Natural Language Processing

Course Notes

Fall 2025

Contents

Chapter 1

Language Modelling

Language modelling represents one of the foundational concepts in natural language processing, serving as the backbone for numerous NLP applications and systems. This chapter provides a comprehensive treatment of language models, their design principles, estimation procedures, and evaluation methodologies.

The primary objectives of this chapter are to understand what language models are, how to design them appropriately, how to estimate their parameters from data, and how to evaluate their performance. These four interconnected goals form the theoretical and practical foundation for working with language models in modern NLP.

1.1 Fundamentals of Language Models

1.1.1 Definition and Core Properties

A language model is formally defined as a probability distribution over the set of all strings in a language. This definition immediately implies two fundamental properties that distinguish language models from other probabilistic systems. First, a language model can assign a probability value to any piece of text within its domain. Second, a language model can generate text by drawing samples from its underlying probability distribution. These capabilities make language models extraordinarily versatile and explain their widespread adoption across various NLP tasks.

The practical applications of language models span a remarkable range of scenarios. In speech recognition systems, language models can order alternative sentences generated by the acoustic component, selecting the most probable sequence. In interactive systems, they enable autocomplete functionality by generating contextually appropriate continuations of user input. More broadly, language models serve as the backbone of sophisticated NLP systems including machine translation systems, text summarization pipelines, and conversational AI applications such as chatbots.

1.1.2 A Critical Perspective on Probability

A crucial insight that distinguishes rigorous probability theory from naive applications is understanding the nature of probability itself. Probability is not an intrinsic property of text that exists independently in the world waiting to be discovered and measured. This philosophical position, famously articulated in Bruno de Finetti's seminal work on probability theory with the provocative claim that "probability does not exist," underscores a fundamental truth: probability is an expression of preference, an artifact of the model or observer assigning it.

This understanding has profound methodological implications. We cannot learn language models through a regression-based approach that maps observed text to ground-truth target probabilities, since no such ground truth exists. Instead, we learn to assign probabilities to text

through a constructive process: we design a probabilistic model that articulates assumptions about how texts come about, and we optimize the parameters of this model using observed data and a chosen statistical criterion. In certain circumstances, particularly when working with large, representative corpora, these assigned probabilities can approximate sample frequencies observed in populations, lending them an empirical grounding. However, this remains a consequence of our model design choices and estimation procedure, not an inherent property of the text itself.

1.1.3 Criteria for Good Language Models

A language model earns the designation "good" when it satisfies a statistical notion of quality: it is a probability distribution whose samples resemble observed text. This criterion operates at the level of statistical patterns rather than individual sentences. If typical sentences in observed text have approximately thirty words, a good language model will generate samples that reproduce this pattern. If the typical sentence exhibits a subject-verb-object structure in a particular order, samples from a good language model will exhibit this pattern as well. This pattern-matching criterion is what we mean by statistical goodness, distinct from other possible notions of quality such as semantic coherence, factual accuracy, or adherence to particular linguistic conventions.

1.2 Designing Language Models

1.2.1 Components of Language Model Design

Designing a language model, being fundamentally a problem of specifying a probability distribution, requires two key decisions. The first is the choice of sample space: the set of outcomes that the language model can generate and assign probability to. The second is the specification of a probability mass function (pmf): a function that maps each and every outcome in the sample space to its assigned probability mass. These two components completely determine the structure and capabilities of the language model.

1.2.2 Text as a Data Type

Before proceeding with the formal design of language models, we must establish how text will be represented. Text is conceptually and formally treated as a finite sequence of discrete symbols, which are generically referred to as words, or more precisely as tokens. Digital text in its raw form consists of characters, but through the application of tokenization algorithms, this character sequence is segmented into a meaningful sequence of tokens. The choice of tokenization algorithm significantly affects the structure of the resulting language model. For example, with a tokenization procedure based on spaces and punctuation, the English text "what a nice dog!" becomes a sequence of five tokens: (what, a, nice, dog, !). Modern, general-purpose tokenization approaches are often grounded in compression algorithms such as byte pair encoding, which learn tokenization patterns from data rather than relying on hand-crafted rules.

1.2.3 Sample Space Construction

The construction of the sample space begins with defining a finite vocabulary W of words, typically comprising all unique tokens encountered in some large, representative dataset. Given this vocabulary, the sample space is conventionally chosen to be the set of all finite-length sequences composed of symbols from this vocabulary. This set, formally denoted as W^*, is what we call the language in a formal, non-linguistic sense.

Consider a concrete example with vocabulary $W = \{a, \text{cat}, \text{dog}, \text{nice}\}$. Within the language W^*, we find sentences such as "a nice cat" and "a dog," which are both valid sequences over this vocabulary. Interestingly, sequences such as "a a" or "nice a nice a nice" are also technically in W^*, even though they are not sensible English sentences. The sequence "a cute dog," by contrast, is not in W^* because the token "cute" does not appear in the vocabulary. It is possible to constrain the language to a meaningful subset of W^*, for instance to include only grammatical text, though such constraints are beyond the scope of this introductory treatment.

1.2.4 Formal Notation and Terminology

To establish precise formal foundations, we introduce the following notation. Let W denote a random variable whose outcome w is a symbol drawn from vocabulary W of size V. Let $X = \langle W_1, \ldots, W_L \rangle$ denote a random sequence of L words, which we may also write compactly as $W_{1:L}$. The outcome of X is denoted $\langle w_1, \ldots, w_\ell \rangle$ and represents a sequence of ℓ symbols from W, belonging to W^*.

A language model is a mechanism to assign a probability value $P_X(w_{1:\ell})$ to each and every outcome $w_{1:\ell} \in W^*$. Our task is to design a probability mass function that maps $w_{1:\ell}$ to its probability mass in a computationally and statistically efficient manner.

1.2.5 The Challenge of Infinite Sample Spaces

The fundamental challenge in language modelling arises from the fact that P_X is a distribution over a countably infinite space of variable-length sequences. To appreciate the magnitude of this challenge, consider attempting to represent the language model using a tabular representation, the most straightforward approach to specifying a discrete probability distribution.

In such a tabular approach, we would create a table in which each row corresponds to a unique outcome x in the sample space, and each row is associated with a scalar parameter $\theta_{\text{id}(x)}$ representing the probability assigned to that outcome. The table would begin with entries such as (nice!, θ_1), (a cat!, θ_2), (a cute cat!, θ_3), and so forth, extending indefinitely through the infinite set of possible sentences. The natural question that emerges is: how many parameters do we need to fully specify P_X? The answer is immediately apparent: infinitely many.

1.2.6 Limitations of Tabular Representations

The tabular representation approach to specifying discrete distributions, while conceptually simple, suffers from severe practical limitations. In this representation, assigning probability to an outcome $X = x$ amounts to a simple table lookup: $P_X(x) = \theta_{\text{id}(x)}$. The parameters in the tabular representation are statistically independent of one another, meaning that the data provides independent evidence for each parameter. This independence structure, while enabling straightforward statistical inference, is computationally and statistically inefficient. With countably infinite sample spaces, the tabular approach becomes entirely unusable, as we cannot store or estimate infinitely many independent parameters.

1.3 Factorization of Language Model Probabilities

1.3.1 Core Principle of Decomposition

The key insight that resolves the intractability of the direct tabular approach is recognizing that an outcome $w_{1:\ell} \in W^*$ can be viewed as a decomposable structure. Rather than assigning probability to the complete sequence as an atomic unit, we can imagine a procedure by which this structure is derived through a sequence of incremental steps. The probability of any given sequence $w_{1:\ell}$ can then be expressed using probabilities assigned to the individual steps

that jointly derive it. This decomposition, far from being merely a mathematical convenience, corresponds to a natural generative process for creating text.

1.3.2 Generative Procedure: A Concrete Example

To make this abstract principle concrete, consider the sequence $x = \langle \text{He}, \text{went}, \text{to}, \text{the}, \text{store}, \text{EOS} \rangle$, where EOS denotes a special end-of-sequence symbol marking the termination of the sequence. We imagine this sequence as the result of incrementally expanding an initially empty sequence, one symbol at a time, following these steps:

We begin with an empty history $h_1 = \langle\rangle$ and set $i = 1$. Given the current history h_1, we select the first word "He" with some probability, creating the new history $h_2 \leftarrow h_1 \circ \langle \text{He} \rangle = \langle \text{He} \rangle$. Given this new history h_2, we select the second word "went", creating $h_3 \leftarrow h_2 \circ \langle \text{went} \rangle = \langle \text{He}, \text{went} \rangle$. We continue this process: given h_3, we select "to" to create $h_4 = \langle \text{He}, \text{went}, \text{to} \rangle$; given h_4, we select "the" to create $h_5 = \langle \text{He}, \text{went}, \text{to}, \text{the} \rangle$; given h_5, we select "store" to create $h_6 = \langle \text{He}, \text{went}, \text{to}, \text{the}, \text{store} \rangle$. Finally, given h_6, we select the EOS symbol, which terminates the sequence.

1.3.3 Probability Assignment via Chain Rule

Using this generative procedure, we assign probability to the complete sequence $X = x$ by assigning probability to each of these sequential steps, all of which must be executed in order to reproduce x. For the specific example, this yields:

$$
\begin{aligned}
P_X(\langle \text{He}, \text{went}, \text{to}, \text{the}, \text{store} \rangle) = {} & P_{W|H}(\text{He}|\langle\rangle) && (1.1)\\
& \times P_{W|H}(\text{went}|\langle \text{He} \rangle) && (1.2)\\
& \times P_{W|H}(\text{to}|\langle \text{He}, \text{went} \rangle) && (1.3)\\
& \times P_{W|H}(\text{the}|\langle \text{He}, \text{went}, \text{to} \rangle) && (1.4)\\
& \times P_{W|H}(\text{store}|\langle \text{He}, \text{went}, \text{to}, \text{the} \rangle) && (1.5)\\
& \times P_{W|H}(\text{EOS}|\langle \text{He}, \text{went}, \text{to}, \text{the}, \text{store} \rangle) && (1.6)
\end{aligned}
$$

This decomposition specifies how the distribution P_X assigns probability to the text. The key insight is that we express the overall probability as a product of conditional probabilities from distributions of the form $P_{W|H}$, where we traverse the sequence from left to right. Each time we assign probability to a word, we condition on an ordered history of words that precede it, capturing the linguistic intuition that each word's probability depends on the words that came before it.

1.3.4 Autoregressive Factorization Formula

The general form of this factorization is captured by the autoregressive decomposition formula:

$$
P_X(w_{1:\ell}) \triangleq \prod_{i=1}^{\ell} P_{W|H}(w_i|w_{<i}) \tag{1.7}
$$

This equation defines what we call an autoregressive factorization of the probability of a sequence. The notation $w_{<i}$ denotes the prefix of the sequence up to but not including position i, i.e., $\langle w_1, \ldots, w_{i-1} \rangle$. The $\triangleq$ symbol indicates that this is a definition or design choice, not a mathematical derivation from first principles.

This autoregressive factorization is the most general form of language model we will discuss. All specific language model architectures we subsequently encounter can be understood as particular ways of designing the conditional distributions $P_{W|H}$ that appear on the right-hand side

of this equation. The problem of language modelling thus reduces to the problem of designing appropriate conditional probability distributions for $P_{W|H}$.

1.3.5 Generative Story and Sampling

The procedure used to assign probabilities to sequences also directly prescribes a sampler, simulator, or generator—an algorithm for generating outcomes from the language model distribution P_X. This is sometimes called a generative story because it describes a narrative process by which text comes about. The generative story consists of the following algorithm:

Begin with an empty history $h_1 = \langle\rangle$ and set $i = 1$. At each step, condition on the available history h_i and draw a word w_i with probability $P_{W|H}(w_i|h_i)$, then extend the history with this word. If w_i is the special end-of-sequence symbol (EOS), terminate the procedure. Otherwise, increment i and repeat the step of drawing and history extension. This procedure generates sequences sample-by-sample from the language model distribution.

1.3.6 Summary of Factorization

The factorization approach resolves the tractability problem by reframing the language modelling problem. Rather than working with P_X directly—which would require us to specify probabilities for infinitely many variable-length sequences—we re-express it through the chain rule of probability as a product of conditional probabilities $P_{W|H}$. For each given history h, we must be able to prescribe a distribution $P_{W|H=h}$ over the vocabulary. Critically, the vocabulary is finite, so the pmf of $P_{W|H=h}$ is representable by a tractable vector of size equal to the vocabulary. The apparent unbounded complexity arises from the fact that the set of possible histories is unbounded (any sequence of any number of words, so long as it does not end in EOS). Addressing this residual complexity is the subject of subsequent sections.

1.4 Parameterization of Language Models

1.4.1 Distinguishing Factorization from Parameterization

Having introduced factorization, we now turn to parameterization. These two concepts, while related, serve distinct purposes. Factorizing $P_X(x)$ means decomposing this quantity as a product of elementary factors. Using the chain rule, we have shown how to express $P_X(w_{1:\ell}) = \prod_i P_{W|H}(w_i|h)$, decomposing the full sequence probability into conditional probabilities. Parameterization, by contrast, refers to the design of a concrete mechanism to compute $P_{W|H}(w|h)$ for any given choice of history-word pair (h, w).

The distinction is analogous to factorization in arithmetic. The composite number 24 can be factorized into a product of primes: $2 \times 2 \times 2 \times 3 = 2^3 \times 3$. The prime numbers, being "elementary," cannot be further factorized. In language modelling, factorization specifies the structure of the product, while parameterization specifies how to compute each elementary factor.

1.4.2 Overview of Parameterization Approaches

Several major ideas have emerged in the NLP literature for assigning probability to any word $w \in W$ given any history $h \in W^*$. The first approach leverages the relative frequency of the complete history concatenated with the candidate word as observed in a large corpus. The second approach informs probability assignment by the count of the complete history-word pair and the counts of its subsequences in a large corpus, incorporating smoothing and backoff techniques. These first two approaches are grounded in frequentist statistical theory. The third approach uses log-linear models that extract features $\phi(h) \in \mathbb{R}^D$ from the history and

combine them linearly in a log-probability space. The fourth approach employs non-linear models, particularly neural networks, to map from histories directly to the parameters of the probability mass function. This chapter addresses the first two approaches in detail, while subsequent chapters will cover the more sophisticated log-linear and neural network approaches.

1.4.3 Maximum Likelihood Estimation via Relative Frequency

The most straightforward approach to parameterization uses relative frequency. To assign probability to a word w given a history h, such as assigning probability to "store" given the history "he went to the," we use the frequency of the complete sequence "he went to the store" relative to the frequency of "he went to the" followed by any known word in a large corpus.

