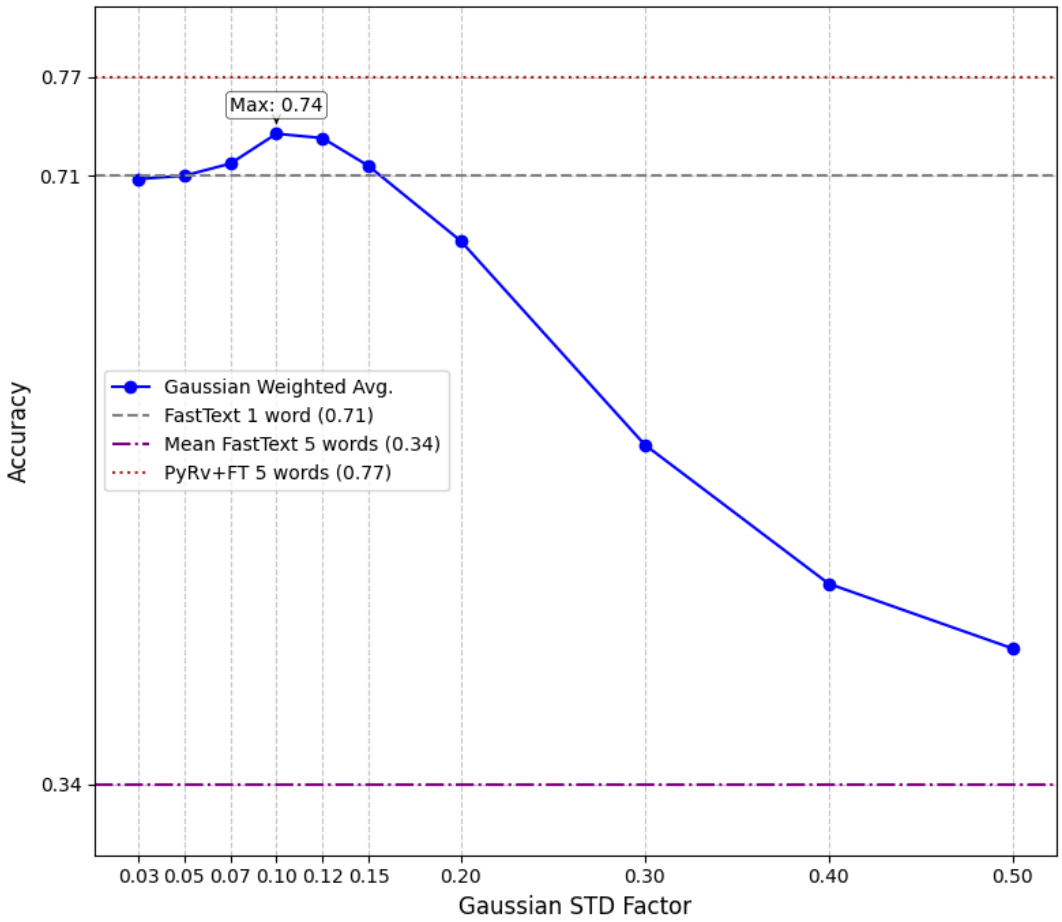
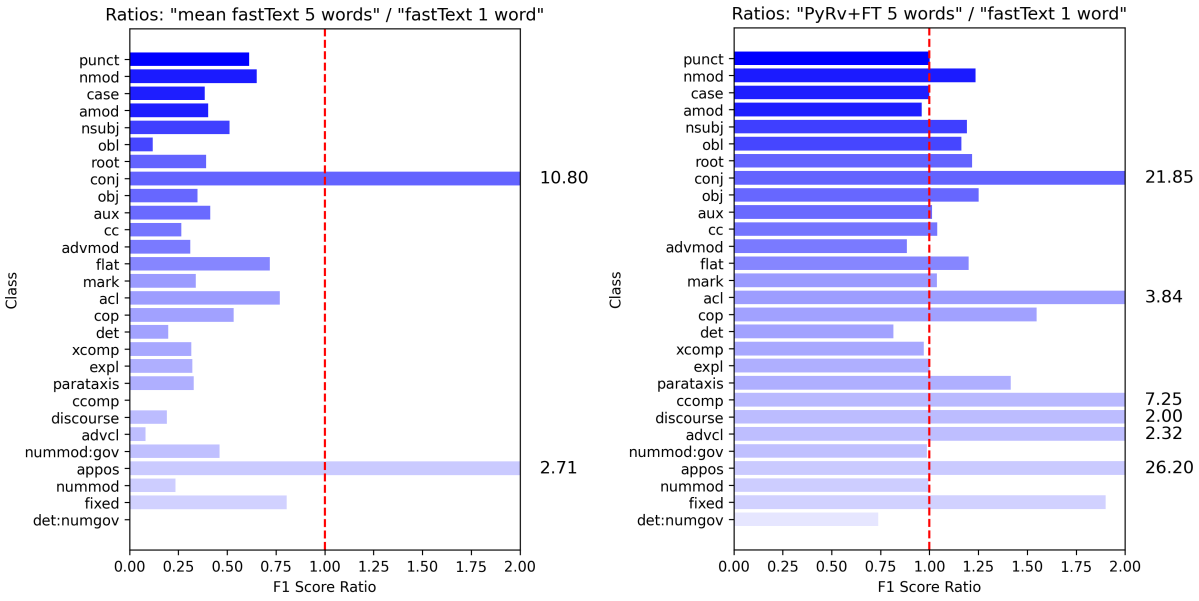


Figure 5.6: DEPREL task, weighted average: accuracy as a function of the Gaussian STD factor. Maximum performance (74%) is reached at STD factor 0.1, where the middle word receives 79% of the total weight. As the STD factor increases, the performance degrades toward the baseline of uniform averaging (34%). Plots for F1 score are shown in Appendix B: Figures B.3 and B.4.



(a) "mean fastText 5 words" F1 scores divided by "fastText 1 word" F1 scores.

(b) "PyRv+FT 5 words" F1 scores divided by "fastText 1 word" F1 scores.

Figure 5.7: DEPREL relative F1 score ratios (by class) comparing different composition methods. The left plot (a) shows the ratio of F1 scores for "mean fastText 5 words" versus "fastText 1 word", while the right plot (b) compares "PyRv+FT 5 words" versus "fastText 1 word". Each bar represents a class, and the length of the bar indicates the relative performance of the model. Classes are ordered by support value in the test set (larger on the top).

single word fastText embedding). Weighted averaging yields an F1 score of 0.28, a significant improvement over using the target word alone (0.03), though slightly below the performance of uniform averaging (0.32). This suggests that while Gaussian weighting helps integrate context, too narrow a focus may exclude useful syntactic signals necessary for accurate conj classification. Composing these embeddings using PyRv further boosts performance, reaching an F1 score of 0.64 (a 21.85x improvement over the single word embedding).

Adnominal clause (acl) refers to finite or non-finite clauses that modify a nominal. For instance, in "the *issues* as he *sees* them," the word "sees" is labeled as acl (connected to "issues"). In "There are many online *sites offering* booking facilities," the word "offering" is labeled as acl (connected to "sites"). Using a single fastText word embedding results in an F1 score of 0.14. Averaging the target word's embedding with its four neighbors actually degrades performance slightly, reducing the F1 score to 0.11. Weighted averaging yields an F1 score of 0.16, outperforming both single word embeddings (0.14) and uniform averaging (0.11). This indicates that Gaussian-weighted context can capture some of the syntactic dependencies without overwhelming the representation with irrelevant context. Composing these word embeddings with PyRv substantially improves performance, raising the F1 score to 0.53 (a 3.84x improvement over the single word embedding).

Qualitative analysis. To gain qualitative insights into the structure of PyRv+FT embeddings, we visualize the representation space. A portion of this space is shown in Figures 5.8 and 5.9. This qualitative exploration aims to complement the quantitative results by offering an intuitive understanding of whether PyRv+FT learns to group semantically or structurally similar phrases. In these visualizations, each node represents a phrase consisting of two or three words. To illustrate the hierarchical relationship captured by PyRv's composition, a two-word phrase is connected to a three-word phrase if the shorter phrase is part of the longer one.

Tables 5.5 and 5.6 (with original Croatian phrases in Tables 5.7 and 5.8) provide concrete examples of phrases found within specific highlighted clusters (Areas A, B, C, D) from the visualizations. While the figures offer a global view of the embedding space structure, these tables allow for a detailed examination of the phrases that cluster together.

The areas circled in the figures contain phrases built around the prepositions "u" (Croatian for "in") and "na" (Croatian for "on"). For example, Area C contains two-word phrases like "primjena **na**" (eng. "application on"), while Area A includes phrases such as "**na** svijet" (eng. "on the world"). Similarly, Area B features three-word phrases like "primjena **na** svijet" (eng. "application on the world"), and Area D includes phrases like "**na** svijet oko" (eng. "on the world around"). Same holds for the preposition "u".

The organization of phrases in the representation space is not random: phrases with similar syntactic structures (e.g., where the preposition appears at the beginning, middle, or end of the phrase) tend to cluster together. Furthermore, within these broader areas, smaller sub-clusters form based on the specific preposition ("u" or "na") present in the phrase. This suggests that the PyRv+FT composition method captures and reflects both structural and lexical similarities in the resulting phrase embeddings.

5.2.3 Summary and Conclusions

In this section, we explored the use of PyRv for the composition of word embeddings and evaluated its ability to generate meaningful representations of multi-word units. Our findings show that PyRv outperforms simple averaging methods in embedding composition.

In the part-of-speech tagging task, where word context is less crucial, single word embeddings achieve an accuracy of 95%. Averaging 3-word context embeddings reduces this to 61% due to the loss of word order information, while PyRv retains a high accuracy of 93% by effectively preserving word order.

In the more complex task of dependency relation labeling, where single word embeddings reach 71% accuracy, averaging embeddings for 5-word contexts results in a sharp decline to 34%. In contrast, composing context words with PyRv attains a significantly higher accuracy of 77%, demonstrating its superior capability in integrating multiple word embeddings into cohesive and expressive representations. This validates the effectiveness of the compositionality

Table 5.5: Phrases by areas (A, B, and C) in the PyRv+FT representation space, translated to English (some phrases are longer when translated).

Preposition "on" (cro. "na")		
Area C	**Area B**	**Area A**
application on	application on the world	on the world
are on	are on local	on local
assistant on	assistant on the subject	on the subject
media on	media on protest	on the protest
vat on	vat on tickets	on tickets
based on	based on data	on data
finance on	finance on revenues	on revenues
os on	os on which	on which
dollars on	dollars on google	on google
found on	found on the third	on the third
relation on	relation on the past	on the past
Preposition "in" (cro. "u")		
Area C	**Area B**	**Area A**
enthusiast in	enthusiast in the river	in the river
currently in	currently in testing	in testing
released in	released in circulation	in circulation
circulation in	circulation in June	in June
by activity in	by activity in teaching	in teaching
work in	work in the wood	in the wood
tickets in	tickets in europe	in europe
musician in	musician in croatia	in croatia
only in	only in the past	in the past
drop in	drop in the sea	in the sea
is in	is in the past	in the past
year in	year in croatia	in croatia
percent in	percent in relation	in relation
and in	and in the average	in the average
. in	. in this	in this

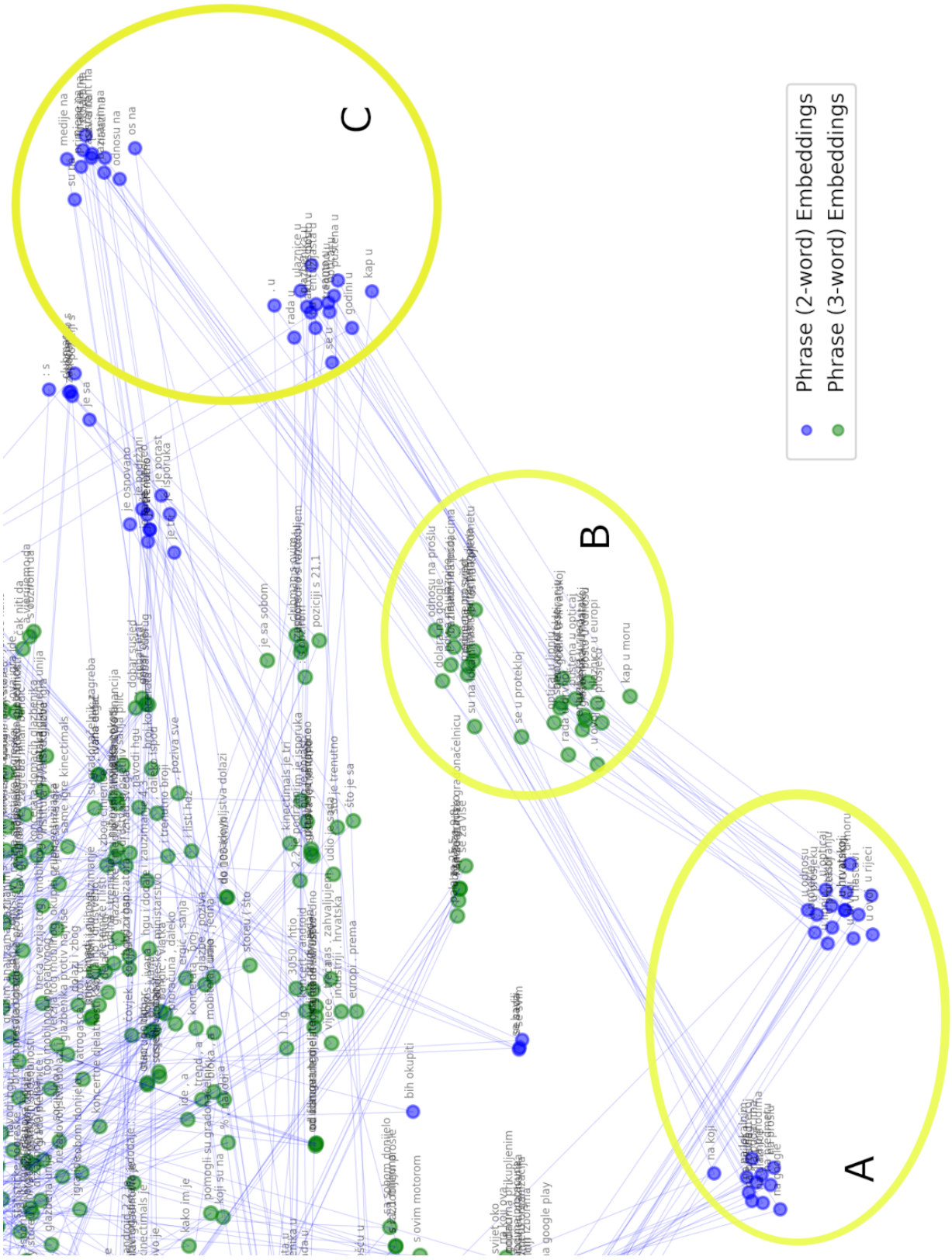


Figure 5.8: Visualization of the PyRv+FT representation space (reduced from 300 dimensions using t-SNE). Highlighted areas A, B, and C contain phrases with the prepositions 'na' (eng. 'on') and 'u' (eng. 'in'). Tables 5.7 and 5.8 (translated: 5.5 and 5.6) provide detailed examples of these phrases and their connections within the space.