Formally, for a word w and a history h, the maximum likelihood estimate is:

$$P_{W|H}(w|h)^{\text{MLE}} = \frac{\text{count}_{HW}(h, w)}{\sum_{o \in W} \text{count}_{HW}(h, o)} \tag{1.8}$$

where $\text{count}_{HW}(h, w)$ represents the number of times the history-word pair (h, w) appears in the training corpus, and the denominator sums this count over all possible words in the vocabulary following history h. This estimation corresponds to maximum likelihood estimation for tabular Categorical conditional probability distributions of the form $P_{W|H=h}$.

While this approach is intuitively appealing and statistically principled for observed data, it immediately reveals a fundamental problem.

1.4.4 The Data Sparsity Problem

The data sparsity problem is an unavoidable truth confronting empirical methods. If we have not observed a given history h followed by a certain word w in our training corpus, the relative frequency is zero, assigning zero probability to this previously unseen context-word pair. More severely, if we have not observed the given history h at all, the relative frequency is not even mathematically defined. This situation arises frequently because of the vast combinatorial space of possible histories. The history is fundamentally unbounded—it can be any sequence of words not ending in EOS.

An important philosophical principle must be emphasized here: not observing something in a dataset is emphatically not evidence that the phenomenon is impossible. Rather, it is evidence of data sparsity, a limitation of our training data. This distinction is crucial for principled statistical reasoning. An answer to the data sparsity problem that remained popular for decades involved changing the factorization structure itself, specifically by simplifying the history h to retain only the words closest to the next word.

1.4.5 N-Gram Language Models

The N-Gram language model introduces a simplifying Markov assumption that dramatically reduces the complexity of the conditioning history. Specifically, the N-Gram model assumes that the next word is conditionally independent of all but the $N - 1$ immediately preceding words.

This assumption can be visualized as a hierarchy. Consider the sequence "He went to the store". An autoregressive language model (corresponding to $N = \infty$) would condition on the entire history. A unigram language model ($N = 1$) would condition on no previous words, treating each word as independent. A bigram language model ($N = 2$) would condition on exactly one previous word. A trigram language model ($N = 3$) would condition on exactly two previous words. The hierarchy progresses in increasing order of historical context.

1.4.6 Conditional Independence and Markov Assumptions

The core assumption in N-Gram language models is formally expressed as a conditional independence statement:

$$P_X(w_{1:\ell}) \overset{\text{ind.}}{=} \prod_{i=1}^{\ell} P_{W|H}\left(w_i | \langle w_{i-N+1}, \ldots, w_{i-1} \rangle\right) \tag{1.9}$$

This equation asserts that each word depends only on the $N-1$ words immediately preceding it, not on any earlier words. This is called the Markov assumption of order $N-1$. A crucial observation is that the elementary factors of the N-Gram model all depend on the same fixed number $(N-1)$ of previous words, in contrast to the general autoregressive model where history length is unbounded.

1.4.7 Tabular CPDs for N-Gram Models

Implementing N-Gram language models in practice involves storing relative frequencies of observed N-Grams in a table. Suppose we are designing a trigram language model ($N = 3$) to assign probability to "store" given a long history such as "he went to the". The model first truncates this history to the last $N - 1 = 2$ words, yielding "to the", which has the required fixed length. The model then uses the frequency of the complete trigram "to the store" relative to the frequency of "to the" followed by any known word in the training corpus.

Formally, for a word w, a history h, and N-Gram size N:

$$P_{W|H}(w|h) = \frac{\text{count}_{HW}(\text{last}_{N-1}(h), w)}{\sum_{o \in W} \text{count}_{HW}(\text{last}_{N-1}(h), o)} \tag{1.10}$$

where $\text{last}_{N-1}(h)$ extracts the last $N-1$ tokens from history h. While this approach reduces the space of possible contexts, it still faces the data sparsity problem.

1.4.8 Smoothing Techniques

To address the data sparsity problem, we employ smoothing techniques that reserve probability mass for unseen N-Grams. The fundamental idea is to add a regularization term to both the numerator and denominator of the frequency-based estimate:

$$P_{W|H}(w|h) = \frac{\text{count}_{HW}(h, w) + \alpha(h)}{\sum_{o \in W}(\text{count}_{HW}(h, o) + \alpha(h))} = \frac{\text{count}_{HW}(h, w) + \alpha(h)}{V \cdot \alpha(h) + \text{count}_H(h)} \tag{1.11}$$

The regularization term $\alpha(h)$ is typically a history-independent constant. A particularly common choice is $\alpha(h) = 1$, known as Laplace smoothing or add-one smoothing. With Laplace smoothing, consider a concrete example: if the history "a nice" appears 100 times in the corpus, and the word "rabbit" is in the vocabulary but has never appeared after "a nice", then the smoothed probability would be $P_{W|H}(\text{rabbit}|\langle \text{a}, \text{nice} \rangle) = \frac{1}{V+100}$, a small but non-zero probability.

A practical tip for implementation: when implementing smoothed language models, it is more efficient to store raw counts rather than computing and storing the resulting probability parameters. This is because counts are sparse (many zeros) and benefit from efficient storage, whereas probability parameters are typically dense and require more storage. The probabilities can be computed on-the-fly when needed.

1.4.9 Handling Unknown Words

The treatment of unknown words presents a distinct challenge from data sparsity over seen histories. We must address symbols that are not in the vocabulary at all. A widely-used solution is to augment the vocabulary with a special placeholder symbol, typically denoted UNK (for "unknown"). During training, the model is configured such that any token type encountered that has sufficient frequency to warrant explicit vocabulary membership is retained as-is. However, when processing new text, any previously unseen token is treated as if it were the UNK symbol. This approach is particularly effective when combined with smoothing, as the smoothing procedure will have assigned non-zero probability to UNK following any history, thereby preventing zero-probability assignments to novel text.

1.4.10 Computational and Statistical Limitations

N-Gram language models, while addressing the problem of data sparsity through Markov assumptions, introduce new limitations. The primary advantage is that the order N directly controls the trade-off between model capacity and data efficiency. However, increasing the order comes with exponential computational cost: moving from N-Grams to $(N+1)$-Grams increases the space of possible contexts from V^{N-1} to V^N. The longer the conditioning history, the less likely it is that we have observed it in the training data. Most of the combinatorially possible history-word pairs will never be seen in any finite dataset.

Various techniques have been developed to partially mitigate these limitations. Smoothing techniques were previously discussed. Interpolation involves combining multiple N-Gram models of different orders, leveraging the empirical frequency estimates from smaller-order models to support estimation of larger-order models. Backoff techniques employ a smoothing mechanism that uses lower-order N-Gram estimates when higher-order contexts are not observed.

1.4.11 Fundamental Limitation: Long-Range Dependencies

Despite the various engineering tricks applied to N-Gram models, they suffer from a fundamental limitation rooted in their design: the Markov assumptions that motivate the restriction to fixed-order conditioning are driven by computational convenience alone, not by linguistic necessity. Long-range dependencies, in which the interpretation of a word depends on other words far earlier in the sentence, are ubiquitous in natural languages. N-Gram models are inherently incapable of capturing these dependencies within their local conditioning context.

The resolution to this fundamental limitation requires moving beyond the idea of "storing" probabilities or their related counts. Instead, the key innovation that enables the most powerful language models is learning to predict probability masses using parametric models. These parametric models—first log-linear models, then non-linear neural networks—learn shared representations that can generalize to unseen contexts, thereby capturing long-range dependencies and other complex patterns in language.

1.5 Evaluation of Language Models

Language models must be rigorously evaluated to assess whether they have learned meaningful patterns from data. The evaluation landscape encompasses multiple complementary perspectives: intrinsic evaluation that measures the model's inherent probability assignments, extrinsic evaluation that measures task performance, and statistical evaluation that examines properties of generated samples.

1.5.1 Intrinsic Evaluation: Perplexity

Intrinsic evaluation focuses on assessing the average surprisal, measured as negative log probability, that a language model assigns to held-out test texts. Given a held-out dataset consisting of texts $\{x^{(1)}, \ldots, x^{(S)}\}$, we compute:

$$\frac{1}{S}\sum_{s=1}^{S} \log P_X(x^{(s)}) \tag{1.12}$$

For easier interpretation, this metric is re-expressed in terms of perplexity per token, a measure of the average confusion of the model:

$$\text{Perplexity} = \exp\left(-\frac{1}{T}\sum_{t=1}^{T} \log P_X(w_t)\right) \tag{1.13}$$

where T is the total number of tokens in the test set. The intuition behind perplexity is that it measures the effective vocabulary size that the model's uncertainty spreads over, on average, when predicting the next token. If perplexity per token is 5, for instance, this indicates that on average, across different histories, the model's uncertainty over the next token effectively spreads over approximately 5 candidates from the vocabulary. Lower perplexity indicates a better model.

1.5.2 Extrinsic Evaluation

While intrinsic metrics like perplexity provide valuable insights into model behavior, they do not necessarily correspond to performance on downstream NLP tasks. Extrinsic evaluation involves embedding the language model within a larger system designed for a specific task (such as autocomplete, speech recognition, or machine translation) and measuring the model's contribution to performance on that task. This approach directly assesses the practical utility of the language model.

1.5.3 Statistical Evaluation

A third evaluation perspective examines whether the distributions of linguistic statistics in text generated by the model resemble the distributions in observed training data. Rather than assessing token-level probabilities, this approach looks at aggregate properties of generated samples. For instance, one might examine whether the mean sentence length in generated text matches the mean in observed text, whether frequency distributions of part-of-speech tags match, or whether other linguistic properties exhibit similar distributions. Recent research has explored such statistical evaluation methods in detail.

1.5.4 Critical Considerations in Evaluation

A fundamental principle must be understood when evaluating language models: the statistical model is only as good as the statistical assumptions underlying its design, the estimation procedure used to optimize its parameters, and the data used to fit it. Most statistical assumptions are at best approximations and at worst fundamentally false or insufficient. Any dataset, however large, represents at best a snippet of language production by a limited set of speakers. No dataset is comprehensive enough to fully represent the linguistic behavior of an entire world of speakers, nor is any dataset adequate to represent any single specific group of speakers in its entirety.

It is crucial to maintain a critical perspective on what language models actually learn and what they are incapable of learning. Language models are trained to generate samples that

look like they could have been found in the training data, not to understand language, not to respond appropriately to prompts, not to comply with human values, and not to produce factually correct text. The alignment between any of these latter objectives and the explicit training objective—resembling the training distribution—is contingent and must be explicitly engineered through additional techniques.

1.6 Summary

Language models are probability distributions over sequences of text. The chain rule of probability is the central tool enabling language model design, allowing us to decompose the probability of a complete sequence into a product of conditional probabilities over individual tokens given their histories. Classical N-Gram language models implement this idea through two design choices: a Markov assumption that limits conditioning on only the immediately preceding $N-1$ words, and a tabular parameterization that stores empirical frequency counts of observed N-Grams. However, tabular parameterization proves statistically and computationally inefficient. The modern approach parameterizes conditional probability distributions using neural networks, which enable learning of shared representations and effective capture of long-range dependencies. Rigorous evaluation of language models requires assessing probability assignments through intrinsic metrics, task performance through extrinsic evaluation, and consistency of statistical properties through statistical evaluation of samples.

Chapter 2

Sequence Labelling

The field of natural language processing has historically treated words in language models as atomic, unrelated symbols. This approach, while mathematically tractable, leads to two significant limitations. First, it requires large tabular conditional probability distributions (CPDs) to store probabilities for every possible conditional outcome. Second, it suffers from severe statistical inefficiency due to data sparsity, where linguistically related outcomes cannot share statistical evidence because they are treated as completely independent entities.

Sequence labelling represents a fundamental shift in how we model linguistic phenomena. Rather than treating words as isolated atomic symbols, we introduce word categories—structured classes that capture linguistic relatedness based on semantic, formal, or distributional criteria. This allows us to learn patterns that generalize across all words sharing a given property, dramatically reducing both model complexity and data requirements.

2.1 Word Categories and Classification

2.1.1 Organizing Words into Classes

Words can be organized into categories using three complementary perspectives. From a semantic perspective, we classify words based on what they refer to or represent. For instance, nouns typically refer to people, places, or things, while verbs denote actions or states. The formal perspective examines the structure and morphology of words themselves, recognizing that suffixes like *-ly* in English reliably convert adjectives to adverbs, *-tion* transforms verbs to nouns, and patterns like *-ness*, *-ity*, and *-ance* indicate nominalization. Finally, the distributional perspective considers the contexts in which words can occur. In English, adjectives consistently appear before nouns, and this contextual behavior provides strong evidence for word categorization.

2.1.2 Parts-of-Speech Tagging

Parts-of-Speech (POS) tagging focuses on categorizing words primarily by their syntactic function, representing one of the most well-studied sequence labelling tasks in NLP. The granularity of POS categories varies depending on the corpus and annotation scheme. The Penn Treebank uses 45 distinct categories, while the earlier Brown corpus employed 87 categories. To promote cross-lingual compatibility and standardization, the Universal POS tagset simplifies these distinctions into 12 major categories: ADJ (adjectives), ADP (prepositions and postpositions), ADV (adverbs), CONJ (conjunctions), DET (determiners and articles), NOUN (nouns), NUM (numerals), PRON (pronouns), PRT (particles), PUNCT (punctuation marks), VERB (verbs), and X (miscellaneous items such as abbreviations or foreign words).

An example of POS-tagged text in Penn Treebank style is as follows: `The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.`. Here,

each word is paired with its corresponding tag, enabling the model to process both lexical and syntactic information simultaneously.

2.2 Hidden Markov Models

2.2.1 Formal Framework and Definitions

The Hidden Markov Model (HMM) provides a principled probabilistic framework for sequence labelling. We define the random variable W as a single word, where any outcome w is a symbol from a vocabulary $\mathcal{W}$ of size V. Similarly, C represents a POS tag, where any outcome c belongs to a tagset $\mathcal{C}$ of size K. For sequences, we define $X = \langle W_1, \ldots, W_L \rangle$ as a random word sequence and $Y = \langle C_1, \ldots, C_L \rangle$ as a random tag sequence. An outcome of these random sequences is written as $w_{1:\ell}$ and $c_{1:\ell}$ respectively, denoting sequences of length ℓ.

The fundamental task is to design a mechanism that assigns probability $P_{XY}(w_{1:\ell}, c_{1:\ell})$ to any outcome $(w_{1:\ell}, c_{1:\ell}) \in \mathcal{W}^* \times \mathcal{C}^*$. This requires three key steps: factorizing the joint probability distribution using chain rules and conditional independence assumptions, parameterizing the resulting elementary factors using tabular categorical distributions, and estimating these parameters from annotated training data using maximum likelihood estimation.

2.2.2 Factorization via Conditional Independence

The HMM factorization rests on two critical conditional independence assumptions. First, the word W_i is conditionally independent of all positions except position i, given the tag C_i at that position. Second, the tag C_i is conditionally independent of all non-adjacent tags in the sequence, given only the immediately preceding tag C_{i-1}. These assumptions lead to a linear-chain structure that is computationally tractable.

The joint probability distribution factorizes according to equation:

$$P_{XY}(w_{1:\ell}, c_{1:\ell}) = \prod_{i=1}^{\ell} P_{C|C_{\text{prev}}}(c_i|c_{i-1}) \cdot P_{W|C}(w_i|c_i) \tag{2.1}$$

Here, $P_{C|C_{\text{prev}}}(c_i|c_{i-1})$ represents the transition probability—the probability of generating tag c_i given that the previous tag was c_{i-1}. The term $P_{W|C}(w_i|c_i)$ denotes the emission probability—the probability of generating word w_i given that the current tag is c_i.

The sequences are padded with special boundary symbols: a Beginning-of-Sequence (BoS) symbol at the start of the tag sequence to provide context for the first transition, and an End-of-Sequence (EoS) symbol at the end of both sequences. This padding ensures that all steps of the generative process have well-defined conditioning contexts.

2.2.3 Generative Story

The HMM can be understood through the following generative process. We initialize with $X = \langle W_0 = \text{BoS} \rangle$ and $Y = \langle C_0 = \text{BoS} \rangle$, setting the step counter to $i = 1$. At each iteration, we condition on the previous tag c_{i-1} and draw a new tag c_i according to the transition distribution $P_{C|C_{\text{prev}}}(c_i|c_{i-1})$, extending the tag sequence Y with this new tag. We then condition on the current tag c_i and draw a word w_i from the emission distribution $P_{W|C}(w_i|c_i)$, extending the word sequence X. If the generated word w_i is the special end-of-sequence token (EoS), the process terminates. Otherwise, we increment i and repeat from the tag generation step. This process naturally generates both a word sequence and its corresponding tag sequence, capturing the joint distribution over labelled text.

2.2.4 Tabular Parameterization

The elementary factors in the HMM factorization are parameterized using categorical distributions. The transition distribution given a previous tag r is:

$$C|C_{\text{prev}} = r \sim \text{Categorical}(\boldsymbol{\lambda}^{(r)}_{1:K}) \tag{2.2}$$

where $\boldsymbol{\lambda}^{(r)}_{1:K}$ is a K-dimensional probability vector. This gives us the transition probability:

$$P_{C|C_{\text{prev}}}(c|r) = \lambda^{(r)}_c \tag{2.3}$$

Similarly, the emission distribution given a tag c is:

$$W|C = c \sim \text{Categorical}(\boldsymbol{\theta}^{(c)}_{1:V}) \tag{2.4}$$

yielding the emission probability:

$$P_{W|C}(w|c) = \theta^{(c)}_w \tag{2.5}$$

The complete probability mass function for a tagged sequence is therefore:

$$P_{XY}(w_{1:\ell}, c_{1:\ell}) = \prod_{i=1}^{\ell} \lambda^{(c_{i-1})}_{c_i} \times \theta^{(c_i)}_{w_i} \tag{2.6}$$

For example, the sequence "a/DT nice/JJ dog/NN" receives probability:

$$\lambda^{(\text{BoS})}_{\text{DT}} \cdot \theta^{(\text{DT})}_{\text{a}} \times \lambda^{(\text{DT})}_{\text{JJ}} \cdot \theta^{(\text{JJ})}_{\text{nice}} \times \lambda^{(\text{JJ})}_{\text{NN}} \cdot \theta^{(\text{NN})}_{\text{dog}} \times \lambda^{(\text{NN})}_{\text{EoS}} \cdot \theta^{(\text{EoS})}_{\text{EoS}} \tag{2.7}$$

2.2.5 Parameter Estimation via Maximum Likelihood

Given a dataset $\mathcal{D}$ of texts annotated with POS tags, we estimate model parameters using maximum likelihood estimation. The MLE for transition probabilities is:

$$\lambda^{(r)}_c\Big|_{\text{MLE}} = \frac{\text{count}(r, c)}{\sum_{k=1}^{K} \text{count}(r, k)} = \frac{\text{count}(r, c)}{\text{count}(r)} \tag{2.8}$$

where $\text{count}(r, c)$ represents the number of times tag c immediately follows tag r in the training data. Similarly, the MLE for emission probabilities is:

$$\theta^{(c)}_w\Big|_{\text{MLE}} = \frac{\text{count}(c, w)}{\sum_{o=1}^{V} \text{count}(c, o)} = \frac{\text{count}(c, w)}{\text{count}(c)} \tag{2.9}$$

where $\text{count}(c, w)$ is the frequency with which word w receives tag c in the training corpus.

2.2.6 Data Sparsity Considerations

While the HMM suffers less from data sparsity than n-gram language models, the problem persists. Unseen transitions or emissions can still occur in test data. However, the reduction in sparsity is substantial because contextual information is compressed through the POS tag dimension. Rather than maintaining statistics over all possible n-grams of words (which scales as V^{n-1}), the HMM maintains statistics over sequences of tags (which scale as K^{n-1}). Since the number of tags K is typically much smaller than the vocabulary size V, this represents a significant reduction in the parameter space and improves statistical efficiency.

2.2.7 Limitations of HMM Assumptions

Despite their utility, HMMs make strong conditional independence assumptions that oversimplify linguistic phenomena. Consider the word "PLAN", which can function as either a verb in the sentence "I read that the government plans to..." or as a noun in "I read the government plans to...". The older context ("read that" versus "read the") affects the correct analysis, but the HMM only conditions on the immediately previous tag. Similarly, for the pronoun "HER", determining whether it functions as a determiner ("I read her book") or as an object pronoun ("I saw her there") depends on the semantic properties of the verb, which the model cannot access. Agreement features like "a cat" versus "a cats" sometimes cannot be properly analyzed without looking beyond the immediate neighbors. Finally, analyzing words like "LIKE" sometimes requires looking ahead beyond the current position, which the HMM's left-to-right generation process does not naturally support.

Possible improvements include relaxing independence assumptions by incorporating trigram transitions that condition C_i on both C_{i-2} and C_{i-1}, allowing bigram emissions where W_i depends on W_{i-1}, or creating other dependencies such as having the first word depend on the previous tag. However, these extensions increase the size of tabular CPDs and exacerbate sparsity, leading to new challenges.

2.3 Evaluation Methods

2.3.1 Tagging Performance

The primary evaluation task for sequence labelling is to predict the tag sequence for novel text. We solve the following optimization problem:

$$\hat{c}_{1:\ell} = \arg\max_{c_{1:\ell} \in \mathcal{C}^\ell} P_{Y|X}(c_{1:\ell}|w_{1:\ell}) \tag{2.10}$$

The predicted tag sequence is compared to a human-annotated reference sequence, typically treating this as a per-position multiclass classification problem. Performance is evaluated using per-POS F1 scores, with results reported as either macro-averaged or weighted-averaged F1 across all tag classes.

2.3.2 Language Modelling Performance

Alternatively, we can use the HMM as a language model, assigning probability to untagged text through marginalization:

$$P_X(w_{1:\ell}) = \sum_{c_{1:\ell} \in \mathcal{C}^\ell} P_{XY}(w_{1:\ell}, c_{1:\ell}) \tag{2.11}$$

This evaluates the model's ability to assign probability mass to observed text, measured via perplexity on a held-out test set. The perplexity indicates how well the model generalizes to unseen data.

2.3.3 Computational Complexity of Search

Naively solving equation (2.10) requires enumerating all K^ℓ possible tag sequences, assessing the probability of each, sorting by probability, and selecting the maximum. This is computationally intractable for even moderate sequence lengths.

However, the conditional independences in the HMM structure enable efficient computation. Changing the POS tag at position i affects only three probability terms: the emission probability $P_{W|C}(w_i|c_i)$ and two transition probabilities $P_{C|C_{\text{prev}}}(c_i|c_{i-1})$ and $P_{C|C_{\text{prev}}}(c_{i+1}|c_i)$. This locality

of dependencies allows solving both the mode-seeking problem (via the Viterbi algorithm) and the marginalization problem (via the Forward algorithm) incrementally from left to right in time complexity $O(L \times K^2)$.

2.4 Sequence Labelling Tasks

2.4.1 Part-of-Speech Tagging

POS tagging remains the canonical sequence labelling application. Given a text, the goal is to assign the most likely POS tag to each word. Unlike language modelling objectives that assign probability to the text itself, in POS tagging we treat the word sequence as given and focus on predicting the corresponding tag sequence.

2.4.2 Named Entity Recognition

Named Entity Recognition (NER) extends sequence labelling to semantic tasks. Rather than classifying words by syntactic function, NER identifies and categorizes named entities—proper nouns referring to specific types of entities such as persons, organizations, locations, or other domain-specific categories. The input word sequence is given, and the goal is to identify spans of consecutive tokens that form named entities and assign appropriate type labels to these spans.

2.4.3 Chunking as Sequence Labelling

NER and similar tasks can be framed within the sequence labelling paradigm by labelling individual tokens as either inside or outside named entity spans. Various annotation schemes encode such information, including the BIO scheme (Begin, Inside, Outside), where each token is marked as the beginning of an entity, inside an entity, or outside any entity.

2.4.4 Technical Limitations

A fundamental limitation of HMMs for sequence labelling is their reliance on text generation. To maintain computational tractability of marginalisation and mode-seeking, HMMs restrict how tags can interact with words. Allowing tags to depend on words beyond the current position would dramatically increase the difficulty of inference tasks. Additionally, using tabular CPDs with such dependencies would result in extremely sparse parameter tables, as the conditioning context becomes exponentially larger.

This constraint limits the linguistic context accessible to the model. Phenomena like unseen words (proper names, acronyms, inflected forms) are common in practice, yet their likely interpretations are often identifiable from fine-grained surface features including capitalization patterns, morphological structure (prefixes, suffixes, roots), and the surrounding word context (e.g., a window of surrounding words). HMMs cannot efficiently leverage such features within their framework.

2.5 Local Log-Linear Models

2.5.1 Motivation for Conditional Modelling

Rather than generating both text and its labelling through a joint distribution, an alternative approach directly models the conditional distribution of tags given text:

$$\hat{c}_{1:\ell} = \arg\max_{c_{1:\ell} \in \mathcal{C}^\ell} P_{Y|X}(c_{1:\ell} | w_{1:\ell}) \tag{2.12}$$

This framing abandons the generative story of the HMM and treats the word sequence as a predictor from which to forecast the tag sequence. By focusing directly on the conditional distribution rather than the joint, we can incorporate arbitrary features of the text without the computational constraints imposed by generative modelling.

2.5.2 Zero-Order Conditional Model

We begin with a simplified zero-order Markov assumption for the tag sequence, where each tag is conditionally independent of all other tags given the text and position:

$$C_i \perp C_{j \neq i} | X = x, I = i \tag{2.13}$$

This yields a factorization:

$$P_{Y|X}(c_{1:\ell}|w_{1:\ell}) = \prod_{i=1}^{\ell} P_{C|XI}(c_i|w_{1:\ell}, i) \tag{2.14}$$

Unlike the HMM, the conditioning context here is now the entire text $w_{1:\ell}$ along with the position i. This conditioning context is high-dimensional and variable-length, making a tabular parameterization infeasible—we cannot pre-compute and store conditional probabilities for every possible text and position combination.

2.5.3 Feature Functions and Log-Linear Parameterization

The solution is to learn to predict conditional probabilities from a learned representation of the conditioning context. We define a feature function $\boldsymbol{\phi}(w_{1:\ell}, i) \in \mathbb{R}^D$ that represents the i-th position of text $w_{1:\ell}$ as a D-dimensional vector. Features might encode the current word (via indicator features such as `word:to`), the preceding word (`before:went`), the following word (`after:the`), normalized position (`position`=3/5), or other aspects of context. If the vocabulary has size V and each of these three templates is binary-valued, the feature space has dimensionality $D = 3V$.

We map the conditioning context to a probability distribution over tags via three operations. First, we map the conditioning context to the real coordinate space by extracting the feature vector. Second, we linearly transform this vector to K scores, one for each possible tag. Third, we project these scores to the probability simplex using the softmax function:

$$f(w_{1:\ell}, i; \boldsymbol{\theta}) = \text{softmax}(\mathbf{W}\boldsymbol{\phi}(w_{1:\ell}, i) + \mathbf{b}) \tag{2.15}$$

Here, $\boldsymbol{\theta} = \{\mathbf{W} \in \mathbb{R}^{K \times D}, \mathbf{b} \in \mathbb{R}^K\}$ contains the learnable parameters: a weight matrix $\mathbf{W}$ and a bias vector $\mathbf{b}$.

2.5.4 Zero-Order Model Probability Mass Function

Under this parameterization, the conditional probability mass function becomes:

$$P_{Y|X}(c_{1:\ell}|w_{1:\ell}) = \prod_{i=1}^{\ell} P_{C|XI}(c_i|w_{1:\ell}, i) = \prod_{i=1}^{\ell} [f(w_{1:\ell}, i; \boldsymbol{\theta})]_{c_i} \tag{2.16}$$

The notation $[\mathbf{v}]_i$ denotes the i-th coordinate of vector $\mathbf{v}$. Notably, this factorization amounts to designing a single "text classifier"-like component that predicts a distribution over K tags, then reusing this component independently at each position of the sequence.

2.5.5 Parameter Estimation via Gradient-Based Optimization

Given a training corpus $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ of N labelled sequences, where the n-th sequence has length ℓ_n, we assess the log-likelihood:

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log P_{Y|X}(\mathbf{y}_n|\mathbf{x}_n) = \sum_{n=1}^{N} \sum_{i=1}^{\ell_n} \log P_{C|XI}(y_{n,i}|\mathbf{x}_n, i) = \sum_{n=1}^{N} \sum_{i=1}^{\ell_n} \log[f(\mathbf{x}_n, i; \boldsymbol{\theta})]_{y_{n,i}} \tag{2.17}$$

Unlike HMM parameter estimation, which has a closed-form solution, this optimization problem has no analytical solution. Instead, we approximately solve it via iterative gradient-based optimization:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \gamma \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) \tag{2.18}$$

where γ is a learning rate. We typically employ automatic differentiation (autodiff) to compute gradients and use stochastic optimization algorithms such as stochastic gradient descent, Adam, or other modern optimizers.