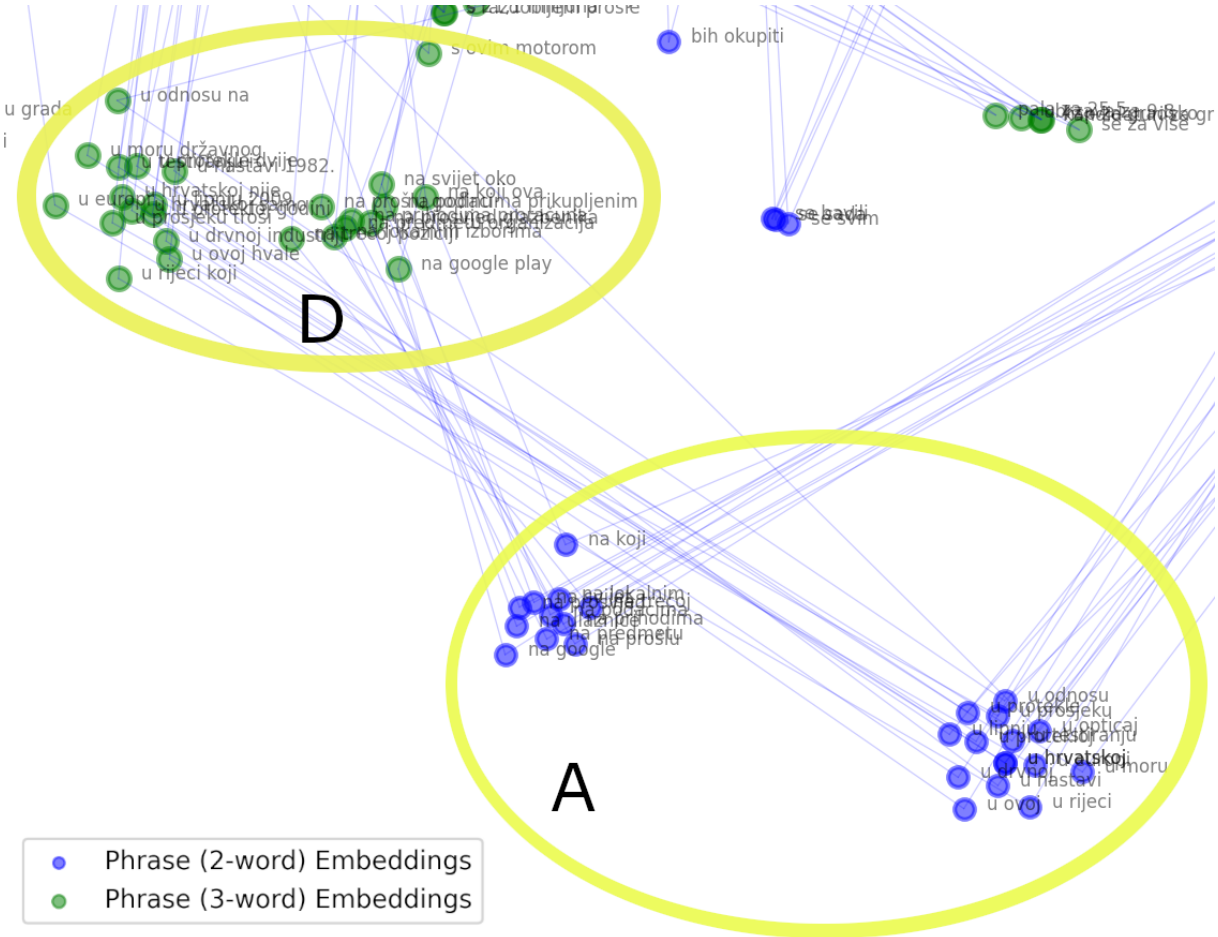


Figure 5.9: Visualization of the PyRv+FT representation space (reduced from 300 dimensions using t-SNE). Highlighted areas A and D contain phrases structured around the prepositions 'na' (eng. 'on') and 'u' (eng. 'in'). Tables 5.7 and 5.8 (translated: 5.5 and 5.6) present examples of these phrases and their relationships in the embedding space.

Table 5.6: Phrases in area D in the PyRv+FT representation space, translated to English (some phrases are longer when translated).

Area D	
Preposition "on" (cro. "na")	**Preposition "in" (cro. "u")**
on the world around	in the river which
on local elections	in testing and
on the subject of organization	in circulation in
on the protest of musicians	in june 2009
on tickets for	in teaching 1982
on the data collected	in the wood industry
on budget revenues	in europe .
on which this	in croatia only
on google play	in the past two
on the third position	in the sea of state
on the past year	in the past year
	in croatia is not
	in relation to
	in the average spends
	in this praise

property of PyRv in real-world tasks.

Table 5.7: Phrases by areas (A, B, and C) in the PyRv+FT representation space (original Croatian phrases).

Preposition "na" (eng. "on")		
Area C	**Area B**	**Area A**
primjena na	primjena na svijet	na svijet
su na	su na lokalnim	na lokalnim
asistentent na	asistentent na predmetu	na predmetu
medije na	medije na prosvjed	na prosvjed
pdv-a na	pdv-a na ulaznice	na ulaznice
baziranim na	baziranim na podacima	na podacima
financija na	financija na prihodima	na prihodima
os na	os na koji	na koji
dolara na	dolara na google	na google
nalazi na	nalazi na trećoj	na trećoj
odnosu na	odnosu na prošlu	na prošlu
Preposition "u" (eng. "in")		
Area C	**Area B**	**Area A**
entuzijasta u	entuzijasta u rijeci	u rijeci
trenutno u	trenutno u testiranju	u testiranju
puštena u	puštena u opticaj	u opticaj
opticaj u	opticaj u lipnju	u lipnju
aktivnošću u	aktivnošću u nastavi	u nastavi
rada u	rada u drvnoj	u drvnoj
ulaznice u	ulaznice u europi	u europi
glazbenika u	glazbenika u hrvatskoj	u hrvatskoj
samo u	samo u protekle	u protekle
kap u	kap u moru	u moru
se u	se u protekloj	u protekloj
godini u	godini u hrvatskoj	u hrvatskoj
posto u	posto u odnosu	u odnosu
i u	i u prosjeku	u prosjeku
. u	. u ovoj	u ovoj

Table 5.8: Phrases in area D in the PyRv+FT representation space (original Croatian phrases).

Area D	
Preposition "na" (eng. "on")	**Preposition "u" (eng. "in")**
na svijet oko	u rijeci koji
na lokalnim izborima	u testiranju i
na predmetu organizacije	u opticaj u
na prosvjed glazbanika	u lipnju 2009.
na ulaznice u	u nastavi 1982.
na podacima prikupljenim	u drvnoj industriji
na prihodima proračuna	u europi .
na koji ova	u hrvatskoj samo
na google play	u protekle dvije
na trećoj poziciji	u moru državnog
na prošlu godinu	u protekloj godini
	u hrvatskoj nije
	u odnosu na
	u prosjeku troši
	u ovoj hvale

5.3 Memorization Task

In the memorization experiment, we train a simple language model that memorizes a small dataset. We use a recurrent neural network with fully connected layers as the first and the last layers, and four long short-term memory (LSTM) layers in between. The first fully connected layer has 2000 units, the four LSTM layers subsequently have 2000, 1000, 500, 500, 1000, and 2000 units respectively, and the final fully connected layer has 2000 units. This simple language model is trained on 500 random Wikipedia paragraphs to predict the next token and effectively memorize the entire dataset. The tokens are embedded using PyRvNN, and for comparison, fastText is chosen as a baseline. This comparison is intended to evaluate PyRvNN's quality as a method for generating non-contextual, single-word (or subword-unit) embeddings, a domain where fastText is a well-established and relevant benchmark. The goal is not to compare PyRvNN as a generative language model against state-of-the-art Large Language Models, but rather to assess the utility of its word representations in a sequential task.

5.3.1 Results

The original fastText [2] embeddings with 300 dimensions performed very poorly. The language model trained with fastText embeddings could barely memorize two paragraphs. To have a leveled ground we trained fastText with 2000 dimensional vectors (same as PyRvNN) on the same dataset and the same number of epochs as PyRvNN was trained on, and will refer to it as "FastText2k".

Tables 5.9 and 5.10 show the results of that experiment after 260 epochs and 400 epochs

of LSTM language model training. Each language model generated the first 100 paragraphs (from 500 memorized in total). The generated paragraphs are generated token by token with all the previous tokens from the original paragraph as input. The two measures used are: the percentage of words in the generated paragraph that are identical as in the original paragraph and the average Levenshtein distance between the generated words and the original words. The first 100 paragraphs have 7693 words and 34253 characters in total.

Table 5.9: Memorization results, 260 epochs.

Embedding model	% of identical words	Avg Levenshtein
PyRvNN	71.94	1.35
FastText2k	59.05	3.12

Table 5.10: Memorization results, 400 epochs.

Embedding model	% of identical words	Avg Levenshtein
PyRvNN	95.09	0.45
FastText2k	74.34	1.71

5.4 Plagiarism Task

Another task that can be used for the evaluation of PyRvNN representations is machine-paraphrased plagiarism classification. Wahle et al. [144] constructed a dataset for such a task with 200,767 paragraphs (50% of which are paraphrased using the SpinBot API) extracted from Wikipedia (English) articles. We trained our models on the training set defined by Wahle et al. and evaluated them on their corresponding test set. They evaluate some embedding models (GloVe, Word2Vec, Doc2Vec, and FastText) on that task by training classical machine learning algorithms (Naive Bayes, Logistic Regression, and Support Vector Machine). Crucially, for word embedding models like GloVe and FastText, each paragraph in their study is represented as an average of its constituent word embeddings. To ensure a fair and direct comparison within this specific evaluation paradigm, we adopt the same approach for PyRvNN: PyRvNN is used here to generate embeddings for individual words (composed from subwords), which are then averaged to form paragraph representations. While Wahle et al. also report results for models that embed entire texts contextually (e.g., Doc2Vec, and potentially BERT in broader contexts), our comparison focuses on the word-level embedding averaging setup they established for models like GloVe and FastText, against which PyRvNN's single-word representations are benchmarked. The best-performing machine learning algorithm for every embedding model in their

work is support vector machine (SVM), so we use SVM for comparisons. Specifically, we used an SVM with a linear kernel, a regularization parameter C set to 100, and all other parameters set to their scikit-learn defaults (e.g., degree=3, gamma='scale').

5.4.1 Results

Wahle's et al. evaluations on the machine-paraphrased plagiarism classification task with SVM are as follows (F1 score): GloVe 89.55, Word2Vec 87.27, Doc2Vec 83.04, FastText-rw (without subwords) 86.15, FastText-sw (with subwords) 82.57. PyRvNN is evaluated with F1 of **89**, which is better than all evaluated embeddings but one (GloVe) and is better than the only other evaluated subword embedding model (FastText-sw). All the results are in the Table 5.11.

Table 5.11: Machine-paraphrased plagiarism classification results.

Embedding model	**F1 score**
GloVe	89.55
Word2Vec	87.27
Doc2Vec	83.04
FastText-rw (without subword)	86.15
FastText-sw (with subwords)	82.57
PyRvNN	89

It is worth mentioning that embedding models evaluated by Wahle et al. were trained on much bigger datasets than PyRvNN. The PyRvNN model evaluated here was trained on only ∼300,000 paragraphs from Wikipedia (one epoch).

PyRvNN embeddings, with a dimensionality of 2000 compared to ∼300 in other embeddings, prompted an investigation into dimensionality significance. To level the ground, we compare FastText2k embeddings and PyRvNN embeddings by training an SVM on 10% of the plagiarism dataset (due to the extended SVM training time with FastText2k embeddings). SVM with FastText2k resulted in an F1 score of 79. In contrast, the SVM with PyRvNN embeddings achieved a higher F1 score of 87. This suggests that, in this case, dimensionality may not be decisive, leaving the role of dimensionality in this context as an open research question.

5.5 Readability Task

In our readability experiment, we assess text embeddings for readability classification using the CommonLit Ease of Readability (CLEAR) dataset [148]. It contains approximately 5000 texts, each accompanied by readability scores derived from teachers' evaluations of text complexity

for student readers. We normalize readability scores from the CLEAR dataset to the values between 0 and 1. Texts with scores below 0.45 are labeled as less readable (1812 texts) and texts with scores above 0.55 as more readable (2030 texts). Texts with scores between 0.45 and 0.55 were excluded to create a clearer distinction between the two classes for this binary classification task. Figure 5.10 shows the density plot of scores before this exclusion.

We employ PyRvNN, BERT (base model, uncased), fastText, and Term Frequency-Inverse Document Frequency (TFIDF) embeddings. To evaluate the inherent separability of these embedding spaces with respect to readability, we train a Gaussian Naive Bayes classifier. Gaussian Naive Bayes is chosen for its simplicity and efficiency, providing a baseline indication of how well the different embedding methods create feature spaces where the two readability classes can be distinguished by a relatively straightforward probabilistic model. The model was trained on 3000 texts and tested on a separate set of 800 texts from the binarized dataset.