2.5.6 First-Order Conditional Model

To capture dependencies between adjacent tags, we can make a first-order Markov assumption:

$$C_i \perp C_{j \notin \{i-1, i\}} | X = x, I = i, C_{\text{prev}} = r \tag{2.19}$$

This factorizes as:

$$P_{Y|X}(c_{1:\ell}|w_{1:\ell}) = \prod_{i=1}^{\ell} P_{C|XIC_{\text{prev}}}(c_i|w_{1:\ell}, i, c_{i-1}) \tag{2.20}$$

The feature function now has access to the previous tag, allowing us to include templates such as **??**. For example, if a position receives the previous tag Verb, this information can be encoded in the feature vector. Compared to the zero-order feature function, the first-order model introduces K additional feature coordinates corresponding to the K possible previous tags.

2.5.7 Limitations of Zero-Order Models

The zero-order conditional model has important limitations. First, it cannot be used as a language model because it is not derived from a joint probability distribution over the space of labelled text—it only defines a conditional distribution over tag sequences given text. Second, finding the optimal tag sequence can be performed by solving ℓ independent classification problems, since each position's tag is independent of all others given the text. While this independence simplifies computation, it misses important tag-to-tag dependencies that often exist in language.

2.5.8 Conditional Chain Rule Factorization

For more expressive models, we can apply the chain rule directly to the conditional distribution:

$$P_{Y|X}(c_{1:\ell}|w_{1:\ell}) = \prod_{i=1}^{\ell} P_{C|XH}(c_i|w_{1:\ell}, c_{<i}) \tag{2.21}$$

Here, the conditioning includes the entire history $c_{<i}$ of previous tags. This allows the model to capture complex tag interdependencies. However, designing an effective feature function

for this factorization is non-trivial. The challenge is that encoding the complex dependencies in the tag history using indicator-based features exponentially increases the feature space dimensionality with history length. Modern neural approaches address this by learning implicit representations that capture such dependencies efficiently.

2.5.9 Advantages and Challenges of Log-Linear Models

Log-linear models offer substantial advantages over HMMs. First, they enable the use of richer context by incorporating entire word sequences rather than single words, accessing word-internal features such as capitalization and morphology, and combining multiple information sources flexibly. Second, they achieve superior statistical efficiency compared to tabular CPDs: model size depends on the dimensionality of learned representations, not on the number of possible condition-outcome pairs. Third, they have been successfully applied to diverse sequence labelling tasks including POS tagging, NER, and semantic role labelling.

However, designing effective log-linear models requires careful consideration. Good feature functions demand sufficient intuition about task-relevant properties. Additionally, effective feature spaces often have very high dimensionality, requiring careful engineering. Modern approaches use neural networks to learn feature representations automatically, circumventing much of this manual engineering.

2.6 Summary and Extensions

2.6.1 Key Concepts

Hidden Markov Models combine generative modelling with classification, enabling their use either as sequence taggers or as language models. HMMs achieve computational tractability through strong factorisation assumptions and are parameterised using tabular categorical distributions. However, these choices lead to statistical inefficiency from data sparsity and limit the model's ability to incorporate rich linguistic context.

Log-linear models provide an alternative parameterisation of categorical distributions, maintaining flexibility while enabling gradient-based learning. Local log-linear models directly model conditional distributions of tags given text, facilitating the incorporation of complex features without the computational constraints of generative modelling. These approaches power practical sequence labelling systems for diverse NLP tasks.

2.6.2 Further Extensions

Two major extensions of these ideas exist. First, global probabilistic models such as Conditional Random Fields (CRFs) apply log-linear parameterisation globally to entire sequences while maintaining the ability to model tag-to-tag dependencies, providing a middle ground between local independence and fully sequential dependencies. Second, neural parameterisation enables the learning of feature representations automatically from data, reducing manual feature engineering. These extensions form the foundation of modern sequence labelling systems and deep learning approaches to NLP.

Chapter 3

Modelling Syntactic Structure in Natural Language Processing

3.1 Introduction to Syntactic Modeling

The study of syntax in natural language processing represents a fundamental shift from models that ignore the structured nature of language. Throughout the history of computational linguistics, we have evolved from simple representations that discard word order entirely to increasingly sophisticated models that capture the hierarchical organization of linguistic constituents. To understand this progression, we must first recognize what previous modeling approaches have accomplished and where they fall short.

The bag-of-words approach, exemplified by naive Bayes classifiers, operates by completely ignoring word order. While this simplification enables efficient computation, it discards crucial linguistic information that is essential for understanding meaning and structure. Markov models, such as bigram language models, represent an improvement by memorizing short observed phrases and maintaining localized dependencies. However, they remain limited in their ability to model the broader structural patterns that characterize natural language. Hidden Markov Models (HMMs) advance this further by capturing shallow syntactic patterns through the use of adjacent word classes, introducing the notion of part-of-speech categories as latent variables. Nevertheless, HMMs do not model semantic dependencies amongst words, as they treat each word as conditionally independent given only the previous word class. These progressively more sophisticated models prepare the conceptual ground for understanding why we need explicit models of syntactic structure.

3.2 From Words to Phrases: Categories and Substitutability

3.2.1 Part-of-Speech and Phrasal Categories

The foundation of syntactic modeling rests on a fundamental principle of linguistic abstraction: words can be grouped into categories based on their distributional properties. A part-of-speech (POS) category captures the fact that certain words are substitutable in the same syntactic contexts. For instance, in the sentence "I saw a [ADJ] cat," the adjective slot can be filled with words such as "red," "large," or "angry" without violating the grammatical structure of the sentence. This substitutability principle extends beyond individual words to larger linguistic units.

Just as we abstract from individual words to their part-of-speech categories, we can perform a similar abstraction at the phrasal level. Phrasal categories recognize that groups of words can function as coherent units and exhibit substitutability in higher-level syntactic contexts. For example, in the sentence "[NP ...] sleep soundly," the noun phrase slot can be filled with

various noun phrases such as "the cat," "many dogs," or "that sleeping creature," without disrupting the sentence structure. Common phrasal categories include noun phrases (NP), verb phrases (VP), and prepositional phrases (PP), each representing distinct types of syntactic units in English.

3.2.2 The Structure of Noun Phrases and Verb Phrases

The structure of phrases in English follows relatively consistent patterns that reflect the central role of a lexical head around which other elements organize themselves. In English noun phrases, the typical structure begins with an optional determiner, followed by zero or more adjectives, followed by the noun head, and concluded with zero or more prepositional phrases or relative clauses. This pattern can be schematically represented as $(Det)\ Adj^*\ Noun\ (PP|\text{RelClause})^*$. A concrete example instantiating this pattern would be the noun phrase "the angry duck that tried to bite me," where "the" serves as the determiner, "angry" as the adjective, "duck" as the noun head, and "that tried to bite me" as the relative clause.

Verb phrases follow a parallel organizational principle, with the verb as the central head element. The typical verb phrase structure in English is $(Aux)\ Adv^*\ Verb\ Arg^*\ Adjunct^*$, where the auxiliary is optional, adverbs may appear before the verb, arguments appear after the verb, and adjuncts provide additional modification. An example of this pattern is the verb phrase "would usually eat pasta for dinner," where "would" is the auxiliary, "usually" is the adverb, "eat" is the verb, "pasta" is the argument (direct object), and "for dinner" is the adjunct (prepositional phrase indicating purpose or time).

3.3 Constituency and Hierarchical Structure

3.3.1 The Concept of Syntactic Constituency

Syntactic constituency is the fundamental principle that groups of words can function as single linguistic units, termed constituents, that behave cohesively in syntactic operations. One of the strongest pieces of evidence for the psychological and linguistic reality of constituents comes from their distributional properties: constituents tend to appear in similar syntactic environments. For example, noun phrases characteristically appear before verbs in declarative English sentences, a distributional property that holds regardless of the internal complexity of the noun phrase itself. The sequence "the cat" and the sequence "the very large and fluffy cat that belonged to my neighbor" both function identically in the structure "[NP ...] sleeps," demonstrating that constituents are not bound by their size or internal complexity but rather by their functional role in larger structures.

3.3.2 Hierarchical Nesting of Constituents

A crucial aspect of syntactic structure is that constituents do not exist in isolation but rather embed hierarchically within other constituents, creating tree-like structures. Consider a simple sentence such as "the dog runs." This sentence consists of a noun phrase (NP) containing the determiner "the" and noun "dog," combined with a verb phrase (VP) containing the verb "runs." The entire structure is dominated by the sentence node (S). More complex examples demonstrate deeper nesting: a sentence like "dogs and cats run" contains a verb phrase (VP) headed by "run" that dominates a noun phrase consisting of two coordinated noun phrases ("dogs" and "cats" conjoined by the coordinator "and"), each of which contains a single noun. This hierarchical organization is not merely a notational convenience but reflects the actual structure of grammatical dependencies and semantic relationships in natural language.

3.4 Context-Free Grammars

3.4.1 Formal Definition and Components

A Context-Free Grammar (CFG) is a formal system that provides a mechanism for rewriting strings through the application of grammar rules. Unlike the probabilistic models discussed previously, CFGs operate on a purely symbolic, non-probabilistic foundation, serving as the backbone upon which probabilistic extensions are built. A CFG consists of two types of symbols that work in concert to generate or recognize strings.

Terminal symbols, also called constants, represent the actual words or tokens that appear in sentences. These are the base elements that cannot be further rewritten and correspond to the vocabulary of the language being modeled. Nonterminal symbols, also called variables, represent abstract syntactic categories such as parts of speech and phrasal categories. Nonterminals serve as intermediate representations that can be rewritten into other sequences of terminals and nonterminals. Crucially, the sets of terminals and nonterminals must be disjoint; no symbol can belong to both categories simultaneously.

The rewriting rules of a CFG take the form $X \to \beta$, where X is a nonterminal symbol and β is any string consisting of terminal and nonterminal symbols. These rules specify how a given nonterminal can be expanded or replaced. The distinguishing characteristic that makes a grammar "context-free" is that the expansion of a nonterminal depends only on the nonterminal itself and not on the surrounding context in which it appears.

Formally, a context-free grammar is defined as a tuple $\langle \Sigma, V, S, R \rangle$, where Σ is a finite set of terminal symbols, V is a finite set of nonterminal symbols with a distinguished start symbol $S \in V$ such that $V \cap \Sigma = \emptyset$, and R is a finite set of rewriting rules of the form $X \to \beta$ with $X \in V$ and $\beta \in (\Sigma \cup V)^*$.

3.4.2 Concrete Example of a Context-Free Grammar

To ground these formal concepts, consider a concrete example grammar designed to parse sentences with the structure "I eat pizza with anchovies." This grammar specifies nonterminals S, NP, VP, PP, $Pron$, N, V, and P, and terminals "I," "eat," "pizza," "with," and "anchovies." The grammar rules are:

$$
\begin{aligned}
S &\to NP\,VP \\
NP &\to NP\,PP \\
VP &\to VP\,PP \\
VP &\to VP\,NP \\
PP &\to P\,NP \\
VP &\to V \\
NP &\to Pron \\
NP &\to N \\
V &\to \text{eat} \\
Pron &\to \text{I} \\
P &\to \text{with} \\
N &\to \text{pizza} \\
N &\to \text{anchovies}
\end{aligned}
$$

This grammar permits multiple derivations for the same sentence, reflecting genuine structural ambiguity in natural language. The sentence "I eat pizza with anchovies" can be analyzed

as having the prepositional phrase "with anchovies" modify either the noun "pizza" or the verb "eat," corresponding to different attachment sites in the parse tree. Such ambiguities are a fundamental characteristic of natural language and any model of syntax must confront them.

3.4.3 Rule Arity and Chomsky Normal Form

The arity of a rule is defined as the length of the right-hand side (RHS) of the rule, specifying how many symbols appear on the right side of the arrow. A unary rule has the form $A \to B$, containing a single symbol on the right side. A binary rule has the form $X \to B\,C$, containing exactly two symbols. An n-ary rule has the form $X \to X_1, \ldots, X_n$, containing n symbols. The arity of a grammar is defined as the maximum arity among all rules in that grammar.

An important theorem in formal language theory establishes that any context-free grammar, regardless of its original arity, can be converted into an equivalent grammar in Chomsky Normal Form (CNF) that has arity exactly two. This transformation is significant for computational purposes, as it simplifies the design of parsing algorithms and reduces the branching factor in the parse forest. The conversion process involves introducing new nonterminals to break down rules of higher arity into sequences of binary rules, and handling unit rules (unary rules whose right side is a nonterminal) through rule composition or elimination techniques.

3.4.4 Derivations and String Generation

A derivation is a sequence of strings that demonstrates how a string can be generated from the start symbol through repeated application of grammar rules. The derivation process begins with the initial string consisting solely of the start symbol $\langle S \rangle$. At each step of the derivation, the leftmost nonterminal in the current string is selected and replaced by the right-hand side of some applicable rule. This process continues until all nonterminals have been replaced by terminals, leaving only a terminal string.

To denote that a string $w_1 \cdots w_\ell$ can be derived from the start symbol S through zero or more rule applications, we write $S \overset{*}{\Rightarrow} w_1 \cdots w_\ell$. Individual derivation steps are denoted using a single arrow, $\Rightarrow$, while the multi-step derivation uses the double arrow with asterisk. When the specific derivation process is of interest, we can denote it explicitly as $S \overset{\delta}{\Rightarrow} w_1 \cdots w_\ell$, where δ represents the sequence of rule applications constituting the derivation.

To illustrate the derivation process concretely, consider deriving the sentence "the dog barks" using a simple grammar with appropriate rules. The derivation proceeds as follows:

$$
\begin{aligned}
\langle S \rangle &\xRightarrow{S \to NP\,VP} \langle NP\,VP \rangle \\
&\xRightarrow{NP \to Det\,N} \langle Det\,N\,VP \rangle \\
&\xRightarrow{Det \to \text{the}} \langle \text{the}\,N\,VP \rangle \\
&\xRightarrow{N \to \text{dog}} \langle \text{the dog}\,VP \rangle \\
&\xRightarrow{VP \to V} \langle \text{the dog}\,V \rangle \\
&\xRightarrow{V \to \text{barks}} \langle \text{the dog barks} \rangle
\end{aligned}
$$

At each derivation step, we select the leftmost nonterminal and apply a rule that rewrites it. The sequence of rule applications encodes the structural analysis imposed by the grammar on the string.