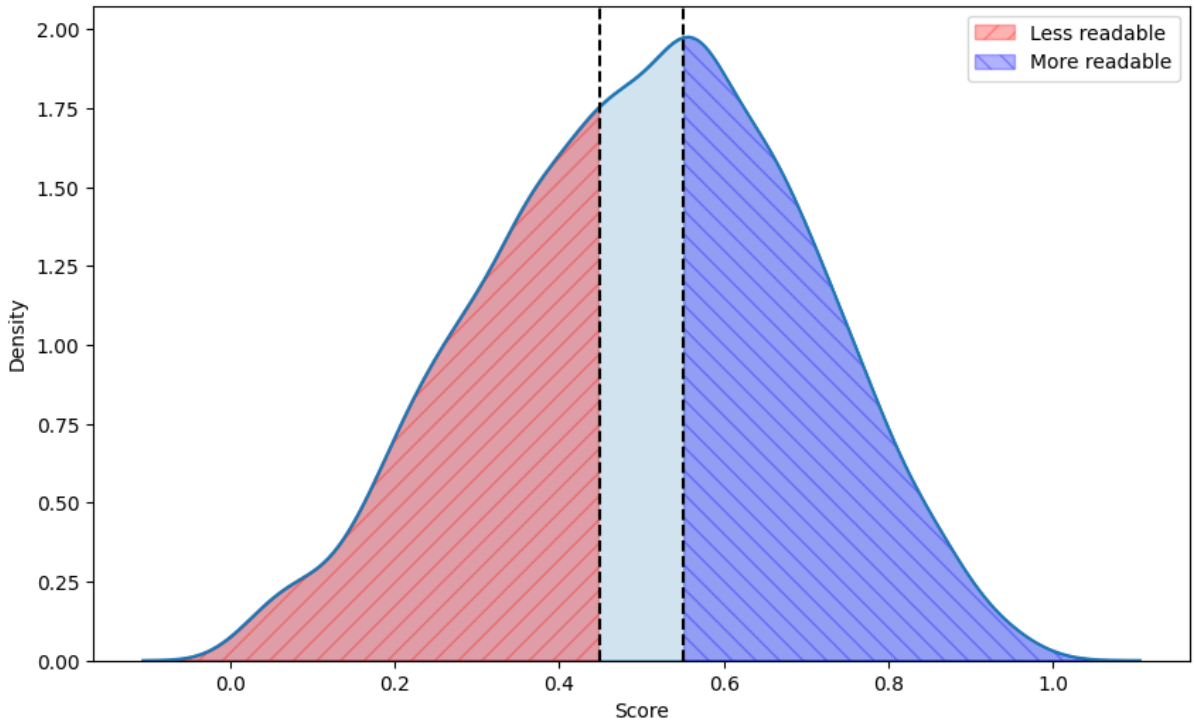


Figure 5.10: Density plot of scores.

5.5.1 Results

To assess the separability of the PyRvNN representation space for the task of readability classification, we employ Gaussian Naive Bayes to classify embedded texts as either less or more readable. Additionally, we train and evaluate the same model using BERT, FastText, and TFIDF embeddings for comparison. Table 5.12 presents the accuracies achieved on both the training and test sets using the embeddings mentioned.

The results in Table 5.12 indicate that BERT embeddings provide the most separable space

Table 5.12: Readability classification results.

Embedding model	Train Acc	Test Acc
BERT	0.85	0.84
PyRvNN	0.74	0.73
FastText	0.72	0.73
TFIDF	1.00	0.70

for this Gaussian Naive Bayes classifier, achieving the highest test accuracy of 0.84. This suggests its contextual representations effectively capture features related to text readability. PyRvNN and FastText embeddings yield comparable test accuracies (0.73), performing noticeably better than TFIDF (0.70) on the test set, despite TFIDF achieving perfect accuracy on the training set. The perfect training accuracy for TFIDF, coupled with its lower test accuracy, suggests overfitting, which is a common characteristic when using high-dimensional sparse features like TFIDF with simpler classifiers [149, Chapter 3]. PyRvNN's performance, on par with FastText and substantially better than TFIDF on unseen data, demonstrates that its learned representations capture meaningful signals related to text readability.

Chapter 6

Discussion

This chapter discusses the thesis's main findings. It first highlights the work's main scientific contributions and novel aspects in comparison to existing research. The discussion then proceeds to an evaluation of the hypotheses, detailing how they were tested and validated through extensive experimentation and analysis.

While Large Language Models (LLMs) show significant capabilities, this thesis investigates an alternative approach through dedicated text embedding models. The PyRv method, developed in this work, aims to combine several key characteristics: (1) meaningful compositionality from sub-components; (2) hierarchical encoding across different granularities; (3) decodability of representations to their original text for better understanding; and (4) self-supervised learning using unlabeled data, without needing external tools like parsers. To our knowledge, achieving all these features together in one model is not common. For example, obtaining robust compositionality with an explicit, multi-level decodable hierarchy via purely self-supervised methods is an ongoing research challenge, which motivated the development presented in this thesis.

6.1 Contributions

This section highlights the main scientific contributions of this thesis, which focus on the development and evaluation of a method for learning structured text representations. The work introduces a novel method for learning text representations, evaluates its performance, and demonstrates its applicability to morphologically rich languages. These contributions are detailed as follows.

Contribution 1: A novel method for learning text representations with pyramid-structured recursive neural networks. This thesis introduces Pyramidal Recursive learning (PyRv), a method for learning text representations using recursive neural networks. PyRv addresses certain limitations of existing techniques by effectively capturing the hierarchical and composi-

tional nature of language in a decodable and unsupervised manner. Unlike traditional models such as one-hot vectors, bag-of-words, TF-IDF, and even word embedding techniques like Word2Vec [1], FastText [2], and GloVe [3], PyRv explicitly models hierarchical relations. Transformer-based models like BERT [4] and GPT [5] (and subsequent LLMs) provide dynamic representations but lack both decodability and hierarchical compositionality. In contrast, PyRv utilizes a pyramidal recursive architecture to construct hierarchical representations from subwords to sentences, ensuring a structured composition of learned features. Moreover, its representations are inherently decodable in a self-supervised manner, enhancing both interpretability and utility.

PyRv's methodology diverges from many recursive models in its construction of an explicit, multi-level decodable hierarchy without reliance on linguistic parse trees. Traditional recursive networks like MV-RNN [77] and RNTN [25], as well as more recent approaches such as Transformer Grammars [86] and the model by Simoulin and Crabbé [85], typically require or operate over predefined syntactic structures. While other self-supervised recursive models including DIORA [83], CRvNN [84], Fast-R2D2 [87], and ReCAT [88] can induce latent tree structures, PyRv instead forms its hierarchy through a deterministic, pyramidal pairwise merging process. This fixed structure, combined with its autoencoding objective at each level, facilitates direct multi-level decodability. Furthermore, unlike supervised non-parsing hierarchical models like AdaSent [79], PyRv is entirely self-supervised.

Contribution 2: Extensive evaluation of the proposed method in terms of training stability, quality of learned embeddings, and predictive performance on downstream tasks. The thesis provides an extensive evaluation of PyRv's ability to learn hierarchical, compositional, and decodable representations without relying on labeled data or predefined parsing trees.

The evaluation covers intrinsic tasks, such as representation decodability and compositionality, and extrinsic NLP tasks, including memorization, plagiarism detection, and readability assessment. These experiments validate the proposed mechanisms and demonstrate its real-world applicability, highlighting its effectiveness in capturing linguistic properties. The detailed results and analysis of these evaluations are further discussed in the context of the hypotheses presented in the next section.

PyRv improves upon simple embedding composition techniques, such as averaging (as used by Joulin et al. [129] and Arora et al. [130]), which fail to capture word order and syntactic structures. Instead, PyRv recursively combines lower-level representations into meaningful higher-level ones, ensuring both structural and semantic integrity. Unlike some other recursive autoencoders (e.g., RAE [76] and DIORA [83]), PyRv incorporates both autoencoding and autoregressive objectives without relying on parsing trees, ensuring that the learned representations retain both decodability and contextual coherence without propagating possible parsing tree errors and biases.

Contribution 3: Improved pre-trained representations for morphologically rich languages (e.g. Croatian) in terms of less training material when adapting to new domains. The research demonstrates PyRv's ability to enhance pre-trained representations for morphologically rich languages, such as Croatian, where linguistic complexity poses challenges for traditional models. Many NLP models are primarily optimized for English and struggle to generalize effectively to languages with complex morphology.

PyRv's approach supports effective application to new tasks using composed word embeddings. This is demonstrated in the Representation Compositionality experiments (5.2). In these experiments, PyRvNN was trained on a general Croatian Wikipedia corpus, using Croatian fastText word embeddings as input (this combination is termed PyRv+FT). This training allowed PyRvNN to learn how to compose these fastText embeddings into multi-word representations. Subsequently, these PyRv+FT composed embeddings were used to train a simple MLP model for Croatian UPOS tagging and DEPREL labeling, where they demonstrated superior performance. This indicates that PyRvNN, after learning a general composition function from a corpus like Wikipedia, can effectively create multi-word representations from standard word embeddings for downstream tasks, without needing task-specific retraining of the core PyRvNN model itself.

6.2 Hypotheses

The research presented in this thesis was guided by the following hypotheses, which were evaluated through experiments and analyses.

H1: The training of pyramidal recursive networks for text representation can be stable and efficient. The training procedure, outlined in Section 4.3, demonstrated stable convergence across various pyramid levels due to the application of loss normalization. As detailed in Subsection 4.3.3, loss normalization was designed to address the cascading effect of errors between pyramid levels. Specifically, the normalization factor (NF) was made inversely proportional to the total loss of all lower pyramid levels (LB). By stabilizing the losses across levels, this approach ensured that low-quality representations generated at one level did not disproportionately degrade the performance of subsequent levels. The effectiveness of this stable learning process is directly evidenced by the model's performance in the Decoding Accuracy experiment (Section 5.1). PyRvNN achieved high accuracy in decoding representations from various pyramid levels back into their original textual components (achieving over 99% decoding accuracy for words up to 6 tokens). This strong decoding performance indicates that the model successfully learned to form reconstructible representations at each stage of its recursive, hierarchical encoding process. Such consistent decodability across levels serves as a direct measure of both training stability and the model's ability to learn effectively. The results presented in Chapter 5 support the claim that PyRvNN's training procedure achieves both stability and efficiency in

learning these structured representations.

H2: Pyramidal recursive training can simultaneously generate text representations for different levels of linguistic units. The results of the Decoding Accuracy experiments in Section 5.1 strongly support this hypothesis as well. PyRvNN successfully generated representations for subwords, words, and phrases. While accuracy declined for longer sequences, the model effectively captured hierarchical linguistic structures for shorter sequences.

Further validation came from the Representation Compositionality experiments in Section 5.2. In these experiments, PyRvNN demonstrated its ability to generate representations at different levels of linguistic hierarchy by composing varying numbers of word embeddings. For instance, in the UPOS tagging task, PyRvNN composed three fastText word embeddings to create a phrase-level representation, maintaining 93% accuracy, compared to 61% for simple averaging of the same context embeddings. For the more complex DEPREL task, PyRvNN composed five fastText word embeddings, forming an even higher-level phrase representation, and achieved 77% accuracy, again outperforming the averaging approach's 34%. This process of combining different numbers of word embeddings (e.g., three or five) corresponds to generating representations at different nodes or levels within the PyRvNN pyramidal structure. The qualitative analysis of PyRv+FT embeddings further illustrated this by showing how distinct representations for two-word phrases and three-word phrases were formed and how these representations clustered based on syntactic similarities, effectively demonstrating the generation of text representations for different levels of linguistic units (phrases of varying lengths).

H3: The proposed method has better performance compared to other competing methods on selected tasks and domains. This hypothesis was supported by multiple extrinsic evaluations presented in Chapter 5. In the Plagiarism Detection Task (Section 5.4), PyRvNN achieved an F1 score of 89, surpassing Word2Vec (87.27), Doc2Vec (83.04), and FastText (86.15), though slightly underperforming compared to GloVe (89.55). In the Readability Classification Task (Section 5.5), PyRvNN demonstrated competitive performance with a test accuracy of 73%, matching fastText and outperforming TFIDF (70%), though falling short of BERT's 84%. The Memorization Task (Section 5.3) further highlighted the model's robustness, with PyRvNN achieving 95.09% identical words, significantly outperforming fastText2k (74.34%). The Representation Compositionality experiments further supported this hypothesis, with PyRvNN outperforming FastText in DEPREL and maintaining high performance for UPOS tagging when compared to context-averaging techniques. These results collectively demonstrate PyRvNN's strengths across diverse tasks, despite being trained on significantly smaller datasets in comparison to some competing methods, indicating strong potential for further improvements with larger-scale training.

Chapter 7

Conclusion

This doctoral thesis has explored the intersection of natural language processing (NLP), text representation learning, and recursive neural networks (RvNNs), culminating in the development of a novel method called Pyramidal Recursive learning (PyRv). The research addresses core challenges in representation learning by introducing a self-supervised approach that effectively captures the hierarchical and compositional nature of language. In a landscape where large language models (LLMs) have demonstrated significant capabilities, this work focuses on an alternative paradigm emphasizing structured, hierarchical representations through recursive mechanisms.

The core component of this research is the Pyramidal Recursive Neural Network (PyRvNN), an architecture trained using the Pyramidal Recursive learning (PyRv) method. PyRvNN is designed to learn multilevel text representations for subwords, words, and phrases within a pyramidal structure. Unlike traditional RvNNs, which rely on explicit parse trees, PyRvNN leverages a pyramidal approach to enable hierarchical representation learning without requiring labeled data or explicit parsing. By combining recursive autoencoding and autoregressive objectives, the model ensures that the learned representations are both decodable and capable of capturing sequential dependencies.

This thesis has demonstrated PyRvNN's strengths across several dimensions:

- Decodability and Compositionality: PyRvNN achieves high decoding accuracy, reconstructing text from its learned representations at various levels. It also exhibits effective compositionality, combining lower-level embeddings (e.g., word embeddings) into meaningful higher-level representations, as evidenced by its performance on universal part-of-speech (UPOS) tagging and universal dependency relation (DEPREL) labeling tasks.
- Self-Supervised Learning: The model's reliance on self-supervised learning objectives eliminates the need for labeled data, making it versatile and adaptable across different domains and languages.
- Promising Downstream Performance: PyRvNN demonstrates potential in downstream applications, with notable performance in tasks like memorization, plagiarism detection,

and readability assessment.

In summary, the main contributions are: (1) a novel method for learning text representations with pyramid-structured recursive neural networks; (2) extensive evaluation of the proposed method in terms of training stability, quality of learned embeddings, and predictive performance on downstream tasks; (3) improved pre-trained representations for morphologically rich languages (e.g. Croatian) in terms of less training material when adapting to new domains.

The hypotheses proposed in this study are as follows: (H1) the training of pyramidal recursive networks for text representation can be stable and efficient; (H2) pyramidal recursive training can simultaneously generate text representations for different levels of linguistic units; (H3) the proposed method has better performance compared to other competing methods on selected tasks and domains.

While PyRvNN demonstrates strong potential in text representation learning, several limitations identified during its development and evaluation highlight opportunities for future research. Below, we outline directions to improve and expand upon the current work.

Exploring Alternative Neural Network Architectures. Future research could investigate alternative neural network architectures within the PyRv framework, such as integrating hierarchical attention mechanisms or dynamic weighting strategies. These additions may help address challenges such as diminishing decoding accuracy at higher recursion depths while maintaining computational efficiency.

Training Data Diversity. The primary pre-training of PyRvNN was conducted on a single domain (Wikipedia). This may limit its performance on text from different domains. Future work should involve pre-training PyRvNN on more diverse corpora to improve its robustness and generalization.

Application to a Broader Range of Downstream Tasks, Datasets, and Baselines. The current evaluation of PyRvNN focuses on a select set of NLP tasks and datasets. Expanding its application to a wider variety of downstream tasks, such as machine translation, sentiment analysis, and question answering, as well as testing it on diverse datasets, including general text embedding benchmarks, could further establish its generalizability and practical utility. Future work should also include rigorous comparisons of PyRvNN against current state-of-the-art text embedding methods on these benchmarks to better position its performance among general text embedding approaches.

Developing Generative Capabilities and Perplexity Metric. The current PyRvNN is not designed for text generation, making perplexity an unsuitable metric for its evaluation. Future work could involve developing a generative variant of PyRvNN. For such a model, perplexity

would be an appropriate evaluation metric, allowing comparison with other generative language models.

Investigating the Impact of Tokenization Methods. Different tokenization methods can significantly influence the quality of learned representations. Exploring the effects of various tokenization strategies, such as subword units, byte-pair encoding, or character-level models, could provide insights into optimizing PyRvNN's performance, particularly for morphologically rich languages.

Addressing Memory Constraints and Batch Size Limitations. The hierarchical structure of PyRvNN imposes high memory demands, restricting batch sizes to a single paragraph during training. Employing techniques such as gradient checkpointing or model parallelism could reduce memory usage, enabling larger batch sizes and improving training efficiency and stability.

Enhancing Training Stability and Efficiency. While multi-level loss aggregation and gradient normalization stabilize PyRv's training, further exploration of advanced optimization techniques (such as gradient clipping, alternative activation functions, and adaptive learning rate schedules) could enhance convergence rates and mitigate vanishing or exploding gradient issues inherent in recursive architectures.

Improving Computational Efficiency. The recursive and hierarchical nature of PyRvNN increases computational complexity, especially when generating input-output pairs for autoencoder and autoregression heads at each pyramid level. Streamlining this process through more efficient computation methods or architectural modifications could reduce training times without sacrificing performance.

Handling Long-Range Dependencies. Although PyRvNN captures hierarchical relationships effectively, it struggles with long-range dependencies, a limitation common to recursive models. Incorporating attention mechanisms into PyRvNN could enhance its ability to model distant relationships in text, improving its capacity to process longer sequences and capture more nuanced linguistic structures.

Bibliography

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[2] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[3] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, 2014, pp. 1532–1543.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[5] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

[6] J. Wei, M. Bosma, V. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=gEZrGCozdqR

[7] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.

[8] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[9] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[10] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "PaLM: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[11] T. Zhong, Z. Liu, Y. Pan, Y. Zhang, Y. Zhou, S. Liang, Z. Wu, Y. Lyu, P. Shu, X. Yu *et al.*, "Evaluation of OpenAI O1: Opportunities and challenges of AGI," *arXiv preprint arXiv:2409.18486*, 2024.

[12] K. Babić, S. Martinčić-Ipšić, and A. Meštrović, "Survey of neural text representation models," *Information*, vol. 11, no. 11, p. 511, 2020.

[13] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[14] K. Jing, J. Xu, and B. He, "A survey on neural network language models," *arXiv preprint arXiv:1906.03591*, 2019.

[15] J. Camacho-Collados and M. T. Pilehvar, "From word to sense embeddings: A survey on vector representations of meaning," *Journal of Artificial Intelligence Research*, vol. 63, pp. 743–788, 2018.

[16] S. Ruder, I. Vulić, and A. Søgaard, "A survey of cross-lingual word embedding models," *Journal of Artificial Intelligence Research*, vol. 65, pp. 569–631, 2019.

[17] M. Aßenmacher and C. Heumann, "On the comparability of pre-trained language models," *arXiv preprint arXiv:2001.00781*, 2020.

[18] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin, "Placing search in context: The concept revisited," in *Proceedings of the 10th International Conference on World Wide Web*, 2001, pp. 406–414.

[19] M. Marelli, S. Menini, M. Baroni, L. Bentivogli, R. Bernardi, R. Zamparelli *et al.*, "A SICK cure for the evaluation of compositional distributional semantic models," in *LREC*, 2014, pp. 216–223.

[20] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning, "A large annotated corpus for learning natural language inference," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.

[21] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, 2018, pp. 1112–1122. [Online]. Available: http://aclweb.org/anthology/N18-1101

[22] B. Dolan, C. Quirk, and C. Brockett, "Unsupervised construction of large paraphrase

corpora: Exploiting massively parallel news sources," in *Proceedings of the 20th International Conference on Computational Linguistics*. Association for Computational Linguistics, 2004, p. 350.

[23] E. Agirre, C. Banea, C. Cardie, D. Cer, M. Diab, A. Gonzalez-Agirre, W. Guo, R. Mihalcea, G. Rigau, and J. Wiebe, "SemEval-2014 task 10: Multilingual semantic textual similarity," in *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, 2014, pp. 81–91.

[24] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013, pp. 746–751.

[25] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment Treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA, 2013, pp. 1631–1642.

[26] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. Association for Computational Linguistics, 2011, pp. 142–150.

[27] B. Pang and L. Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2005, pp. 115–124.

[28] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 168–177.

[29] S. Rosenthal, N. Farra, and P. Nakov, "SemEval-2017 task 4: Sentiment analysis in Twitter," in *Proceedings of the 11th International Workshop on Semantic Evaluation*, ser. SemEval '17. Vancouver, Canada: Association for Computational Linguistics, August 2017.

[30] E. M. Voorhees and D. Harman, "Overview of TREC 2002," in *TREC*, 2002.

[31] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, J. Su, K. Duh, and X. Carreras, Eds. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2383–2392. [Online]. Available: https://aclanthology.org/D16-1264/

[32] J. Wiebe, T. Wilson, and C. Cardie, "Annotating expressions of opinions and emotions in

language," *Language Resources and Evaluation*, vol. 39, no. 2-3, pp. 165–210, 2005.

[33] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee *et al.*, "Natural Questions: A benchmark for question answering research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 453–466, 2019.

[34] J. Berant, A. Chou, R. Frostig, and P. Liang, "Semantic parsing on Freebase from question-answer pairs," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013, pp. 1533–1544.

[35] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[36] J. Wang, Z. Wang, D. Zhang, and J. Yan, "Combining knowledge with deep convolutional neural networks for short text classification," in *IJCAI*, 2017, pp. 2915–2921.

[37] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, T. Linzen, G. Chrupała, and A. Alishahi, Eds. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 353–355. [Online]. Available: https://aclanthology.org/W18-5446/

[38] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy, "RACE: Large-scale reading comprehension dataset from examinations," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, 9 2017, pp. 785–794. [Online]. Available: https://www.aclweb.org/anthology/D17-1082

[39] B. Pang and L. Lee, "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts," in *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2004, p. 271.

[40] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, R. Soricut, L. Specia, and A. s. Tamchyna, "Findings of the 2014 workshop on statistical machine translation," in *Proceedings of the Ninth Workshop on Statistical Machine Translation*. Baltimore, Maryland, USA: Association for Computational Linguistics, June 2014, pp. 12–58. [Online]. Available: http://www.aclweb.org/anthology/W/W14/W14-3302

[41] O. r. Bojar, R. Chatterjee, C. Federmann, B. Haddow, M. Huck, C. Hokamp, P. Koehn, V. Logacheva, C. Monz, M. Negri, M. Post, C. Scarton, L. Specia, and M. Turchi, "Findings of the 2015 workshop on statistical machine translation,"

in *Proceedings of the Tenth Workshop on Statistical Machine Translation*. Lisbon, Portugal: Association for Computational Linguistics, September 2015, pp. 1–46. [Online]. Available: http://aclweb.org/anthology/W15-3001

[42] E. Charniak, D. Blaheta, N. Ge, K. Hall, J. Hale, and M. Johnson, "BLLIP 1987-89 WSJ corpus release 1," *(No Title)*, 2000.

[43] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "SuperGLUE: A stickier benchmark for general-purpose language understanding systems," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[44] A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, and A. G.-A. at al., "Beyond the imitation game: Quantifying and extrapolating the capabilities of language models," *Transactions on Machine Learning Research*, 2023, featured Certification. [Online]. Available: https://openreview.net/forum?id=uyTL5Bvosj

[45] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "MTEB: Massive text embedding benchmark," in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, A. Vlachos and I. Augenstein, Eds. Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 2014–2037. [Online]. Available: https://aclanthology.org/2023.eacl-main.148/

[46] L. Svoboda and S. Beliga, "Evaluation of Croatian word embeddings," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, N. Calzolari, K. Choukri, C. Cieri, T. Declerck, S. Goggi, K. Hasida, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, S. Piperidis, and T. Tokunaga, Eds. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. [Online]. Available: https://aclanthology.org/L18-1240/

[47] J. Botha and P. Blunsom, "Compositional morphology for word representations and language modelling," in *International Conference on Machine Learning*, 2014, pp. 1899–1907.

[48] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, 2013, pp. 3111–3119.

[49] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[50] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, M. Walker, H. Ji, and A. Stent, Eds. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp.

2227–2237. [Online]. Available: https://aclanthology.org/N18-1202/

[51] P. Gage, "A new algorithm for data compression," *C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.

[52] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, Long Beach, CA, USA, 2017.

[53] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, E. Blanco and W. Lu, Eds. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 66–71. [Online]. Available: https://aclanthology.org/D18-2012/

[54] M. Schuster and K. Nakajima, "Japanese and Korean voice search," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012, pp. 5149–5152.

[55] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized autoregressive pretraining for language understanding," in *Advances in Neural Information Processing Systems*, 2019, pp. 5754–5764.

[56] Y. Goldberg, "A primer on neural network models for natural language processing," *Journal of Artificial Intelligence Research*, 2016.

[57] T. Shi and Z. Liu, "Linking GloVe with Word2Vec," *arXiv preprint arXiv:1411.5595*, 2014.

[58] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *Advances in Neural Information Processing Systems*, 2014, pp. 2177–2185.

[59] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[60] J. Firth, "A synopsis of linguistic theory 1930-1955," in *Studies in Linguistic Analysis*. Philological Society, Oxford, 1957, reprinted in Palmer, F. (ed. 1968) Selected Papers of J. R. Firth, Longman, Harlow.

[61] E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng, "Improving word representations via global context and multiple word prototypes," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics, 2012, pp. 873–882.

[62] O. Levy and Y. Goldberg, "Dependency-based word embeddings," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2014, pp. 302–308.

[63] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International Conference on Machine Learning*. Bejing, China: PMLR, 2014, pp. 1188–

1196.

[64] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, A. Moschitti, B. Pang, and W. Daelemans, Eds. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. [Online]. Available: https://aclanthology.org/D14-1179/

[65] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, 2014, pp. 3104–3112.

[66] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler, "Skip-thought vectors," in *Advances in Neural Information Processing Systems*, 2015, pp. 3294–3302.

[67] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[68] J. Li, T. Luong, and D. Jurafsky, "A hierarchical neural autoencoder for paragraphs and documents," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, C. Zong and M. Strube, Eds. Beijing, China: Association for Computational Linguistics, Jul. 2015, pp. 1106–1115. [Online]. Available: https://aclanthology.org/P15-1107/

[69] F. Hill, K. Cho, and A. Korhonen, "Learning distributed representations of sentences from unlabelled data," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, K. Knight, A. Nenkova, and O. Rambow, Eds. San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 1367–1377. [Online]. Available: https://aclanthology.org/N16-1162/

[70] B. McCann, J. Bradbury, C. Xiong, and R. Socher, "Learned in translation: Contextualized word vectors," in *Advances in Neural Information Processing Systems*, 2017, pp. 6294–6305.

[71] A. Akbik, D. Blythe, and R. Vollgraf, "Contextual string embeddings for sequence labeling," in *Proceedings of the 27th International Conference on Computational Linguistics*, 2018, pp. 1638–1649.

[72] S. Subramanian, A. Trischler, Y. Bengio, and C. J. Pal, "Learning general purpose distributed sentence representations via large scale multi-task learning," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=B18WgG-CZ

[73] M. Artetxe and H. Schwenk, "Massively multilingual sentence embeddings for zero-shot

cross-lingual transfer and beyond," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 597–610, 2019.

[74] M. Auli, M. Galley, C. Quirk, and G. Zweig, "Joint language and translation modeling with recurrent neural networks," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, October 2013, pp. 18–21.

[75] A. Conneau, R. Rinott, G. Lample, A. Williams, S. Bowman, H. Schwenk, and V. Stoyanov, "XNLI: Evaluating cross-lingual sentence representations," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 2475–2485. [Online]. Available: https://aclanthology.org/D18-1269/

[76] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, "Semi-supervised recursive autoencoders for predicting sentiment distributions," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK: Association for Computational Linguistics, 2011, pp. 151–161.