3.5 Probabilistic Context-Free Grammars

3.5.1 Motivation and Basic Definition

While context-free grammars provide a principled framework for modeling syntactic structure, they lack the ability to express preferences or disambiguate among multiple valid analyses. A sentence may have multiple valid derivations under a given CFG, each yielding a different syntactic analysis. Without probabilities, the grammar provides no principled mechanism for preferring one analysis over another. Probabilistic Context-Free Grammars (PCFGs) address this limitation by assigning probability distributions over the space of all possible derivations supported by a CFG, including their yields (the terminal strings that result from the derivations).

3.5.2 Factorization via the Chain Rule and Markov Assumption

The key insight underlying probabilistic modeling of derivations is that we can factorize the probability of a complete derivation as a product of probabilities of individual rule applications. A random derivation is a sequence $D = \langle R_1, \ldots, R_M \rangle$ consisting of M random rule applications. Each rule application is a pair (N, S) where N is a nonterminal (the left-hand side) and S is a string of terminals and nonterminals (the right-hand side).

Using the chain rule of probability, we can write the probability of a derivation as:

$$P_D(r_{1:m}) = \prod_{j=1}^{m} P_R(r_j \mid r_{<j}) \tag{3.1}$$

where $r_{<j}$ denotes the history of rule applications prior to step j. However, this factorization potentially requires computing conditional probabilities over an exponentially large space of possible histories. PCFGs leverage a critical simplifying assumption: the Markov assumption. Under this assumption, each rule application depends only on the left-hand side nonterminal of that rule, not on the entire history of prior derivations. This assumption is justified by the "context-free" nature of the grammar: the applicability and probability of rewriting a nonterminal do not depend on the surrounding context.

Under the Markov assumption, the factorization simplifies to:

$$P_D(r_{1:m}) = \prod_{j=1}^{m} P_S \mid N(r_j \mid v_j) = \prod_{j=1}^{m} P_S \mid N(\beta_j \mid v_j) \tag{3.2}$$

where v_j is the left-hand side nonterminal of the j-th rule and β_j is the right-hand side string. The notation $P_{S|N}$ denotes the conditional probability of a right-hand side string given a nonterminal, making explicit that we are conditioning only on the nonterminal being rewritten, not on any broader context.

As a technical convention, we treat every derivation as implicitly beginning with an initial rule $r_0 = \text{BoS} \to S$, where BoS stands for "Beginning of Sentence." This rule introduces the start symbol and is treated as having probability 1 (or alternatively, is omitted from the factorization as it contributes a factor of 1).

3.5.3 Generative Process

The probabilistic model underlying a PCFG can be expressed as a generative process that produces both a derivation and its corresponding yield (the terminal string). The generative process proceeds as follows:

First, initialize the derivation to $D = \langle S \rangle$, containing only the start symbol. Next, check whether all symbols in the current derivation are terminals. If so, the process terminates and

we have a complete string. If not, identify the leftmost nonterminal symbol ν in the derivation. Condition on this nonterminal and draw a right-hand side string β from the categorical distribution $P_{S|N}(\beta \mid \nu)$. Replace the nonterminal ν in the derivation with the string β and return to the second step.

This process corresponds to a depth-first, left-to-right expansion of nonterminals. The specification of left-to-right processing is important for ensuring a well-defined sequential process, though the mathematical properties of the resulting distribution do not depend on this particular processing order if we use the chain rule correctly.

3.5.4 Parametrization and Categorical Distributions

To operationalize the probabilistic model, each nonterminal must be associated with a categorical probability distribution that specifies the probability of each possible right-hand side string. If a nonterminal ν can be rewritten in K different ways, we associate with ν a K-dimensional categorical distribution. Each dimension corresponds to one possible right-hand side, and the parameters are positive values that sum to 1.

For example, if the start symbol S can be rewritten as either "NP VP" or "VP" (ignoring the realism of this grammar), we would specify:

$$P_{S|N=S} \sim \text{Categorical}(\theta_{S\to NP\,VP}, \theta_{S\to VP}) \tag{3.3}$$

where $\theta_{S\to NP\,VP}$ and $\theta_{S\to VP}$ are parameters such that $\theta_{S\to NP\,VP} + \theta_{S\to VP} = 1$ and both parameters are non-negative. Similarly, if the nonterminal N can be rewritten as "cat," "dog," or "bird," we specify:

$$P_{S|N=N} \sim \text{Categorical}(\theta_{N\to\text{cat}}, \theta_{N\to\text{dog}}, \theta_{N\to\text{bird}}) \tag{3.4}$$

Given a specific derivation $r_{1:m} = \langle \nu_1 \to \beta_1, \ldots, \nu_m \to \beta_m \rangle$, the probability mass assigned by the model is given by the product of the relevant parameters:

$$P_D(r_{1:m}) = \prod_{j=1}^{m} \theta_{\nu_j \to \beta_j} \tag{3.5}$$

The parametrization directly maps grammatical rules to categorical distribution parameters, making the relationship between the formal grammar and its probabilistic extension explicit and transparent.

3.5.5 Parameter Estimation via Maximum Likelihood Estimation

Given a corpus of sentences along with their correct derivations (parses), we can estimate the parameters of a PCFG using maximum likelihood estimation (MLE). The MLE principle selects parameter values that maximize the probability of the observed data. For PCFG parameters, this results in a particularly simple form based on relative frequency.

The MLE estimate for a parameter $\theta_{\nu\to\beta}$ is given by:

$$\theta_{\nu\to\beta} = \frac{\text{count}(\nu \to \beta)}{\sum_{(\nu\to\gamma)\in R} \text{count}(\nu \to \gamma)} \tag{3.6}$$

The numerator counts the number of times the specific rule $\nu \to \beta$ was applied in the annotated derivations of the training corpus. The denominator sums the counts of all rules with the same left-hand side nonterminal ν, ensuring that probabilities for all possible expansions of ν sum to 1. This relative frequency estimator is intuitive: the probability of rewriting a nonterminal in a particular way is simply the proportion of times that rewriting was observed in the training data when that nonterminal was expanded.

This estimator is guaranteed to be optimal in the sense of maximizing the likelihood of the training data, provided that we have an adequate supply of labeled training data. However, it suffers from the significant limitation that it assigns zero probability to any rule that does not appear in the training data, which can lead to poor generalization and is problematic for practical systems.

3.6 Using PCFGs: Language Modeling and Parsing

3.6.1 Marginal Probability of Sentences

One important use of PCFGs is as a language model that assigns probabilities to sentences based on their syntactic structure. Due to the potential for structural ambiguity, a given sentence may be generated by multiple distinct derivations, each representing a different syntactic analysis. The PCFG defines a marginal probability for a sentence $w_{1:\ell}$ by summing the probabilities of all derivations whose yield is that sentence:

$$P_X(w_{1:\ell}) = \sum_{\delta : S \overset{\delta}{\Rightarrow} w_{1:\ell}} P_D(\delta) \tag{3.7}$$

Here, the sum ranges over the entire space of derivations δ such that starting from the start symbol S and applying the rules in δ yields the target sentence. This marginalization operation is crucial because it means the model accounts for all possible syntactic structures that are consistent with the grammar and the input sentence. Sentences with multiple valid parses receive higher probability than sentences with a single parse, and sentences with no valid parse receive zero probability. This behavior captures an intuition that sentenceswith richer structural interpretations should be considered more likely under the model.

3.6.2 Language Model Evaluation

To evaluate a PCFG as a language model, we typically compute the perplexity on a held-out test set of sentences known to be valid in the language. Perplexity is defined as $\text{Perplexity} = 2^{-\frac{1}{N}\sum_i \log_2 P_X(w^{(i)})}$, where N is the total number of words in the test set and the sum ranges over test sentences. Lower perplexity indicates better language modeling performance. This evaluation metric allows us to assess how well the model predicts held-out data without having gold-standard parse annotations.

3.6.3 Parsing as Optimization

A second major use of PCFGs is as a syntactic parser that selects the most probable derivation consistent with the observed sentence. Given a sentence $w_{1:\ell}$, we seek the maximum probability derivation:

$$\hat{\delta} = \arg \max_{\delta : S \overset{\delta}{\Rightarrow} w_{1:\ell}} P_D(\delta) \tag{3.8}$$

This optimization problem asks for the single derivation that best explains the sentence under the model. The structure of this derivation can then be visualized as a parse tree and compared to human annotations to evaluate parsing accuracy.

3.6.4 Parsing Accuracy Evaluation

When evaluating parsing performance against human-annotated reference parses, we typically adopt a span-based evaluation scheme. This scheme treats syntactic analysis as a classification

problem over spans of the input sentence. For each contiguous span of words $w_{i:j}$, we ask whether the parser correctly identified this span as a constituent of a particular category (e.g., NP, VP, etc.). Standard metrics from classification include precision (the proportion of predicted constituents that are correct), recall (the proportion of true constituents that were identified), and the F1 score, which is the harmonic mean of precision and recall. These metrics are defined over constituent boundaries, so disagreements about internal structure within a constituent do not affect the evaluation, only whether boundaries are correctly identified. For a thorough treatment of parsing evaluation, see Section 18.8 of standard NLP textbooks.

3.7 The Parse Forest and the CKY Algorithm

3.7.1 Motivation for Efficient Parsing

Both major uses of PCFGs—language modeling and parsing—require us to compute statistics over all derivations of a sentence. The fundamental challenge is that the number of possible parse trees for a sentence can be exponentially large. When we restrict attention to binary-branching trees (arity 2), the number of distinct tree structures for a sentence of ℓ words is given by the Catalan number:

$$C_\ell = \frac{(2\ell)!}{(\ell+1)!\ell!} \tag{3.9}$$

These numbers grow extremely rapidly with sentence length: for 10 words, there are 16,796 possible binary parse trees; for 20 words, there are over 58 billion. Naive enumeration of all derivations is therefore computationally infeasible even for moderately long sentences.

The key insight that makes efficient parsing possible is the observation that parse trees are not arbitrary structures but rather are subject to the compositional constraints imposed by the context-free grammar. A parse tree can be decomposed into elementary building blocks—non-overlapping phrase categories over input spans—that are reused across derivations. The Cocke-Kasami-Younger (CKY) algorithm exploits this structure to provide a compact representation of the parse forest (the set of all possible parses) that permits efficient computation.

3.7.2 Spans and the Structure of the Parse Forest

The fundamental data structure in the CKY algorithm is the notion of a span, which identifies a contiguous substring of the input. A span is specified by a pair of positions (i, j) with $0 \leq i < j \leq \ell$, where ℓ is the sentence length. The span (i, j) corresponds to words $w_i, w_{i+1}, \ldots, w_{j-1}$ (using half-open interval notation where the end index is exclusive).

A parse forest can be viewed as a collection of phrase categories over all spans of the input sentence. POS tags (pre-terminal symbols in the grammar) correspond to spans covering single words, so span $(i, i+1)$ corresponds to word w_i. Longer spans are built up through two mechanisms: unary rules and binary rules. Unary rules allow us to derive a phrase category from a single nonterminal; for example, a nonterminal NP can be derived from another nonterminal N through a unary rule $NP \to N$. Binary rules combine two nonterminals covering different, adjacent spans; for example, a VP covering span $(1, 3)$ might be derived from a V covering span $(1, 2)$ and an NP covering span $(2, 3)$ via a binary rule $VP \to V\ NP$.

Importantly, the same phrase category can often be derived by multiple different rule applications, corresponding to different derivations within the parse forest. The CKY representation maintains a compressed encoding of all such derivations by storing, for each span and each nonterminal, all the possible ways that nonterminal can be derived over that span. This compression is what makes efficient computation possible: even though the total number of derivations may be exponential, the compressed representation is only cubic in sentence length.

3.7.3 CKY Algorithm: Overview

The CKY algorithm constructs a graph-like representation of the parse forest in a bottom-up manner. The algorithm maintains a table with one entry for each pair (i, j) of span boundaries and each possible nonterminal X. The entry Table$[i, j, X]$ stores information about all possible ways to derive the nonterminal X over the span (i, j).

The algorithm proceeds in stages, processing spans in order of increasing length. For spans of length 1 (corresponding to individual words), the algorithm applies all unary rules whose right-hand side is a terminal symbol that matches the word at that position. For spans of increasing length, the algorithm considers all ways to partition the span into two adjacent smaller spans and applies all binary rules whose right-hand side matches the nonterminals that were found to dominate those two smaller spans. After applying all possible rules at each span, the algorithm also applies all unary rules to the newly derived nonterminals, which may trigger further unary rules in a transitive closure process.

The algorithm's time complexity is $O(\ell^3 \cdot |R| \cdot |\text{max\_right\_hand\_side}|)$, where ℓ is the sentence length, $|R|$ is the number of grammar rules, and the final factor accounts for the cost of matching rule right-hand sides. For grammars in Chomsky Normal Form with binary branching, this reduces to cubic time in sentence length, making the algorithm practical for sentences of moderate length.

3.7.4 Computing Marginal Probabilities: The Inside Algorithm

The inside algorithm is a dynamic programming procedure that computes the total probability of all derivations of a given nonterminal over a given span. Let $\alpha(i, j, X)$ denote the inside probability, defined as the sum of probabilities of all derivations that rewrite the nonterminal X into the span of words $w_i, \ldots, w_{j-1}$:

$$\alpha(i, j, X) = \sum_{\delta : X \overset{\delta}{\Rightarrow} w_{i:j}} P_D(\delta) \tag{3.10}$$

The inside algorithm computes these quantities recursively. For base cases (spans of length 1), we compute the inside probabilities for all pre-terminal symbols using the lexical rules of the grammar. For longer spans, we compute $\alpha(i, j, X)$ by considering all ways to partition the span and all grammar rules that could produce X:

$$\alpha(i, j, X) = \sum_{X \to Y\ Z} \theta_{X \to Y\ Z} \sum_{k=i}^{j-1} \alpha(i, k, Y) \cdot \alpha(k, j, Z) + \sum_{X \to Y} \theta_{X \to Y} \cdot \alpha(i, j, Y) \tag{3.11}$$

The first term accounts for binary rules, where we sum over all possible split points k and multiply the inside probabilities of the two children. The second term accounts for unary rules. The cumulative sum of inside probabilities at the root of the tree, $\alpha(0, \ell, S)$, gives the marginal probability of the sentence.