[77] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng, "Semantic compositionality through recursive matrix-vector spaces," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, 2012, pp. 1201–1211.

[78] M.-T. Luong, R. Socher, and C. D. Manning, "Better word representations with recursive neural networks for morphology," in *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, Sofia, Bulgaria, 2013, pp. 104–113.

[79] H. Zhao, Z. Lu, and P. Poupart, "Self-adaptive hierarchical sentence model," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, Buenos Aires, Argentina, 2015.

[80] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, C. Zong and M. Strube, Eds. Beijing, China: Association for Computational Linguistics, Jul. 2015, pp. 1556–1566. [Online]. Available: https://aclanthology.org/P15-1150/

[81] D. Yogatama, P. Blunsom, C. Dyer, E. Grefenstette, and W. Ling, "Learning to compose words into sentences with reinforcement learning," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=Skvgqgqxe

[82] J. Choi, K. M. Yoo, and S.-g. Lee, "Learning to compose task-specific tree structures," in

Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[83] A. Drozdov, P. Verga, M. Yadav, M. Iyyer, and A. McCallum, "Unsupervised latent tree induction with deep inside-outside recursive auto-encoders," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 1129–1141. [Online]. Available: https://aclanthology.org/N19-1116/

[84] J. R. Chowdhury and C. Caragea, "Modeling hierarchical structures with continuous recursive neural networks," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1975–1988.

[85] A. Simoulin and B. Crabbé, "Unifying parsing and tree-structured models for generating sentence semantic representations," in *Proceedings of the 2022 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2022, pp. 267–276.

[86] L. Sartran, S. Barrett, A. Kuncoro, M. Stanojević, P. Blunsom, and C. Dyer, "Transformer Grammars: Augmenting Transformer language models with syntactic inductive biases at scale," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 1423–1439, 2022.

[87] X. Hu, H. Mi, L. Li, and G. de Melo, "Fast-R2D2: A pretrained recursive neural network based on pruned CKY for grammar induction and text representation," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Y. Goldberg, Z. Kozareva, and Y. Zhang, Eds. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 2809–2821. [Online]. Available: https://aclanthology.org/2022.emnlp-main.181/

[88] X. Hu, Q. Zhu, K. Tu, and W. Wu, "Augmenting transformers with recursively composed multi-grained representations," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=u859gX7ADC

[89] T. Nakagawa, K. Inui, and S. Kurohashi, "Dependency tree-based sentiment classification using crfs with hidden variables," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010, pp. 786–794.

[90] S. R. Bowman, J. Gauthier, A. Rastogi, R. Gupta, C. D. Manning, and C. Potts, "A fast unified model for parsing and sentence understanding," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, K. Erk and N. A. Smith, Eds. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1466–1477. [Online]. Available:

https://aclanthology.org/P16-1139/

[91] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning*. ACM, 2008, pp. 160–167.

[92] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, K. Toutanova and H. Wu, Eds. Baltimore, Maryland: Association for Computational Linguistics, Jun. 2014, pp. 655–665. [Online]. Available: https://aclanthology.org/P14-1062/

[93] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu, "Neural machine translation in linear time," *arXiv preprint arXiv:1610.10099*, 2016.

[94] Z. Gan, Y. Pu, R. Henao, C. Li, X. He, and L. Carin, "Learning generic sentence representations using convolutional neural networks," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, M. Palmer, R. Hwa, and S. Riedel, Eds. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 2390–2400. [Online]. Available: https://aclanthology.org/D17-1254/

[95] K. Shuang, Z. Zhang, J. Loo, and S. Su, "Convolution-deconvolution word embedding: An end-to-end multi-prototype fusion embedding method for natural language processing," *Information Fusion*, vol. 53, pp. 112–122, 2020.

[96] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[97] J. Chung, K. Cho, and Y. Bengio, "A character-level decoder without explicit segmentation for neural machine translation," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, K. Erk and N. A. Smith, Eds. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1693–1703. [Online]. Available: https://aclanthology.org/P16-1160/

[98] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[99] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 1480–1489.

[100] Y. Kim, C. Denton, L. Hoang, and A. M. Rush, "Structured attention networks," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=HkE0Nvqlg

[101] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured

self-attentive sentence embedding," *arXiv preprint arXiv:1703.03130*, 2017.

[102] T. Shen, T. Zhou, G. Long, J. Jiang, S. Pan, and C. Zhang, "DiSAN: Directional self-attention network for RNN/CNN-free language understanding," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[103] T. Shen, T. Zhou, G. Long, J. Jiang, and C. Zhang, "Bi-directional block self-attention for fast and memory-efficient sequence modeling," *arXiv preprint arXiv:1804.00857*, 2018.

[104] T. Shen, T. Zhou, G. Long, J. Jiang, S. Wang, and C. Zhang, "Reinforced self-attention network: a hybrid of hard and soft attention for sequence modeling," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, ser. IJCAI'18. AAAI Press, 2018, p. 4345–4352.

[105] Y. Liu and M. Lapata, "Learning structured text representations," *Transactions of the Association for Computational Linguistics*, vol. 6, pp. 63–75, 2018.

[106] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, 2019.

[107] A. Conneau and G. Lample, "Cross-lingual language model pretraining," *Advances in neural information processing systems*, vol. 32, 2019.

[108] Q. Guo, X. Qiu, P. Liu, Y. Shao, X. Xue, and Z. Zhang, "Star-transformer," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 1315–1325. [Online]. Available: https://aclanthology.org/N19-1133/

[109] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, A. Korhonen, D. Traum, and L. Màrquez, Eds. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 2978–2988. [Online]. Available: https://aclanthology.org/P19-1285/

[110] K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu, "Mass: Masked sequence to sequence pre-training for language generation," in *International Conference on Machine Learning*, 2019, pp. 5926–5936.

[111] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3982–3992. [Online]. Available: https://aclanthology.org/D19-1410/

[112] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=H1eA7AEtvS

[113] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, "SpanBERT: Improving pre-training by representing and predicting spans," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 64–77, 2020.

[114] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, "Retrieval augmented language model pre-training," in *International conference on machine learning*. PMLR, 2020, pp. 3929–3938.

[115] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "ELECTRA: Pre-training text encoders as discriminators rather than generators," in *ICLR*, 2020. [Online]. Available: https://openreview.net/pdf?id=r1xMH1BtvB

[116] G. Lample, A. Conneau, M. Ranzato, L. Denoyer, and H. Jégou, "Word translation without parallel data," in *International conference on learning representations*, 2018.

[117] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," 2023. [Online]. Available: https://arxiv.org/abs/2310.06825

[118] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican *et al.*, "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.

[119] Anthropic, "The claude 3 model family: Opus, sonnet, haiku." [Online]. Available: https://api.semanticscholar.org/CorpusID:268232499

[120] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[121] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford *et al.*, "Gpt-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.

[122] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[123] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[124] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[125] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, virtual, 2020, pp. 1877–1901.

[126] O. Irsoy and C. Cardie, "Deep recursive neural networks for compositionality in language," in *Advances in Neural Information Processing Systems*, vol. 27, Montreal, Quebec, Canada, 2014.

[127] A. Frandsen and R. Ge, "Understanding composition of word embeddings via tensor decomposition," *arXiv:1902.00613*, 2019.

[128] M. Hartung, F. Kaupmann, S. Jebbara, and P. Cimiano, "Learning compositionality functions on word embeddings for modelling attribute meaning in adjective-noun phrases," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, Valencia, Spain, 2017, pp. 54–64.

[129] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, M. Lapata, P. Blunsom, and A. Koller, Eds. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 427–431. [Online]. Available: https://aclanthology.org/E17-2068/

[130] S. Arora, Y. Liang, and T. Ma, "A simple but tough-to-beat baseline for sentence embeddings," in *International Conference on Learning Representations*, 2017.

[131] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu, "Towards universal paraphrastic sentence embeddings," *arXiv preprint arXiv:1511.08198*, 2015.

[132] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," *Distill*, 2017, https://distill.pub/2017/feature-visualization.

[133] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang, "Large-scale hierarchical text classification with recursively regularized deep graph-CNN," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 1063–1072.

[134] S. Chen and W. Guo, "Autoencoders in deep learning—a review with new perspectives."

[135] M. Tschannen, O. Bachem, and M. Lucic, "Recent advances in autoencoder-based representation learning," in *Third workshop on Bayesian Deep Learning (NeurIPS 2018)*, 2018. [Online]. Available: http://www.nari.ee.ethz.ch/pubs/p/autoenc2018

[136] S. R. Bowman, L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space," in *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, S. Riezler and Y. Goldberg, Eds. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 10–21. [Online]. Available: https://aclanthology.org/K16-1002/

[137] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT press, 2016.

[138] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[139] G. E. Hinton *et al.*, "Learning distributed representations of concepts," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, vol. 1. Amherst, MA, 1986, p. 12.

[140] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," *Advances in Neural Information Processing Systems*, vol. 13, 2000.

[141] C. Goller and A. Kuchler, "Learning task-dependent distributed representations by backpropagation through structure," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 1. IEEE, 1996, pp. 347–352.

[142] R.-M. Dechaine, S. Burton, and E. Vatikiotis-Bateson, *Linguistics for Dummies*. John Wiley & Sons, 2012.

[143] K. Babić and A. Meštrović, "Recursively autoregressive autoencoder for pyramidal text representation," *IEEE Access*, vol. 12, pp. 71 361–71 370, 2024.

[144] J. P. Wahle, T. Ruas, T. Foltỳnek, N. Meuschke, and B. Gipp, "Identifying machine-paraphrased plagiarism," in *International Conference on Information*. virtual: Springer, 2022, pp. 393–413.

[145] J. McAuley and J. Leskovec, "Hidden factors and hidden topics: Understanding rating dimensions with review text," in *Proceedings of the 7th ACM Conference on Recommender Systems*, 2013, pp. 165–172.

[146] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[147] N. Ljubešić, F. Klubička, Ž. Agić, and I.-P. Jazbec, "New inflectional lexicons and training corpora for improved morphosyntactic annotation of croatian and serbian," in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, 2016, pp. 4264–4270.

[148] S. Crossley, A. Heintz, J. S. Choi, J. Batchelor, M. Karimi, and A. Malatinszky, "A large-scale corpus for assessing text readability," *Behavior Research Methods*, vol. 55, no. 2, pp. 491–507, 2023.

[149] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer Science+ Business Media, LLC, 2006.

Chapter 8

Abbreviations

Abbreviation	Description
AE	Autoencoder
BPE	Byte Pair Encoding
BPTS	Backpropagation Through Structure
BPTT	Backpropagation Through Time
CNN	Convolutional Neural Network
DEPREL	Universal Dependency Relation
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MLM	Masked Language Model
MLP	Multilayer Perceptron
NLP	Natural Language Processing
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RvNN	Recursive Neural Network
SP	SentencePiece
SVM	Support Vector Machine
TFIDF	Term Frequency-Inverse Document Frequency
UPOS	Universal Part-of-Speech
WP	WordPiece

List of Figures

2.1. Word2Vec architecture. In addition to an input layer and an output layer, shallow architectures have a small number of hidden layers (one in this case). . . . 13

2.2. Recurrent architecture and the unfolding in time. Recurrent nodes have connections which lead to the next node, and connections which loop back to the same node. The unfolding in time shows the same node in three consecutive iterations. . . . 15

2.3. The LSTM cell. Variables x_t and h_t are input and output, respectively, at time t. Squares with "σ" or "tanh" represent layers, whereas ovals with "X", "+", or "tanh" represent pointwise operations. . . . 16

2.4. A parsing tree of a recursive neural network predicting word sentiment classes. The leaf nodes are input tokens, all the other nodes are representations of the combination of the child nodes. The root node is representation of the entire input text. . . . 19

2.5. Convolutional architecture. A convolution has multiple filters, and each filter has a kernel (a matrix of weights) that is being trained. The kernel slides over the values from the previous layer, producing values that are sent to the next layer. Each filter learns to recognize a different pattern. . . . 24

2.6. A visualization of a learned self-attention head on a sentence. The visualization shows learned relations between the words this self-attention head has learned. Each head learns a different kind of relations between the words. . . . 26

3.1. Two-dimensional principal component analysis (PCA) projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities [48]. This visualization highlights the ability of Skip-gram embeddings to organize semantic concepts and capture implicit relationships, such as the association between countries and their capitals, without explicit supervision. . . . 43

3.2. Visualization of an autoencoder architecture, consisting of an encoder that maps the input (e.g., a picture of a handwritten digit) to a lower-dimensional hidden representation (code), and a decoder that reconstructs the input from the hidden representation. . . . 44

3.3. Illustration of the autoregressive modeling process for sentence generation. The model predicts the next word in a sequence step by step, based on a fixed window of four preceding words. 46
3.4. Morphological Recursive Neural Network [78]. A vector representation for the word "unfortunately" is constructed from morphemic vectors: un^{pre}, $fortunate^{stm}$, ly^{suf}. 48

4.1. A visualized example of a pyramidal recursion in the PyRv method. The lowest-level nodes correspond to input tokens. Moving upward, the nodes within the three pyramids represent combined subword embeddings. At the pyramid peaks, nodes represent word embeddings, and higher nodes signify combined word embeddings, representing phrases. 53
4.2. PyRvNN model architecture. The inputs are representations of adjacent nodes (one-hot or dense embeddings). Boxes connected with an arrow are fully connected layers. Autoregressive and autoencoding heads are identical, only the output labels they recieve differ. 55
4.3. Decoding accuracy versus embedding dimension for different pyramid levels. . 57
4.4. Average processing time per pyramid level versus embedding dimension. . . . 57
4.5. Average processing time per pyramid level versus input size for different embedding dimensions. 58
4.6. Average GPU memory usage per pyramid level versus embedding dimension. . 59
4.7. Average GPU memory usage per pyramid level versus input size for different embedding dimensions. 60
4.8. Density plot of word counts per paragraph. 62
4.9. Utilizing the PyRv approach, the phrase "embed this phrase" undergoes recursive embedding. Tokenized words (subwords) are received at the bottom pyramid level, with each node in the hierarchy representing all the tokens below it, illustrating the recursive embedding process. 64
4.10. Visualization of loss normalization by level. 65
4.11. Loss by pyramid level for the initial untrained model, with and without normalization. 65
4.12. Loss by pyramid level for the trained model, with and without normalization. The loss was normalized while training the model. 66
4.13. Average magnitude of PyRvNN representations across different pyramid levels. The figure shows Magnitude Enc and Magnitude Reg, corresponding to the autoencoding and autoregressive head output magnitudes, respectively. Left and right representations are averaged into these two plots due to their very similar magnitudes. 66