3.7.5 Computing Maximum Probabilities: The Viterbi Algorithm

The Viterbi algorithm is analogous to the inside algorithm but computes maximum probabilities rather than marginal probabilities. Let $\beta(i, j, X)$ denote the Viterbi value, defined as the maximum probability of any single derivation that rewrite X into the span $w_{i:j}$:

$$\beta(i, j, X) = \max_{\delta : X \overset{\delta}{\Rightarrow} w_{i:j}} P_D(\delta) \tag{3.12}$$

The Viterbi algorithm computes these quantities similarly to the inside algorithm, but replaces sums with maxima:

$$\beta(i, j, X) = \max\left(\max_{X \to Y\, Z} \max_{k=i}^{j-1} \theta_{X \to Y\, Z} \cdot \beta(i, k, Y) \cdot \beta(k, j, Z), \max_{X \to Y} \theta_{X \to Y} \cdot \beta(i, j, Y)\right) \quad (3.13)$$

The Viterbi algorithm additionally maintains backpointers that record which rule application and split point achieved the maximum at each cell, permitting reconstruction of the best derivation after the algorithm completes. The maximum probability at the root, $\beta(0, \ell, S)$, is the probability of the best parse, and following the backpointers yields the maximum probability derivation.

3.8 Limitations and Extensions

3.8.1 Limitations of PCFGs

Despite their elegance and theoretical appeal, PCFGs suffer from several important limitations that restrict their applicability to real-world natural language. First, PCFGs are generative models that model the full joint distribution over sentences and derivations. As a consequence, they must make independence assumptions to remain tractable, and these independence assumptions do not always reflect actual linguistic patterns. In particular, the context-free assumption is violated by some linguistic phenomena, such as agreement and certain types of unbounded dependencies that span across structural boundaries.

Second, the limited use of linguistic context is a fundamental constraint of the generative formulation. A nonterminal's expansion depends only on the nonterminal itself, not on the surrounding context in which it appears. However, real linguistic structure is often heavily dependent on context; for example, the rewriting probabilities for a verb phrase may depend on whether it appears as the main verb of a matrix clause or as the head of a relative clause.

Third, the dynamic programming algorithms for PCFGs (both inside and Viterbi) have cubic time complexity in sentence length, which can become prohibitive for very long sentences or for applications requiring real-time performance. While polynomial time is much better than the exponential behavior of naive enumeration, it remains a substantial computational burden compared to models with linear or sub-cubic complexity.

3.8.2 Conditional Parametrization Approaches

One major class of extensions to PCFGs aims to preserve the notion of parsing as structured prediction while allowing more flexible parameterization of the conditional probability of parses given observations. Rather than modeling the joint distribution over sentences and parses, these approaches directly model the conditional distribution of parses given the input sentence, allowing arbitrary feature functions to capture relevant linguistic phenomena without being constrained by generative assumptions.

Examples of conditional parametrization approaches include transition-based parsers, which make parsing decisions incrementally as they process the sentence from left to right, and Conditional Random Field (CRF) parsers, which extend CRFs to structured output spaces. These approaches can easily incorporate sparse or dense features, including features that depend on the broader context surrounding a decision, and can be trained using discriminative objectives that directly optimize parsing accuracy rather than likelihood.

3.8.3 Dependency Grammars

Another important extension recognizes that sometimes the linguistically relevant structure is not the constituency (phrase structure) emphasized by CFGs but rather the dependencies

(relationships) between words. Dependency grammars make explicit the head-dependent relationships between words and often provide simpler, more interpretable representations of certain linguistic phenomena. Where CFGs emphasize which phrases form constituents, dependency grammars emphasize which words govern which other words. This alternative perspective has proven highly valuable for many practical applications and has spawned extensive research into dependency parsing algorithms and models.

3.9 Summary

The progression from simple bag-of-words models through HMMs to context-free and probabilistic context-free grammars represents a gradual incorporation of linguistic structure into computational models. PCFGs provide a principled framework that combines the symbolic expressiveness of context-free grammars with probabilistic modeling, permitting both language modeling applications and syntactic parsing. The CKY algorithm and its variants (inside and Viterbi algorithms) provide efficient dynamic programming solutions to the key computational problems posed by PCFGs, reducing the complexity from exponential enumeration to polynomial time. While PCFGs have limitations arising from their context-free assumptions and generative formulation, they remain theoretically important and provide valuable insights into the structure of natural language. Extensions to conditional models and alternative representations such as dependency grammars address specific limitations while building on the foundational concepts introduced through PCFG-based modeling.

Chapter 4

Morphology and Finite State Techniques

4.1 Introduction to Morphology

Morphology represents the study of word structure and formation in natural language, with particular focus on how words are composed from smaller meaningful units. The fundamental building block in morphological analysis is the **morpheme**, which is defined as the minimal information-carrying unit in language. Morphemes cannot be further decomposed into smaller meaningful parts while retaining their semantic content.

Words are constructed from one or more stems, potentially combined with zero or more affixes. A **stem** constitutes the core meaningful component of a word, while an **affix** is a morpheme that only occurs in conjunction with other morphemes and cannot stand alone. For compound words, multiple stems can be combined together, as illustrated by examples such as `dog+s` (stem: dog, affix: +s) and `book+shop+s` (stems: book and shop, affix: +s).

It is crucial to distinguish between genuine morphemes and superficially similar phonological patterns that lack morphological status. For instance, while words like *slither*, *slide*, and *slip* share somewhat similar meanings and begin with the same sound sequence, the initial *sl-* does not constitute a morpheme because it does not carry consistent, independent meaning across these words.

4.2 Types of Affixation

Affixes can be categorized based on their position relative to the stem they modify. The primary affixation patterns observed across languages include the following types.

Suffixes are affixes that appear after the stem, as seen in English examples like `dog+s` and `truth+ful`. Suffixation represents the most common and productive affixation pattern in English morphology.

Prefixes appear before the stem they modify. In English, prefixes are exclusively derivational and do not serve inflectional functions. Common examples include `un+wise`, where the prefix *un-* modifies the adjective stem *wise*.

Infixes are inserted within the stem itself, breaking the stem into discontinuous parts. While infixation is productive in languages such as Arabic (where the stem *k_t_b* yields forms like *kataba* meaning "he wrote" and *kotob* meaning "books"), English exhibits only non-productive historical remnants of this pattern. The vowel alternation in *sing* to *sang* represents such a historical infix pattern, though it is no longer productive. An arguably productive example in colloquial English might be *absobloodylutely*, though such forms remain marginal.

Circumfixes consist of two parts that surround the stem, appearing both before and after

it. English lacks circumfixation entirely. However, German provides clear examples, such as `ge+kauf+t` (where the stem *kauf* is surrounded by the circumfix *ge-...-t*).

4.3 Productivity in Morphology

The concept of **productivity** refers to whether a particular affixation pattern applies generally to stems in a language and, critically, whether it extends to newly coined or borrowed words. Productivity determines which morphological patterns speakers actively use to create new word forms versus which patterns exist only in historically established irregular forms.

Consider the irregular past tense patterns exemplified by *sing, sang, sung* and *ring, rang, rung*. When speakers encounter the novel verb *ping*, they do not apply this vowel-alternation pattern; instead, they produce the regular form *pinged* for both past tense and past participle. This demonstrates that the vowel-alternation infixation pattern is not productive in modern English. The verbs *sing* and *ring* are consequently classified as irregular forms, representing historical remnants of a once-productive pattern that is no longer generalized to new forms.

4.4 Inflectional Morphology

Inflectional morphology concerns affixes that specify grammatical features within paradigmatic systems without fundamentally altering the core lexical meaning or syntactic category of the stem. English inflectional suffixes include the plural marker `+s` (as in *dogs*) and the past participle marker `+ed` (as in *walked*).

Inflectional affixes serve to fill specific slots within grammatical paradigms, encoding features such as tense, aspect, number, person, gender, and case. A defining characteristic of English inflectional morphology is that inflectional affixes are not combined or stacked; each word receives at most one inflectional suffix.

Inflectional affixation is generally fully productive in English, meaning it applies systematically to all eligible stems, including newly introduced words. This is evidenced by the immediate acceptance of forms like *texted* (the past tense of the novel verb *text* derived from the noun), which speakers produce and understand without hesitation. The primary exception to this productivity is the set of irregular inflectional forms (such as *went*, *sang*, *children*) that must be individually memorized.

4.5 Derivational Morphology

Derivational morphology involves affixes that create new lexical items, often changing the meaning substantially or altering the part of speech of the stem. Common English derivational affixes include prefixes such as *un-*, *re-*, and *anti-*, as well as suffixes like *-ism*, *-ist*, *-ful*, and *-ness*.

Unlike inflectional affixes, derivational affixes present a broad range of semantic possibilities and can fundamentally transform the meaning or syntactic category of their base. For example, *-ness* converts adjectives into nouns (*happy* $\rightarrow$ *happiness*), while *-ful* converts nouns into adjectives (*truth* $\rightarrow$ *truthful*).

A striking characteristic of derivational morphology is that derivational affixes can be combined in indefinite sequences, creating highly complex words. The canonical example *antiantidisestablishmentarianism* decomposes as `anti-anti-dis-establish-ment-arian-ism`, demonstrating the recursive application of multiple derivational affixes.

Derivational affixation is generally **semi-productive**, meaning that while patterns exist, they do not apply uniformly to all potential stems, and acceptability varies. Consider the suffix *-ee* (indicating a person affected by an action): forms like *escapee* and *textee* are acceptable,

dropee and *snoree* are questionable (marked with ?), and forms like *cricketee* are unacceptable (marked with *). This gradation in acceptability characterizes semi-productive patterns.

Zero-derivation (also called conversion) occurs when a word changes syntactic category without any overt affixation. Examples include the nouns *tango* and *waltz*, which can be used directly as verbs (*to tango*, *to waltz*) without any morphological marking.

4.6 Morphological Structure and Ambiguity

Morphological analysis often confronts multiple forms of ambiguity that must be resolved through syntactic and semantic context.

Morpheme ambiguity arises when individual stems or affixes have multiple possible interpretations. For instance, the stem *dog* can function as either a noun or a verb, while the suffix *+s* can indicate either plural number for nouns or third-person singular present tense for verbs. Without broader sentential context, the form *dogs* could theoretically represent either the plural noun or the third-person singular verb form.

Structural ambiguity concerns cases where a single surface form admits multiple possible morphological decompositions. Consider the word *shorts*, which could be analyzed either as a monomorphemic stem *shorts* or as the stem *short* with the plural suffix *-s*. More complex is the case of *unionised*, which permits two distinct analyses: `union-ise-ed` (meaning "formed into a union") or `un-ion-ise-ed` (meaning "not ionized").

The resolution of structural ambiguity requires consideration of morphological **bracketing**, which represents the hierarchical structure of affix application. For the decomposition `un-ion-ise-ed`, we must determine whether the structure is `((un-ion)-ise)-ed` or `(un-((ion-ise)-ed))`. Since *un-ion* is not a possible English word, the former structure is ruled out. Furthermore, the prefix *un-* exhibits systematic ambiguity: when attached to verbs, it indicates reversal of an action (*untie*, *undo*), while when attached to adjectives, it indicates negation (*unwise*, *unsurprised*). Since *ionised* functions as an adjective (the past participle used adjectivally), the correct bracketing is `(un-((ion-ise)-ed))`, with *un-* negating the adjectival form *ionised*.

4.7 Applications of Morphological Processing in NLP

Morphological processing serves several critical functions in natural language processing systems.

Compiling full-form lexicons involves generating all inflected and derived forms of lexical stems, creating comprehensive word lists that can be used for recognition and lookup tasks. This approach is feasible for languages with relatively simple morphology like English but becomes computationally expensive for morphologically rich languages.

Stemming for information retrieval reduces words to a common base form to improve recall in search applications. It is important to note that the "stem" produced by stemming algorithms (such as the widely-used Porter stemmer) is not necessarily a linguistically valid stem; rather, it is a canonical form designed to group morphologically related words together. For example, a stemmer might reduce both *fishing* and *fisher* to the common form *fish*, even though the linguistically correct stems differ.

Lemmatization involves finding the dictionary form (lemma) of a word, typically by stripping inflectional affixes while preserving derivational morphology. Lemmatization commonly handles only inflectional morphology, identifying stems and affixes as a precursor to syntactic parsing. Unlike stemming, lemmatization produces linguistically valid word forms.

Generation refers to the process of producing surface word forms from underlying stems and grammatical specifications, essentially the reverse of morphological analysis. This is essential for natural language generation systems that must produce correctly inflected forms.

Morphological processing systems are typically **bidirectional**, supporting both analysis (parsing surface forms into stems and affixes) and generation (producing surface forms from abstract representations). For example, the relationship between `party + PLURAL` and *parties*, or between `sleep + PAST_VERB` and *slept*, can be processed in either direction depending on the application requirements.

4.8 Aspects of Morphological Processing

A complete morphological processing system must address three interrelated aspects of word structure.

First, the surface form must be mapped to underlying stem(s) and affixes, which can be represented in two ways. Option 1 maintains the explicit affix strings, as in `pinged / ping-ed`, where the surface form *pinged* decomposes into the stem *ping* and the suffix *-ed*. Option 2 abstracts affixes to grammatical feature labels, producing representations like `pinged / ping PAST_VERB` and `pinged / ping PSP_VERB` (where PSP denotes past participle). This second option proves particularly valuable for irregular forms such as `sang / sing PAST_VERB` and `sung / sing PSP_VERB`, where the surface form does not decompose into a simple stem-plus-affix sequence.

Second, the internal structure or bracketing must be determined, capturing the hierarchical organization of morphological operations. As discussed previously, the form *unionised* requires bracketing such as `(un-((ion-ise)-ed))` to properly represent the scope of each affix.

Third, syntactic and semantic effects must be integrated. Syntactic parsing can filter the results of morphological analysis by eliminating analyses that would produce syntactically or semantically ill-formed structures. For instance, the surface form *feed* might be analyzed both as the monomorphemic verb stem *feed* and as the hypothetical derivation *fee-ed*. Syntactic context typically eliminates the spurious *fee-ed* analysis, as it would not fit the grammatical and semantic requirements of the surrounding sentence.

4.9 Lexical Requirements for Morphological Processing

Effective morphological processing systems require carefully structured lexical resources encompassing three essential components.

Affixes must be listed along with the grammatical information they convey. For example, the suffix *-ed* must be associated with both `PAST_VERB` (past tense) and `PSP_VERB` (past participle) features, while the suffix *-s* must be associated with `PLURAL_NOUN` when attaching to nouns and with `3SG_PRESENT_VERB` when attaching to verbs. This feature specification enables the system to properly interpret and generate inflected forms.

Irregular forms must be explicitly listed with information analogous to that provided for regular affixes. Since irregular forms do not follow productive patterns, they cannot be generated by rule and must instead be stored as lexical exceptions. Examples include entries like `began PAST_VERB begin` and `begun PSP_VERB begin`, which specify both the surface irregular form and the underlying stem.