4.14. Loss curves during the initial training phase, showing the impact of the progressive training schedule across different loss components. 68
4.15. Same loss curves as in Figure 4.14, but scaled up to emphasize smaller loss variations. 69

5.1. Decodability accuracy for each word length (token count) and three phrase pyramid levels, tested on Wikipedia texts. 75
5.2. Decodability accuracy for each word length (token count) and three phrase pyramid levels, tested on texts from Amazon reviews. 76
5.3. Histogram illustrating the frequency distribution of the number of tokens per word. Data was computed using 10,000 paragraphs from the Wikipedia dataset. The histogram is truncated at 8 tokens per word. The plot shows that longer words (e.g., 6+ tokens) are significantly less frequent, supporting the hypothesis that reduced training data for these lengths contributes to the accuracy drop seen in Figure 5.1. 77
5.4. A visual representation of pyramidal recursion in the PyRv+FT method. The lowest-level nodes correspond to fastText-embedded words. As we move upward, the nodes represent combined word embeddings, capturing phrase-level meanings. 79
5.5. UPOS task, weighted average: accuracy as a function of the Gaussian STD factor. As the STD factor increases, the weights assigned to context words become more uniform, degrading performance. The best result (95%) is obtained with STD factor 0.15, where the middle word dominates the representation (86% of total weight). Plots for F1 score are shown in Appendix B: Figures B.1 and B.2. 83
5.6. DEPREL task, weighted average: accuracy as a function of the Gaussian STD factor. Maximum performance (74%) is reached at STD factor 0.1, where the middle word receives 79% of the total weight. As the STD factor increases, the performance degrades toward the baseline of uniform averaging (34%). Plots for F1 score are shown in Appendix B: Figures B.3 and B.4. 85
5.7. DEPREL relative F1 score ratios (by class) comparing different composition methods. The left plot (a) shows the ratio of F1 scores for "mean fastText 5 words" versus "fastText 1 word", while the right plot (b) compares "PyRv+FT 5 words" versus "fastText 1 word". Each bar represents a class, and the length of the bar indicates the relative performance of the model. Classes are ordered by support value in the test set (larger on the top). 86

5.8. Visualization of the PyRv+FT representation space (reduced from 300 dimensions using t-SNE). Highlighted areas A, B, and C contain phrases with the prepositions 'na' (eng. 'on') and 'u' (eng. 'in'). Tables 5.7 and 5.8 (translated: 5.5 and 5.6) provide detailed examples of these phrases and their connections within the space. 89
5.9. Visualization of the PyRv+FT representation space (reduced from 300 dimensions using t-SNE). Highlighted areas A and D contain phrases structured around the prepositions 'na' (eng. 'on') and 'u' (eng. 'in'). Tables 5.7 and 5.8 (translated: 5.5 and 5.6) present examples of these phrases and their relationships in the embedding space. 90
5.10. Density plot of scores. 96

A.1. Overview of the PyRv implementation architecture, detailing the modular design and interactions between main components, including data preparation, training algorithm, model architecture, and data manipulation modules. 126

B.1. UPOS task, weighted average: macro F1 score as a function of the Gaussian STD factor. 135
B.2. UPOS task, weighted average: weighted F1 score as a function of the Gaussian STD factor. 136
B.3. DEPREL task, weighted average: macro F1 score as a function of the Gaussian STD factor. 137
B.4. DEPREL task, weighted average: weighted F1 score as a function of the Gaussian STD factor. 138

List of Tables

2.1. The categorization of shallow models by input and representation level. All listed models are unsupervised. 13
2.2. The categorization of recurrent models by input level, representation level, and supervision. 17
2.3. The categorization of recursive models by input level, supervision, and parsing tree source. All the listed models learn sentence+ representations. Un/supervised supervision represents both unsupervised and supervised learning. 20
2.4. The categorization of convolutional models by input level, representation level, and supervision. Un/supervised supervision represents both unsupervised and supervised learning. 24
2.5. The categorization of attention models by input level and supervision. All listed models learn sentence+ representations. Un/supervised supervision represents both unsupervised and supervised learning. 27

4.1. Wikipedia dataset statistics 62

5.1. Decodability accuracy for each word length (token count) and three phrase pyramid levels, tested on Wikipedia texts. 75
5.2. Decodability accuracy for each word length (token count) and three phrase pyramid levels, tested on texts from Amazon reviews. 76
5.3. UPOS results, Macro and Weighted averages. 82
5.4. DEPREL results, Macro and Weighted averages. 84
5.5. Phrases by areas (A, B, and C) in the PyRv+FT representation space, translated to English (some phrases are longer when translated). 88
5.6. Phrases in area D in the PyRv+FT representation space, translated to English (some phrases are longer when translated). 91
5.7. Phrases by areas (A, B, and C) in the PyRv+FT representation space (original Croatian phrases). 92
5.8. Phrases in area D in the PyRv+FT representation space (original Croatian phrases). 93
5.9. Memorization results, 260 epochs. 94
5.10. Memorization results, 400 epochs. 94

5.11. Machine-paraphrased plagiarism classification results. 95
5.12. Readability classification results. 97

B.1. DEPREL evaluation results (by class) using the fastText embedding method (single-word embeddings). 139
B.2. DEPREL evaluation results (by class) using the fastText embedding method (averaging embeddings of five words). 140
B.3. DEPREL evaluation results (by class) using the PyRv+FT embedding method (composing embeddings of five words). 141

Appendix A

A: Implementation

This Appendix details the implementation of the proposed Pyramidal Recursive learning (PyRv) method, specifically designed for training neural networks to hierarchically represent text. The implementation is organized across modular components, each focusing on a distinct aspect of the system's functionality. The design choices emphasize the separation of concerns, scalability, and adaptability to various input configurations. Figure A.1 provides an overview of the system's architecture.

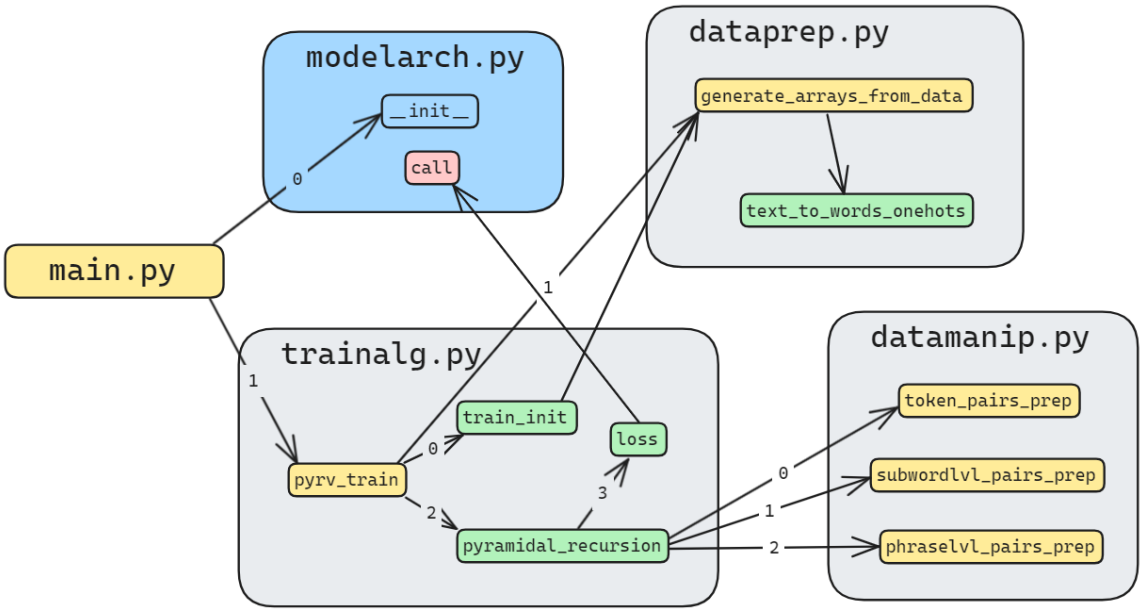


Figure A.1: Overview of the PyRv implementation architecture, detailing the modular design and interactions between main components, including data preparation, training algorithm, model architecture, and data manipulation modules.

The implementation consists of five core modules:

- Module **main.py** (A.1): Orchestrates the training process by progressively training on deeper pyramidal hierarchies, managing learning rates, and defining depth parameters for subword and phrase levels.

- Module **modelarch.py** (A.2): Encapsulates the neural network architecture, supporting hierarchical encoding, decoding, and representation handling.
- Module **trainalg.py** (A.3): Implements the training algorithms, including pyramidal recursion and optimization processes.
- Module **dataprep.py** (A.4): Handles the preparation and streaming of input data, including tokenization and one-hot encoding of text.
- Module **datamanip.py** (A.5): Facilitates the creation and manipulation of hierarchical data structures, generating representation pairs and neighbors for training.

The modular design ensures a clear distinction between the PyRvNN model and the PyRv training method, aligning with the principles of maintainable and extensible codebases. Additionally, the data streaming pipeline supports efficient input-output handling, enabling scalability for large datasets. Each module is detailed in the following sections, with pseudocode illustrating the core functionality implemented within these files.

The implementation is written in Python and utilizes TensorFlow as the primary deep learning framework. The complete source code for this implementation is available at the following link: `https://github.com/karlo-babic/pyrv`

A.1 Module main.py

The `Main.py` module (Algorithm 1) serves as the entry point for orchestrating the training of the PyRv model. It defines a progressive training schedule that incrementally exposes the model to deeper pyramidal hierarchies in a staged manner. This schedule ensures efficient and stable learning by managing the depth of subword and phrase pyramid levels, as well as adjusting the learning rate across training stages.

The training process starts with subword-level pyramid hierarchies, beginning at level SW_{L0} and gradually advancing to level SW_{Lmax}. Once the model is adequately trained on these subword hierarchies, it proceeds to phrase-level pyramids, starting at level PH_{L0} and extending up to level PH_{Lmax}. Each stage is configured with a specific number of training steps and learning rate.

The learning rate is initially set to LR_{init} and is reduced to LR_{reduce} in later stages, promoting stable convergence as the model encounters more complex pyramid levels.

An instance of `PyRvNN` is created in `Main.py` and passed as an argument to the PyRv learning system. This design allows for seamless integration of other neural network architectures into the PyRv framework by replacing `PyRvNN` with alternative implementations.

Algorithm 1: Main (example of progressive model training)

Input: text dataset
Output: trained *model* (PyRvNN instance after training)
$model \leftarrow modelarch.PyRvNN.\underline{\textbf{\_\_init\_\_}}()$
$trainalg.\underline{\textbf{pyrv\_train}}(model, max\_subword\_depth = 1, max\_phrase\_depth = 0, num\_steps = 10000, learning\_rate = 0.001)$
for $i \leftarrow 2$ **to** 9 **do**
    $trainalg.\underline{\textbf{pyrv\_train}}(model, max\_subword\_depth \leftarrow i, max\_phrase\_depth \leftarrow 0, num\_steps \leftarrow 2000, learning\_rate \leftarrow 0.001)$
$trainalg.\underline{\textbf{pyrv\_train}}(model, max\_subword\_depth = 10, max\_phrase\_depth = 1, num\_steps = 10000, learning\_rate = 0.001)$
for $i \leftarrow 1$ **to** ... **do**
    $trainalg.\underline{\textbf{pyrv\_train}}(model, max\_subword\_depth \leftarrow 10, max\_phrase\_depth \leftarrow 2, num\_steps \leftarrow 100000, learning\_rate \leftarrow 0.0005)$

A.2 Module modelarch.py

The `modelarch.py` module implements the core PyRvNN architecture as described in Section 4.2. It defines the network's initialization and forward pass functionality, encapsulated within the `PyRvNN` class. This module is central to the composition process of the PyRv framework.

The core operation of the `modelarch.py` module is implemented in the `call` function, which is invoked during the forward pass. The pseudocode for this function is detailed in Algorithm 2. It takes a pair of input representations as arguments and outputs:

- The composed embedding resulting from the combination of the input pair.
- Decoded representations of the input pair.
- Neighbors to the left and right of the composed embedding.

Algorithm 2: Model Call: **call()**

Input: *inputs* (pair of one-hot and dense vectors),
training (boolean flag indicating training or inference mode),
return_embs (boolean flag to include embeddings in the output)
Output: *outputs* (decoded representations of the input pair and their neighbors),
embs (composed embedding, included if *return_embs* is True)
Function *call(inputs, training, return_embs)*:
    $embs \leftarrow encode(inputs)$
    $outputs \leftarrow decode(embs)$
    if *return_embs* **then**
        return *outputs*, *embs*
    else
        return *outputs*

A.3 Module trainalg.py

The `trainalg.py` module is central to implementing the PyRv method. This module encapsulates the core training algorithms and associated processes. Below, we describe the primary functions included in this module.

A.3.1 Function `pyrv_train()`

The `pyrv_train` function (Algorithm 3) serves as the entry point for the training process. It orchestrates the overall training flow, including initialization, data processing, recursive loss computation, and optimization.

Algorithm 3: Pyramidal Recursive Training: **pyrv_train()**

Input: *model* (PyRvNN instance to be trained),
max_subword_depth (maximum depth for subword pyramid levels),
max_phrase_depth (maximum depth for phrase pyramid levels),
num_steps (number of training steps),
learning_rate (learning rate for the optimizer)
Output: trained *model* (PyRvNN instance after training)
Function *pyrv_train(model, max_subword_depth, max_phrase_depth, num_steps, learning_rate)*:

```
..., data_gen, ... ← train_init()
while do_train do
    input_data, ... ← next(data_gen)    dataprep.generate_arrays_from_data() output
    loss ← pyramidal_recursion(model, input_data, ...)
    optimizer(model, loss)
```

A.3.2 Function `train_init()`

The `train_init` function (Algorithm 4) initializes the training environment. This includes setting up the optimizer and preparing the data generator. The function ensures that all components are ready for training.

A.3.3 Function `pyramidal_recursion()`

The `pyramidal_recursion` function (Algorithm 5) implements the core of the Pyramidal Recursive learning method. It processes input data through successive levels of the hierarchical pyramid, computing loss and composing embeddings at each level. Depending on the level, it prepares token pairs, subword pairs, or word/phrase pairs for encoding and regression.

Algorithm 4: Training Initialization: **train_init()**

Input: *model* (PyRvNN instance to be trained),
optimizer (optimization algorithm instance),
learning_rate (learning rate for the optimizer)
Output: *optimizer* (initialized optimizer),
init_step (starting training step),
data_gen (data generator instance),
do_train (flag indicating whether training should continue)
Function *train_init(model, optimizer, learning_rate)*:

```
...
data_gen ← dataprep.generate_arrays_from_data(...)
return optimizer, init_step, data_gen, do_train
```

Algorithm 5: Pyramidal Recursion: **pyramidal_recursion()**

Input: *model* (PyRvNN instance),
input_data (initial input one-hot representations),
max_subword_depth (maximum depth of the subword pyramid),
max_phrase_depth (maximum depth of the phrase pyramid)
Output: *total_loss* (average loss computed over all levels of the pyramid)
Function *pyramidal_recursion(model, input_data, max_subword_depth, max_phrase_depth)*:

```
total_loss ← 0
for each embedding level do
    if below word level then
        if at 0^th level then
            enc_pairs, reg_pairs ← datamanip.token_pairs_prep(input_data)
        else
            enc_pairs, reg_pairs, word_lvl_lengths, word_embs, reached_words ←
              datamanip.subwordlvl_pairs_prep(embs, ...)
            if reached_words then
                embs ← word_embs
    if at phrase level then
        enc_pairs, reg_pairs ← datamanip.phraselvl_pairs_prep(embs)
    X ← enc_pairs
    y ← concatenate(enc_pairs, reg_pairs)
    lvl_loss, embs ← loss(model, X, y)
    total_loss ← total_loss + lvl_loss
total_loss ← total_loss / (subword_depth + phrase_depth)
return total_loss
```

A.3.4 Function `loss()`

The `loss` function (Algorithm 6) computes the combined loss for the model. It evaluates autoencoding and autoregressive objectives for both one-hot and dense representations using Binary Cross-Entropy and Huber loss, respectively. These losses are aggregated to guide the model's learning process.

Algorithm 6: Loss Calculation: **loss()**

Input: *model* (PyRvNN instance),
X (input pairs for encoding),
y (target pairs for autoencoding and autoregressive heads)
Output: *loss* (aggregated loss from autoencoding and autoregressive objectives)
Function *loss(model,* **X***,* **y***)***:**

$\hat{\mathbf{y}} \leftarrow model.\underline{\textbf{call}}(\mathbf{X})$

$loss_{\text{onehot\_enc}} \leftarrow \frac{\text{BinaryCrossentropy}(\mathbf{y}_{\text{onehot\_enc\_left}}, \hat{\mathbf{y}}_{\text{onehot\_enc\_left}}) + \text{BinaryCrossentropy}(\mathbf{y}_{\text{onehot\_enc\_right}}, \hat{\mathbf{y}}_{\text{onehot\_enc\_right}})}{2}$

$loss_{\text{onehot\_reg}} \leftarrow \frac{\text{BinaryCrossentropy}(\mathbf{y}_{\text{onehot\_reg\_left}}, \hat{\mathbf{y}}_{\text{onehot\_reg\_left}}) + \text{BinaryCrossentropy}(\mathbf{y}_{\text{onehot\_reg\_right}}, \hat{\mathbf{y}}_{\text{onehot\_reg\_right}})}{2}$

$loss_{\text{dense\_enc}} \leftarrow \frac{\text{Huber}(\mathbf{y}_{\text{dense\_enc\_left}}, \hat{\mathbf{y}}_{\text{dense\_enc\_left}}) + \text{Huber}(\mathbf{y}_{\text{dense\_enc\_right}}, \hat{\mathbf{y}}_{\text{dense\_enc\_right}})}{2}$

$loss_{\text{dense\_reg}} \leftarrow \frac{\text{Huber}(\mathbf{y}_{\text{dense\_reg\_left}}, \hat{\mathbf{y}}_{\text{dense\_reg\_left}}) + \text{Huber}(\mathbf{y}_{\text{dense\_reg\_right}}, \hat{\mathbf{y}}_{\text{dense\_reg\_right}})}{2}$

$loss \leftarrow loss_{\text{onehot\_enc}} + loss_{\text{onehot\_reg}} + loss_{\text{dense\_enc}} + loss_{\text{dense\_reg}}$

return *loss*

A.4 Module dataprep.py

The `dataprep.py` module is responsible for preparing and generating data arrays required for training the Pyramidal Recursive Neural Network (PyRvNN) model. This module includes functions for streaming input data, converting text into token-level one-hot encodings, and outputting the necessary data arrays for subsequent processing.

A.4.1 Function `generate_arrays_from_data()`

The `generate_arrays_from_data` function (Algorithm 7) generates arrays of data from the input stream. It operates in a loop, processing paragraphs of text into token-level one-hot encodings and preparing the data in a format suitable for training, outputting one encoded paragraph each time `next(data_gen)` is called (in Algorithm 3).

A.4.2 Function `text_to_onehots_by_words()`

The `text_to_onehots_by_words` function (Algorithm 8) transforms text input into lists of token-level one-hot encodings. It splits the text into words, maps the words to their token

Algorithm 7: Data Generator: **generate_arrays_from_data()**

Input: *init_index_pos* (initial position index for the data stream)
Output: *onehots_by_words* (token-level one-hot encodings of a paragraph)
Function *generate_arrays_from_data(init_index_pos)*:
    while *True* **do**
        $paragraph, \ldots \leftarrow stream(\ldots)$
        $onehots\_by\_words \leftarrow$ **text_to_onehots_by_words**$(paragraph)$
        output $onehots\_by\_words, \ldots$

IDs, and then converts these IDs into one-hot representations. The output is a list where each word is represented by its respective subword-level one-hot encodings.

Algorithm 8: Text One-Hot Encoding: **text_to_onehots_by_words()**

Input: *text* (input text to be encoded as one-hot vectors)
Output: *onehots_by_words* (list of token-level one-hot encodings for each word in the input text)
Function *text_to_onehots_by_words(text)*:
    $words \leftarrow text.split()$
    $words\_tokenIDs \leftarrow \_string\_to\_tokenIDs(words)$
    $words\_onehots \leftarrow \_tokenIDs\_to\_onehots(words\_tokenIDs)$
    return *onehots_by_words*

A.5 Module datamanip.py

The `datamanip.py` module facilitates the preparation of input and output data pairs at various levels of the pyramidal hierarchy in the PyRv method. This module includes functions for organizing and pairing data at the token, subword, and phrase levels, preparing the inputs and outputs required for both the autoencoding and autoregressive tasks.

A.5.1 Function `token_pairs_prep()`

The `token_pairs_prep` function (Algorithm 9) prepares the input and output pairs for the zeroth (subword) level of the pyramid. It takes token-level one-hot representations and generates:

- `enc_pairs`: Pairs of neighboring nodes used as inputs and autoencoding outputs.
- `reg_pairs`: Pairs of left and right neighboring nodes used as autoregressive outputs.

A.5.2 Function `subwordlvl_pairs_prep()`

The `subwordlvl_pairs_prep` function (Algorithm 10) prepares pairs for the subword levels of the pyramid. It organizes subword embeddings into word-specific lists, enabling the

Algorithm 9: Token Pairs Preparation: **token_pairs_prep()**

Input: *input_data* (token-level one-hot representations)
Output: *enc_pairs* (pairs of neighboring nodes for input and autoencoding output), *reg_pairs* (pairs of left and right neighbors for autoregressive output)
Function *token_pairs_prep(input_data)*:
    enc_pairs ← *_make_pairs*(*input_data*) under each word
    reg_pairs ← *_make_neighbors*(*input_data*) under each word
    return *enc_pairs*, *reg_pairs*

recursive generation of:

- `enc_pairs`: Input and autoencoding output pairs at the subword level.
- `reg_pairs`: Autoregressive output pairs at the subword level.

If all subword embeddings are processed and word-level embeddings are reached, the function returns the word-level embeddings and a flag indicating the transition to the word level.

Algorithm 10: Subword Pairs Preparation: **subwordlvl_pairs_prep()**

Input: *embs* (subword embeddings),
word_lvl_lengths (list of word lengths by subword tokens),
word_embs (word-level embeddings),
subword_depth (maximum subword depth in the pyramid)
Output: *enc_pairs* (input and autoencoding output pairs at the subword level),
reg_pairs (autoregressive output pairs at the subword level),
word_lvl_lengths (updated word lengths list),
word_embs (updated word-level embeddings if reached),
reached_words (flag indicating if recursion has reached the word level for every word)
Function *subwordlvl_pairs_prep(embs, word_lvl_lengths, word_embs, subword_depth)*:
    for *each word in token_lvl_lengths* **do**
        Organize embeddings in lists for each word
    if *there are some embeddings of subwords left* **then**
        enc_pairs ← *_make_pairs*(*subword_embs*)
        reg_pairs ← *_make_neighbors*(*subword_embs*)
        return *enc_pairs*, *reg_pairs*, *word_lvl_lengths*, *word_embs*, *False*
    else
        when recursion reached word level
        return *None*, *None*, *None*, *word_embs*, *True*

A.5.3 Function `phraselvl_pairs_prep()`

The `phraselvl_pairs_prep` function (Algorithm 11) prepares pairs for the phrase levels of the pyramid. Using word or phrase embeddings as inputs, it generates:

- `enc_pairs`: Input and autoencoding output pairs at the phrase level.

- `reg_pairs`: Autoregressive output pairs at the phrase level.

Algorithm 11: Phrase Level Pairs Preparation: **phraselvl_pairs_prep()**

Input: *embs* (word or phrase embeddings)
Output: *enc_pairs* (input and autoencoding output pairs at the phrase level), *reg_pairs* (autoregressive output pairs at the phrase level)
Function *phraselvl_pairs_prep(embs)*:
    enc_pairs ← *_make_pairs*(*embs*)
    reg_pairs ← *_make_neighbors*(*embs*)
    return *enc_pairs*, *reg_pairs*

Appendix B

B: Additional Results

This Appendix contains additional compositionality results. Section 5.2 covers that topic.

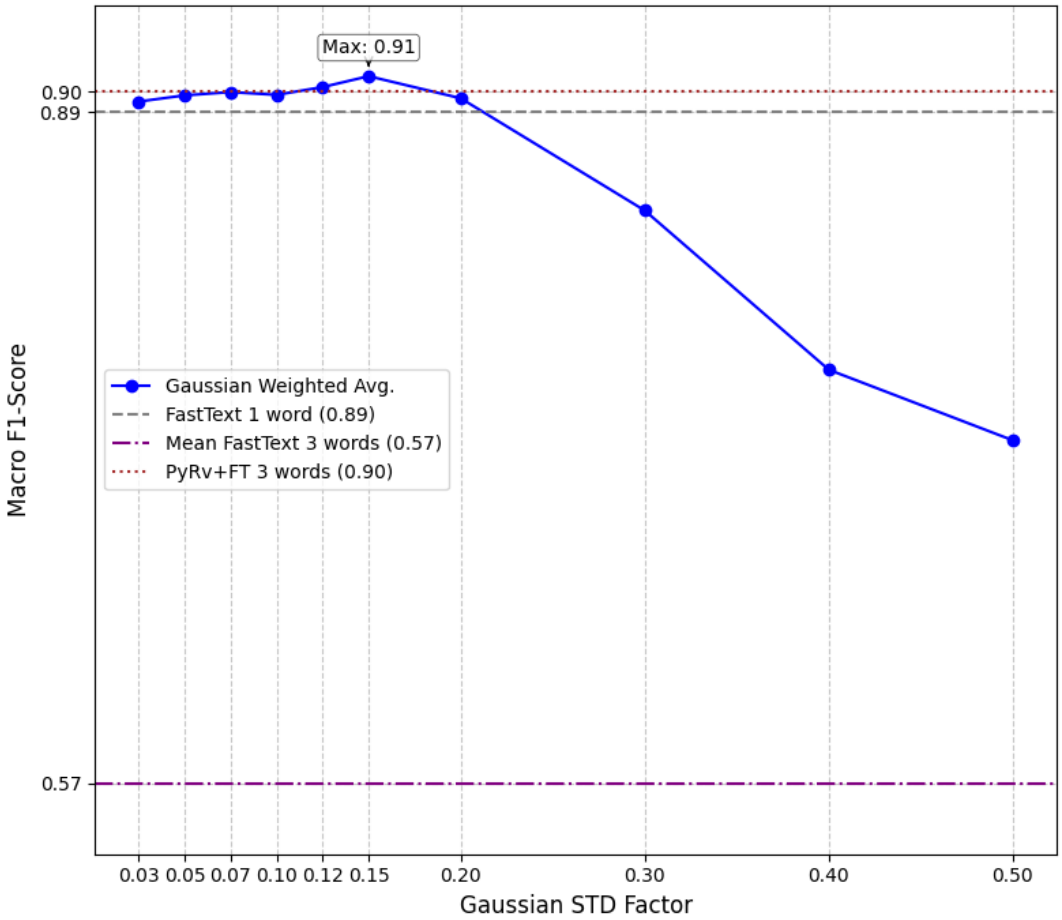


Figure B.1: UPOS task, weighted average: macro F1 score as a function of the Gaussian STD factor.