Stems must be stored with their syntactic categories to prevent spurious analyses. For instance, without knowing that *corpus* is a valid noun stem, a morphological analyzer might incorrectly decompose it as *corpu-s*, treating it as the plural of a nonexistent stem *corpu*. Marking *corpus* as a noun stem blocks this incorrect analysis.

4.10 Spelling Rules in Morphological Processing

English morphology is fundamentally **concatenative**, meaning that morphemes are combined primarily through linear sequencing rather than through complex internal modifications. However, this concatenation is subject to regular phonological and spelling changes that must be modeled explicitly.

Irregular morphology, particularly inflectional irregularity, must be handled through explicit listing of exceptional forms, as these patterns are not predictable by rule. However, regular phonological and spelling changes follow systematic patterns that can be captured through general rules independent of specific stems or affixes.

Consider the pronunciation of the plural suffix *-s*. When this suffix attaches to stems ending in sibilant sounds (represented orthographically as *s*, *x*, or *z*), an epenthetic vowel is inserted for phonological reasons. English spelling reflects this phonological requirement through the insertion of the letter *e*, producing forms like *boxes*, *buses*, and *fizzes* rather than the phonologically impossible **boxs*, **buss*, or **fizzs*.

This **e-insertion rule** can be formalized using rewrite rule notation. The underlying form contains an affix boundary marker (represented as ^) separating the stem from the suffix, so *box+s* is represented as `box^s` at the underlying level. The e-insertion rule is formalized as:

$$\varepsilon \rightarrow e \; / \; \left\{ \begin{matrix} s \\ x \\ z \end{matrix} \right\} \hat{} \; s$$

This notation specifies that the empty string (ε) is rewritten as the letter *e* (the mapping, shown to the left of the slash) in the specific context where it appears between a sibilant consonant (*s*, *x*, or *z*) and the affix boundary followed by the suffix *s* (the context, shown to the right of the slash). The caret symbol (^) represents the affix boundary between the stem and the affix. This same rule applies both to noun plurals (*boxes*) and to third-person singular present tense verbs (*mixes*).

These spelling rules map between an underlying representation (which maintains clear morpheme boundaries and uses abstract representations) and the surface representation (the actual orthographic form). The notation conventions include: the caret (^) for affix boundaries, ε for the empty string, and the slash separating the mapping operation from its conditioning context.

4.11 Finite State Automata for Recognition

Morphological rules and patterns can be formalized and implemented using finite state techniques, which provide efficient computational mechanisms for recognition and transduction.

A **finite state automaton** (FSA) consists of a finite set of states connected by transitions labeled with input symbols. The automaton processes input sequentially, moving from state to state according to the transition labels, and accepts an input string if it ends in an accept state (conventionally indicated by a double circle).

Consider a simple FSA designed to recognize date patterns of the form day/month, such as *12/2* or *3/11*. This automaton includes states and transitions that handle digits before and after the slash separator. However, a simple implementation might be **non-deterministic**, meaning that after reading certain inputs (such as the digit *2*), the automaton could simultaneously be in multiple states. Non-determinism does not affect the recognition power of the automaton but has implications for implementation efficiency.

An important consideration in FSA design is the tradeoff between simplicity and precision. A simple automaton for date recognition might accept the intended patterns *11/3* and *3/12*, but it might also accept invalid patterns like *37/00* through **overgeneration**. More sophisticated automata can enforce tighter constraints, though at the cost of increased complexity.

4.12 Finite State Transducers

While finite state automata recognize patterns, **finite state transducers** (FSTs) map between two representations, making them ideal for morphological processing where we must convert between surface forms and underlying representations.

An FST extends the FSA framework by labeling each transition with a pair of symbols, written as `input:output`. As the transducer processes an input string, it simultaneously produces an output string according to the transition labels. FSTs are bidirectional: they can perform **analysis** (mapping surface forms to underlying forms) or **generation** (mapping underlying forms to surface forms).

The e-insertion spelling rule discussed earlier can be implemented as a finite state transducer. This FST includes transitions labeled with symbol pairs that map between surface and underlying representations. For most characters, the transducer uses identity mappings (`a:a`, `b:b`, etc.), copying the character unchanged. The key transitions handle the e-insertion context:

State 1 serves as the initial state, with transitions that map most characters to themselves (`other:other`). When the transducer encounters a sibilant (*s*, *x*, or *z*), it moves to state 2. From state 2, if the next input character is *e* followed by *s*, the transducer can take a path that maps the affix boundary, producing the output `^s` from the input *es*. Alternatively, if the input contains the boundary marker directly, the transducer can insert the *e* in the output.

Consider the analysis of *boxes*:

- Input string: *b o x e s*
- The transducer processes: `b:b`, `o:o`, `x:x` (transitioning to state 2 due to the sibilant)
- From state 2, the sequence *e s* can be mapped to `^ s`
- Accepted output: `b o x ^ s`

Crucially, FSTs can produce multiple outputs for a given input when ambiguity exists. The transducer might also produce `b o x e s` (treating the word as monomorphemic) or `b o x e ^ s` (analyzing the *e* as part of the stem). Subsequent lexical lookup determines which analysis is valid.

4.13 Computational Implementation with FSTs

The practical deployment of finite state transducers in morphological processing systems requires attention to several implementation considerations.

FSTs assume that input has been **tokenized**, meaning that word boundaries have been identified and each word has been segmented into its component characters. Importantly, each transition in an FST processes exactly one character pair; this granularity ensures that the finite state model remains tractable.

In **analysis mode**, FSTs return character sequences with explicit affix boundary markers (represented by ^), enabling subsequent lexical lookup to verify that the identified stem and affixes exist in the lexicon. This two-stage process (morphological segmentation followed by lexical verification) allows the system to eliminate spurious analyses.

In **generation mode**, the input to the FST comes from stem and affix lexicons, which provide underlying forms with abstract boundary markers. The FST then produces the correct surface orthographic form by applying spelling rules.

Complex morphological systems typically require multiple spelling rules. These can be handled through two architectural approaches: either multiple FSTs are composed (combined) into a single large FST that captures all interactions between rules, or multiple FSTs are run in

parallel, with each FST implementing one rule. Composition produces more efficient runtime performance but increases the system's complexity and construction time.

An important limitation of FSTs is that they cannot represent hierarchical morphological structure. FSTs can segment words into stems and affixes, but they cannot encode bracketing information such as `(un-((ion-ise)-ed))`. This is because FSTs operate through local state transitions and lack the memory capacity to track hierarchical structure. Similarly, FSTs cannot condition processing on the history of previous transitions beyond the current state, potentially leading to redundancy when multiple rules must access the same contextual information.

4.14 Other Applications of Finite State Techniques in NLP

Beyond morphological processing, finite state techniques find application in various natural language processing tasks where sequential structure and limited context suffice for modeling.

Dialogue models for spoken dialogue systems commonly employ finite state automata to track conversation state and determine appropriate system behavior. Consider a simple system for obtaining a date from a user, which must handle four possible information states: (1) no information is known and the system must prompt for both month and day; (2) only the month is known and the system must prompt for the day; (3) only the day is known and the system must prompt for the month; (4) both month and day are known and the system can proceed to the next task.

These states can be organized as nodes in a finite state automaton, with transitions corresponding to user inputs. From the initial state (no information), transitions labeled `month` lead to state 2, transitions labeled `day` lead to state 3, and transitions labeled `day&month` lead directly to state 4. The transition labeled `mumble` (representing unclear or unrecognized input) loops back to the same state, allowing the system to re-prompt.

This basic deterministic FSA can be extended to a **probabilistic finite state automaton** by associating probabilities with each transition. From state 1, we might assign probability 0.5 to the `month` transition, probability 0.1 to the `day` transition, probability 0.3 to the `day&month` transition, and probability 0.1 to the `mumble` transition. These probabilities capture the likelihood of different user responses and can be estimated from dialogue corpus data. Probabilistic FSAs enable systems to make optimal decisions under uncertainty and to identify anomalous interaction patterns.

4.15 Implementation Methods for Morphological Processing

Contemporary NLP systems implement morphological processing through diverse methodological approaches, each with distinct advantages and limitations.

Rule-based methods employ explicitly encoded linguistic rules, often formalized using finite state transducers or similar frameworks. The Porter stemmer exemplifies this approach, implementing a sequence of heuristic rules that iteratively strip common English suffixes to produce stem forms. The Porter stemmer is widely deployed in information retrieval applications due to its simplicity and computational efficiency. It is available as part of the NLTK (Natural Language Toolkit) for Python and is frequently used in practical NLP implementations and instructional contexts.

Probabilistic models for morphological segmentation learn patterns from annotated training data rather than relying on hand-crafted rules. These models treat morphological segmentation as a statistical inference problem, using techniques such as conditional random fields, neural sequence-to-sequence models, or Bayesian segmentation algorithms. Probabilistic approaches can generalize beyond explicitly encoded rules and can be trained for languages where linguistic expertise is limited. These methods are discussed in greater depth in advanced sec-

tions of natural language processing curricula. Here's comprehensive LaTeX code for detailed notes on lexical semantics and word embeddings based on the lecture material:

"'latex

Chapter 5

Lexical Semantics and Word Embeddings

5.1 Introduction to Lexical Semantics

The field of semantics can be broadly divided into two main areas. Compositional semantics studies how meanings of phrases are constructed out of the meaning of individual words, following the principle of compositionality, which states that the meaning of each whole phrase is derivable from the meaning of its parts. Sentence structure conveys meaning that is obtained through syntactic representation. In contrast, lexical semantics studies how the meanings of individual words themselves can be represented and induced. This involves understanding what constitutes lexical meaning at the most fundamental level.

Current understanding of lexical meaning comes from recent results in psychology and cognitive neuroscience, though we do not yet have a complete picture. Different representations have been proposed in the literature, including formal semantic representations based on logic, taxonomies relating words to each other, and distributional representations in statistical natural language processing. However, none of these representations provides a complete account of lexical meaning on its own.

5.1.1 Approaches to Lexical Meaning

Formal semantics employs a set-theoretic approach to meaning representation. Under this framework, the meaning of a word like cat, denoted as cat', is represented as the set of all cats, while bird' represents the set of all birds. This approach uses meaning postulates to define relationships between concepts, such as $\forall x[\text{bachelor}'(x) \rightarrow \text{man}'(x) \wedge \text{unmarried}'(x)]$, which states that for all x, if x is a bachelor, then x is a man and x is unmarried. However, this approach faces significant limitations. Consider the question of whether the Pope qualifies as a bachelor under this definition. More fundamentally, defining concepts through enumeration of all of their features is highly problematic in practice. Attempting to define everyday concepts such as chair, tomato, thought, or democracy proves nearly impossible for most concepts when using strict feature enumeration.

Prototype theory offers an alternative approach to set-theoretic methods. Introduced by Eleanor Rosch in her 1975 work on cognitive representation of semantic categories, prototype theory introduces the notion of graded semantic categories. Under this theory, categories do not have clear boundaries, and there is no requirement that a property be shared by all members of a category. Certain members of a category are considered more central or prototypical, meaning they better instantiate the prototype. For instance, within the furniture category, a chair is more prototypical than a stool. Categories form around these prototypes, and new members are added on the basis of resemblance to the prototype rather than strict feature matching.

5.1.2 Semantic Relations

Words in natural language are connected through various semantic relations that help structure our lexical knowledge. Hyponymy represents an IS-A relationship, where one concept is a specific instance of a more general concept. For example, dog is a hyponym of animal, while animal is a hypernym of dog. These hyponymy relationships form a taxonomy that works particularly well for concrete nouns, creating hierarchical structures of meaning.

Meronomy expresses a PART-OF relationship between concepts. An arm is a meronym of body, and a steering wheel is a meronym of car. This relation captures the compositional structure of physical objects and abstract concepts. Synonymy represents the relationship between words with identical or nearly identical meanings, such as aubergine and eggplant. Antonymy captures oppositional relationships, as seen in word pairs like big and little.

Beyond these basic relations, natural language exhibits near-synonymy or similarity, where words have closely related but not identical meanings. Examples include exciting and thrilling, or the graded series slim, slender, thin, and skinny, each carrying slightly different connotations. WordNet serves as a large-scale lexical resource that systematically links words by their semantic relations, providing a computational framework for accessing these structured relationships.

5.1.3 Polysemy and Word Senses

Words in natural language frequently exhibit polysemy, where a single word form corresponds to multiple related meanings or senses. The verb run provides an illustrative example of this phenomenon. In the sentence "The children ran to the store," run denotes physical movement. In "If you see this man, run!" it conveys urgency or escape. "Service runs all the way to Cranbury" describes route coverage. "She is running a relief operation in Sudan" indicates management or operation. "The story or argument runs as follows" means to proceed or develop in a particular way. "Does this old car still run well?" asks about mechanical function. "Interest rates run from 5 to 10 percent" expresses a range. "Who's running for treasurer this year?" refers to candidacy in an election. "They ran the tapes over and over again" means to play or execute. Finally, "These dresses run small" describes sizing tendencies. This multiplicity of senses presents significant challenges for computational systems attempting to understand and process natural language.

5.2 Distributional Semantics

5.2.1 The Distributional Hypothesis

The foundational principle of distributional semantics is captured in the statement "You shall know a word by the company it keeps," attributed to J. R. Firth. Wittgenstein similarly stated that "The meaning of a word is defined by the way it is used." These principles form the basis of the distributional hypothesis. Consider the word scrumpy appearing in contexts such as "it was authentic scrumpy, rather sharp and very strong," "we could taste a famous local product — scrumpy," "spending hours in the pub drinking scrumpy," and "Cornish Scrumpy Medium Dry." Even without prior knowledge of the word, these contexts provide strong evidence that scrumpy refers to some kind of alcoholic beverage, likely cider.

This leads to the formal distributional hypothesis about word meaning. The context surrounding a given word provides information about its meaning. Words are similar if they share similar linguistic contexts. Therefore, semantic similarity can be approximated by distributional similarity. This provides a computational framework for inducing word meaning from text corpora without explicit human annotation.

5.2.2 Vector Space Representation

Distributions are represented as vectors in a multidimensional semantic space. The semantic space has dimensions which correspond to possible contexts, which serve as features. A distribution can be conceptualized as a point in that space, with the vector being defined with respect to the origin of that space. For example, scrumpy might be represented as a vector with components such as pub with weight 0.8, drink with weight 0.7, strong with weight 0.4, joke with weight 0.2, mansion with weight 0.02, and zebra with weight 0.1, among many other dimensions.