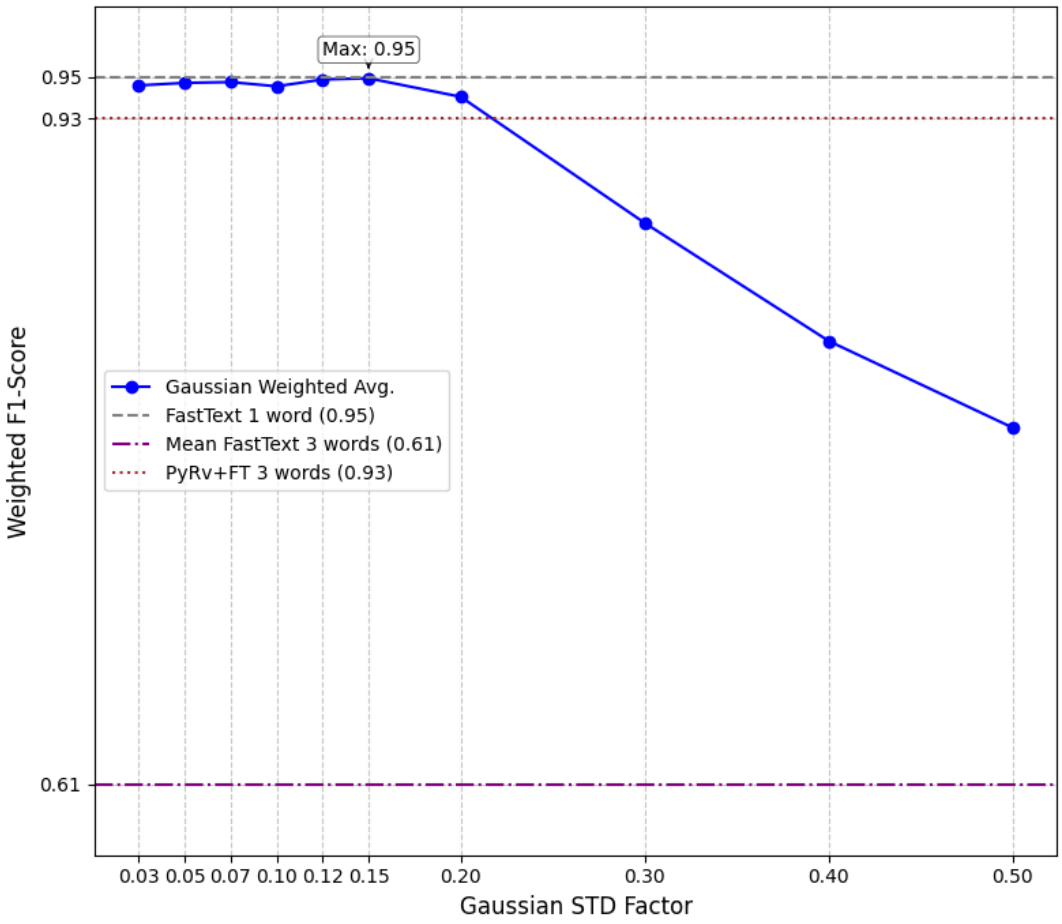


Figure B.2: UPOS task, weighted average: weighted F1 score as a function of the Gaussian STD factor.

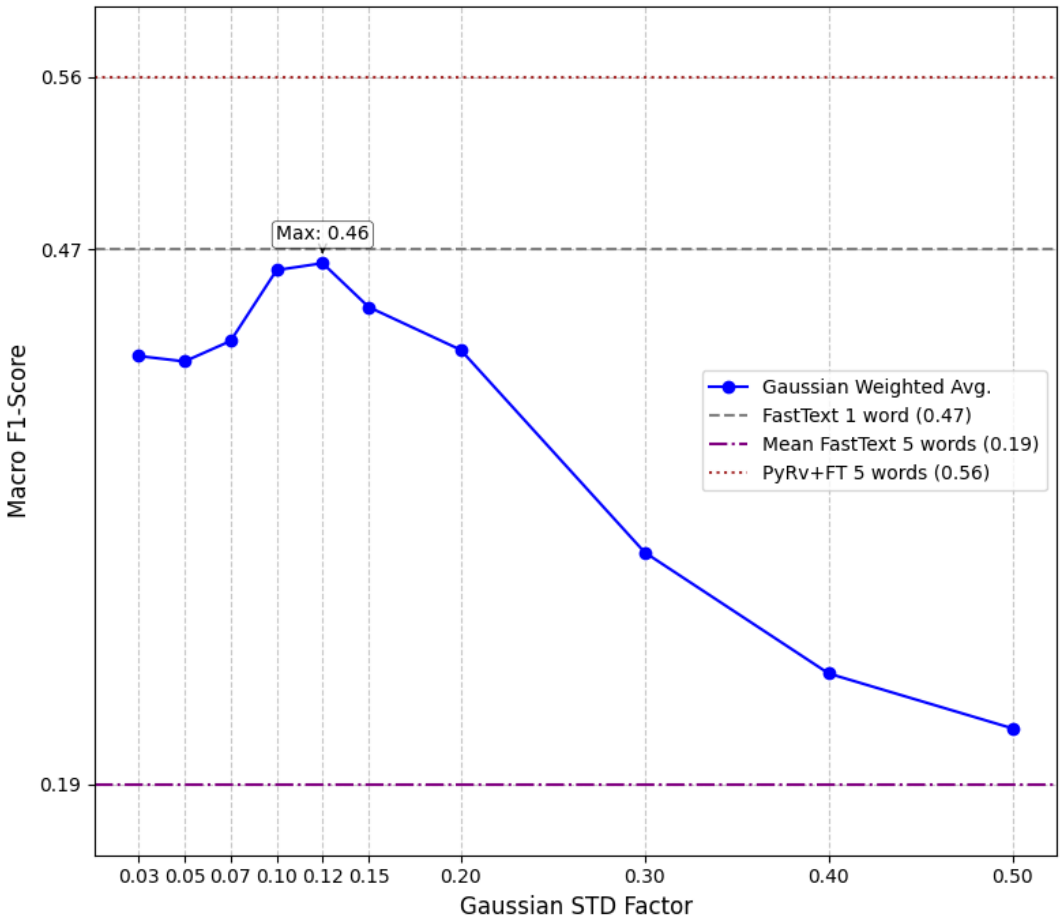


Figure B.3: DEPREL task, weighted average: macro F1 score as a function of the Gaussian STD factor.

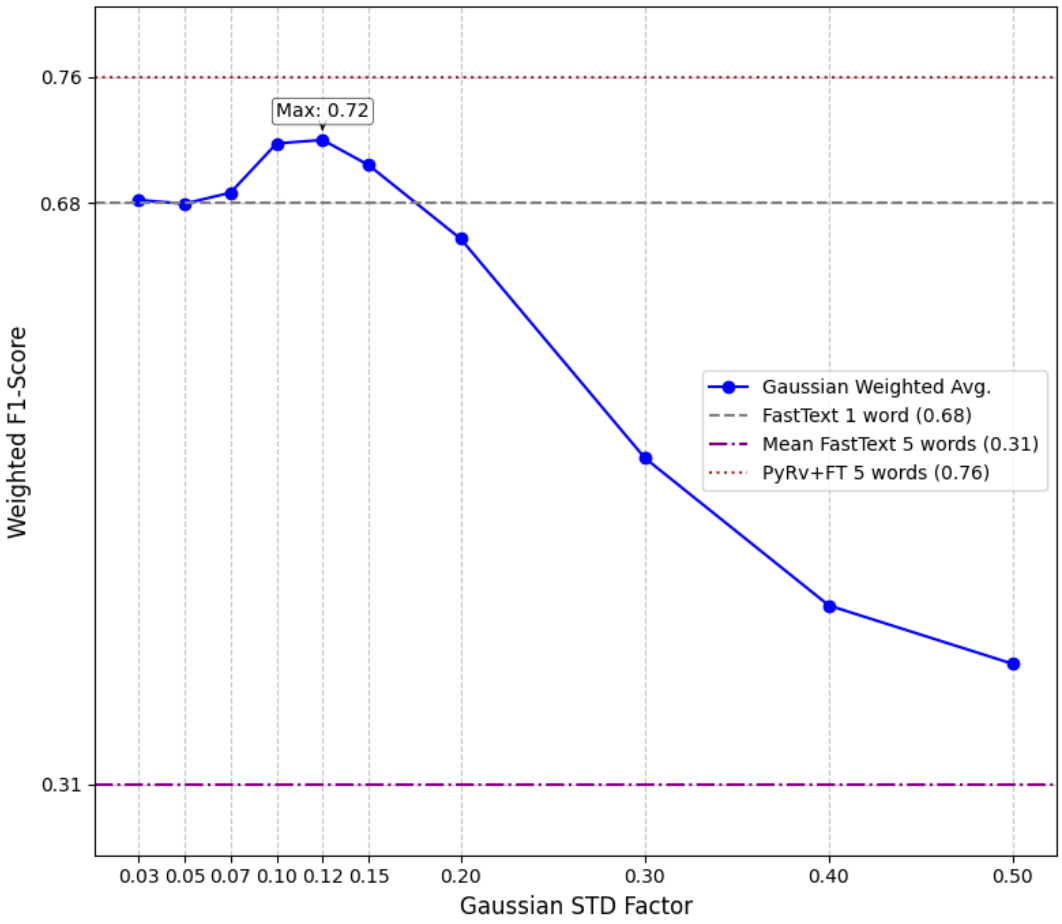


Figure B.4: DEPREL task, weighted average: weighted F1 score as a function of the Gaussian STD factor.

Table B.1: DEPREL evaluation results (by class) using the fastText embedding method (single-word embeddings).

Class	Precision	Recall	F1 score	Support
punct	1	1	1	3037
nmod	0.6	0.58	0.59	2437
case	0.96	0.98	0.97	2364
amod	0.79	0.93	0.85	2355
nsubj	0.53	0.66	0.59	1725
obl	0.48	0.57	0.52	1607
root	0.45	0.67	0.53	1136
conj	0.19	0.02	0.03	1134
obj	0.49	0.44	0.46	1072
aux	0.75	0.96	0.84	1037
cc	0.83	0.97	0.89	887
advmod	0.8	0.88	0.84	825
flat	0.54	0.69	0.61	689
mark	0.85	0.91	0.88	471
acl	0.44	0.08	0.14	452
cop	0.58	0.22	0.32	415
det	0.87	0.87	0.87	400
xcomp	0.61	0.7	0.65	350
expl	0.85	1	0.92	302
parataxis	0.63	0.38	0.47	300
ccomp	0.18	0.02	0.03	230
discourse	0.48	0.21	0.29	208
advcl	0.29	0.08	0.12	198
nummod:gov	0.72	0.9	0.8	187
appos	1	0.01	0.02	130
nummod	0.82	0.63	0.71	117
fixed	0.34	0.33	0.34	100
csubj	0	0	0	40
det:numgov	0.73	0.66	0.69	29
orphan	0	0	0	13
advmod:emph	0	0	0	5
flat:foreign	0	0	0	4
vocative	0	0	0	3
compound	0	0	0	1
macro avg	0.52	0.48	0.47	
weighted avg	0.68	0.71	0.68	

Table B.2: DEPREL evaluation results (by class) using the fastText embedding method (averaging embeddings of five words).

Class	Precision	Recall	F1 score	Support
punct	0.54	0.71	0.61	3037
nmod	0.42	0.35	0.39	2437
case	0.28	0.56	0.37	2364
amod	0.3	0.4	0.34	2355
nsubj	0.3	0.3	0.3	1725
obl	0.29	0.03	0.06	1607
root	0.26	0.17	0.21	1136
conj	0.27	0.38	0.32	1134
obj	0.26	0.12	0.16	1072
aux	0.35	0.34	0.35	1037
cc	0.22	0.25	0.24	887
advmod	0.29	0.24	0.26	825
flat	0.44	0.43	0.44	689
mark	0.33	0.27	0.3	471
acl	0.23	0.07	0.11	452
cop	0.22	0.14	0.17	415
det	0.25	0.13	0.17	400
xcomp	0.35	0.15	0.21	350
expl	0.26	0.35	0.3	302
parataxis	0.76	0.09	0.16	300
ccomp	0	0	0	230
discourse	0.67	0.03	0.06	208
advcl	0.14	0.01	0.01	198
nummod:gov	0.27	0.6	0.37	187
appos	0.2	0.02	0.04	130
nummod	0.24	0.13	0.17	117
fixed	0.31	0.24	0.27	100
csubj	0	0	0	40
det:numgov	0	0	0	29
orphan	0	0	0	13
advmod:emph	0	0	0	5
flat:foreign	0	0	0	4
vocative	0	0	0	3
compound	0	0	0	1
macro avg	0.25	0.19	0.19	
weighted avg	0.34	0.34	0.31	

Table B.3: DEPREL evaluation results (by class) using the PyRv+FT embedding method (composing embeddings of five words).

Class	Precision	Recall	F1 score	Support
punct	1	1	1	3037
nmod	0.74	0.71	0.73	2437
case	0.98	0.97	0.97	2364
amod	0.81	0.82	0.82	2355
nsubj	0.7	0.7	0.7	1725
obl	0.59	0.63	0.61	1607
root	0.61	0.69	0.65	1136
conj	0.66	0.62	0.64	1134
obj	0.52	0.65	0.58	1072
aux	0.78	0.94	0.85	1037
cc	0.91	0.95	0.93	887
advmod	0.68	0.82	0.74	825
flat	0.77	0.69	0.73	689
mark	0.91	0.91	0.91	471
acl	0.6	0.47	0.53	452
cop	0.69	0.39	0.5	415
det	0.72	0.69	0.71	400
xcomp	0.67	0.59	0.63	350
expl	0.86	0.99	0.92	302
parataxis	0.83	0.56	0.67	300
ccomp	0.41	0.16	0.23	230
discourse	0.65	0.52	0.58	208
advcl	0.38	0.22	0.28	198
nummod:gov	0.82	0.76	0.79	187
appos	0.42	0.38	0.4	130
nummod	0.67	0.76	0.71	117
fixed	0.67	0.61	0.64	100
csubj	0	0	0	40
det:numgov	0.5	0.52	0.51	29
orphan	0	0	0	13
advmod:emph	0	0	0	5
flat:foreign	0	0	0	4
vocative	0	0	0	3
compound	0	0	0	1
macro avg	0.58	0.55	0.56	
weighted avg	0.77	0.77	0.76	