5.2.3 Defining Context

The notion of context can be operationalized in several ways. Word windows define context as n words on either side of the lexical item. For example, with $n = 2$ creating a 5-word window around the word minister in the phrase "The prime minister acknowledged the question," the context representation would be minister with the following counts: the appears 2 times, prime appears 1 time, acknowledged appears 1 time, and question appears 0 times within the window.

Lexeme windows operate similarly to word windows but use stems instead of full word forms. In the same example, the representation would be minister with the 2, prime 1, acknowledge 1, and question 0. This approach reduces sparsity by conflating different inflected forms.

Syntactic relations or dependencies define context through the syntactic dependency structure that a lexical item participates in. For the sentence "The prime minister acknowledged the question," the context for minister would include prime with count 1 and acknowledge with count 1. This can be further refined by including the specific dependency relation types, resulting in minister with prime_mod appearing 1 time and acknowledge_subj appearing 1 time. This approach captures more precise syntactic-semantic relationships.

5.2.4 Context Weighting Schemes

Different weighting schemes can be applied to context representations. The binary model assigns a value of 1 to dimension c of vector $\vec{w}$ if context c co-occurs with word w, and 0 otherwise. This is the simplest representation but loses frequency information.

The basic frequency model sets the value of vector $\vec{w}$ for dimension c equal to the number of times that c co-occurs with w. This preserves raw frequency information but does not account for how characteristic a context is.

The characteristic model gives weights to vector components that express how characteristic a given context is for word w. Pointwise Mutual Information (PMI) provides a principled approach to this weighting. PMI is defined as:

$$\text{PMI}(w, c) = \log \frac{P(w, c)}{P(w)P(c)} = \log \frac{P(w)P(c|w)}{P(w)P(c)} = \log \frac{P(c|w)}{P(c)}$$

The probability $P(c)$ is estimated as $\frac{f(c)}{\sum_k f(c_k)}$, where $f(c)$ is the frequency of context c and the sum ranges over all contexts. The conditional probability $P(c|w)$ is estimated as $\frac{f(w,c)}{f(w)}$, where $f(w, c)$ is the frequency of word w occurring in context c and $f(w)$ is the frequency of word w in all contexts. Substituting these estimates yields:

$$\text{PMI}(w, c) = \log \frac{f(w, c) \sum_k f(c_k)}{f(w)f(c)}$$

This formulation captures how much more likely a context c is to occur with word w compared to what would be expected by chance if they were independent.

5.2.5 Choosing the Semantic Space

The dimensionality and construction of the semantic space involves important tradeoffs. Using the entire vocabulary as dimensions includes all information, even rare contexts, but results in inefficient representations with hundreds of thousands of dimensions, introduces noise from spurious contexts, and produces sparse vectors with many zero entries.

Restricting to the top n words with highest frequencies, typically ranging from 2000 to 10000 dimensions, provides more efficient computation and includes only genuine words. However, this approach may miss infrequent but relevant contexts that carry important semantic information.

Dimensionality reduction using matrix factorization techniques yields very efficient representations, typically with 200 to 500 dimensions, and captures generalizations in the data by identifying latent semantic patterns. The primary disadvantage is that the resulting matrices are not directly interpretable, as dimensions no longer correspond to specific words or contexts.

Word frequency in natural language follows a Zipfian distribution, where a small number of words occur with very high frequency while the vast majority of words are rare. This distribution motivates careful selection of which words to use as dimensions in the semantic space.

5.2.6 Measuring Similarity

Once words are represented as vectors in semantic space, similarity can be quantified through distance or similarity metrics. The cosine similarity measure is defined as:

$$\cos(\theta) = \frac{\sum_k v_{1k} \cdot v_{2k}}{\sqrt{\sum_k v_{1k}^2} \cdot \sqrt{\sum_k v_{2k}^2}}$$

The cosine measure calculates the angle between two vectors and is therefore independent of vector length, focusing solely on directional similarity. Other measures include Euclidean distance, which does account for magnitude differences.

Empirical examples of similarity scores demonstrate the scale and interpretation of these measures. The pair house and building achieves a score of 0.43, gem and jewel score 0.31, capitalism and communism score 0.29, motorcycle and bike score 0.29, test and exam score 0.27, school and student score 0.25, singer and academic score 0.17, horse and farm score 0.13, man and accident score 0.09, tree and auction score 0.02, and cat and county score 0.007.

The notion of similarity in distributional semantics is intentionally broad, encompassing synonyms, near-synonyms, hyponyms, taxonomical siblings, antonyms, and other related concepts. This notion correlates with psychological reality and can be evaluated through correlation with human judgments on standardized test sets such as Miller and Charles (1991), WordSim, MEN, and SimLex. Well-constructed distributional models achieve correlations of 0.8 or higher with human judgments.

Distributional methods provide a usage-based representation of meaning. These representations are corpus-dependent, culture-dependent, and register-dependent. For example, the similarity between policeman and cop is measured at 0.23, but this value would vary significantly across different corpora reflecting different linguistic registers and cultural contexts.

5.3 Semantics with Dense Vectors

5.3.1 Count-Based versus Prediction-Based Models

Distributional semantic models can be categorized into two main types. Count-based models use explicit vectors where dimensions correspond to elements in the context. These result in long sparse vectors with interpretable dimensions, where each dimension can be traced back to a specific word or context type.

Prediction-based models train a model to predict plausible contexts for a word and learn word representations during this process. These produce short dense vectors with latent dimensions that do not correspond directly to interpretable features. The distinction between sparse and dense vectors has important practical implications.

Dense vectors are easier to use as features in machine learning tasks because there are fewer weights to tune in downstream models. They may generalize better than storing explicit counts by capturing underlying patterns rather than surface co-occurrence statistics. Dense vectors may perform better at capturing synonymy. In count-based models, car and automobile are distinct dimensions, so the model will not automatically capture similarity between a word that frequently appears near car and a word that frequently appears near automobile. Dense vector models can learn that these contexts are semantically equivalent.

5.3.2 The Skip-Gram Model

Mikolov and colleagues introduced word2vec in their 2013 paper "Efficient Estimation of Word Representations in Vector Space." The skip-gram model, inspired by work on neural language models, trains a neural network to predict neighboring words and learns dense embeddings for words in the training corpus during this process.

The intuition behind skip-gram is that words with similar meanings often occur near each other in texts. Given a word $w(t)$, the model predicts each neighboring word in a context window of $2L$ words from the current word. For $L = 2$, the model predicts four neighboring words: $[w(t-2), w(t-1), w(t+1), w(t+2)]$.

5.3.3 Parameter Matrices and Similarity

The skip-gram model learns two embeddings for each word $w_j \in V_w$. The word embedding $\mathbf{v}$ is stored in word matrix W, while the context embedding $\mathbf{c}$ is stored in context matrix C. Matrix W has dimensions $|V_w| \times d$, where $|V_w|$ is vocabulary size and d is embedding dimensionality. Matrix C has dimensions $d \times |V_w|$. The intuition is that similarity can be computed as the dot product between a target vector and a context vector.

During training, we walk through the corpus pointing at word $w(t)$, whose index in the vocabulary is j, which we call w_j. Our goal is to predict $w(t+1)$, whose index in the vocabulary is k, which we call w_k. To accomplish this, we need to compute $p(w_k|w_j)$. The intuition behind skip-gram is that computing this probability requires computing similarity between w_j and w_k.

Similarity is computed as a dot product between the target vector and context vector: $\text{Similarity}(c_k, v_j) \propto c_k \cdot v_j$. This builds on the connection to cosine similarity. Recall that cosine similarity is defined as:

$$\cos(v_1, v_2) = \frac{\sum_k v_{1k} \cdot v_{2k}}{\sqrt{\sum_k v_{1k}^2} \cdot \sqrt{\sum_k v_{2k}^2}} = \frac{v_1 \cdot v_2}{||v_1||\,||v_2||}$$

Cosine similarity is simply a normalized dot product. In skip-gram, similar vectors have a high dot product: $\text{Similarity}(c_k, v_j) \propto c_k \cdot v_j$.

To convert similarity into a probability, we normalize by passing through a softmax function:

$$p(w_k|w_j) = \frac{e^{c_k \cdot v_j}}{\sum_{i \in V} e^{c_i \cdot v_j}}$$

This ensures that probabilities sum to 1 across all possible context words.

5.3.4 Learning Procedure

The learning procedure begins with randomly initialized embeddings. At training time, we walk through the corpus and iteratively make the embeddings for each word more similar to the

embeddings of its neighbors and less similar to the embeddings of other words. The objective is to learn parameters C and W that maximize the overall corpus probability:

$$\arg\max \prod_{(w_j,w_k)\in D} p(w_k|w_j)$$

where D represents the set of word-context pairs observed in the corpus. Substituting the softmax formulation:

$$\arg\max \prod_{(w_j,w_k)\in D} \frac{e^{c_k\cdot v_j}}{\sum_{i\in V} e^{c_i\cdot v_j}}$$

Taking logarithms transforms the product into a sum, which is more numerically stable and computationally convenient.

5.3.5 Network Architecture

The skip-gram model can be visualized as a neural network with three layers. The input layer receives a one-hot encoded vector representation of the target word. A one-hot vector has length $|V|$ with value 1 for the target word and 0 for all other words. If bear is vocabulary word 5, the one-hot vector is $[0, 0, 0, 0, 1, 0, 0, 0, 0, \ldots, 0]$.

The projection layer has dimensionality d and represents the embedding for w_t. The multiplication of the one-hot input vector (dimension $1 \times |V|$) by the weight matrix W (dimension $|V| \times d$) yields the $1 \times d$ embedding vector. The output layer produces probabilities of context words through matrix C (dimension $d \times |V|$), resulting in a $1 \times |V|$ probability distribution.

5.3.6 Negative Sampling

A significant computational problem with the softmax formulation is that the denominator requires summing over the entire vocabulary:

$$p(w_k|w_j) = \frac{e^{c_k\cdot v_j}}{\sum_{i\in V} e^{c_i\cdot v_j}}$$

Computing this for every training example is prohibitively expensive for large vocabularies. Negative sampling approximates the denominator by sampling k noise samples or negative samples rather than computing over all words.

At training time, we walk through the corpus, and for each target word and positive context, we sample k negative examples. For instance, with the phrase "lemon, a tablespoon of apricot preserves or jam," the positive contexts for apricot are tablespoon, of, jam, and or. Negative examples might include cement, idle, dear, coaxial, attendant, whence, forever, and puddle.

The dataset is converted into word pairs labeled as positive or negative. Positive pairs include (apricot, tablespoon), (apricot, of), (apricot, jam), and (apricot, or). Negative pairs include (apricot, cement), (apricot, idle), (apricot, attendant), and (apricot, dear).

Instead of treating the problem as multi-class classification with a probability distribution over the whole vocabulary, we return a probability that word w_k is a valid context for word w_j: $P(+|w_j, w_k)$. The probability of a negative pair is $P(-|w_j, w_k) = 1 - P(+|w_j, w_k)$.

Modeling similarity as dot product Similarity$(c_k, v_j) \propto c_k \cdot v_j$, we convert this into a probability using the sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Therefore:

$$P(+|w_j, w_k) = \frac{1}{1 + e^{-c_k \cdot v_j}}$$

$$P(-|w_j, w_k) = 1 - P(+|w_j, w_k) = 1 - \frac{1}{1 + e^{-c_k \cdot v_j}} = \frac{1}{1 + e^{c_k \cdot v_j}}$$

The objective with negative sampling is to make the word similar to context words and dissimilar to negative examples:

$$\arg\max \prod_{(w_j, w_k) \in D^+} p(+|w_k, w_j) \prod_{(w_j, w_k) \in D^-} p(-|w_k, w_j)$$

Taking logarithms:

$$\arg\max \sum_{(w_j, w_k) \in D^+} \log p(+|w_k, w_j) + \sum_{(w_j, w_k) \in D^-} \log p(-|w_k, w_j)$$

Substituting the sigmoid formulations:

$$\arg\max \sum_{(w_j, w_k) \in D^+} \log \frac{1}{1 + e^{-c_k \cdot v_j}} + \sum_{(w_j, w_k) \in D^-} \log \frac{1}{1 + e^{c_k \cdot v_j}}$$

This objective can be optimized efficiently using stochastic gradient descent.

5.3.7 Properties of Embeddings

Word embeddings learned through these methods capture semantic similarity in meaningful ways. Words with related meanings have embeddings that are close together in the vector space. For example, country names like France, Austria, Belgium, Germany, Italy, Greece, Sweden, Norway, Europe, Hungary, and Switzerland form tight clusters. Religious concepts like Jesus, God, Sati, Christ, Satan, Kali, Indra, Vishnu, Ananda, Parvati, and Grace cluster together. Technology terms like Xbox, Amiga, PlayStation, MSX, iPod, Sega, PS2, HD, Dreamcast, and GeForce form another cluster.

Word embeddings also capture analogy relationships through vector arithmetic. In the analogy task, given that a is to b as c is to d, the system receives words a, b, and c, and must find d. For example, apple is to apples as car is to what? Or man is to woman as king is to what?

The solution is to capture analogy via vector offsets: $\mathbf{a} - \mathbf{b} \approx \mathbf{c} - \mathbf{d}$. The relationship man to woman should have a similar vector offset as king to queen: $\mathbf{man} - \mathbf{woman} \approx \mathbf{king} - \mathbf{queen}$. To find the answer d_w, we compute:

$$d_w = \arg\max_{d'_w \in V} \cos(\mathbf{a} - \mathbf{b}, \mathbf{c} - \mathbf{d}')$$

In high-dimensional embedding space, multiple relations can be embedded simultaneously for a single word. Empirical results show that embeddings capture a range of semantic relations. Geographic relations are captured, as in France minus Paris yields vectors close to Italy minus Rome, Japan minus Tokyo, and Florida minus Tallahassee. Morphological relations appear, as big minus bigger produces offsets similar to small minus larger, cold minus colder, and quick minus quicker. Geopolitical relations emerge, such as Sarkozy minus France being similar to Berlusconi minus Italy, Merkel minus Germany, and Koizumi minus Japan. Corporate relationships are captured, with Microsoft minus Windows similar to Google minus Android, IBM minus Linux, and Apple minus iPhone.

5.3.8 Practical Applications

Word2vec is frequently used for pretraining in other natural language processing tasks. It helps models start from an informed position by providing semantically meaningful initial representations. The method requires only plain text, of which large quantities are available, and is very fast and easy to use. Pretrained vectors trained on 100 billion words are publicly available. However, for optimal performance on specific tasks, it remains important to continue training and fine-tune the embeddings for the particular application domain.

Baroni and colleagues conducted a systematic comparison of count-based and context-predicting semantic vectors in their 2014 paper "Don't count, predict!" They compared these approaches across five types of tasks and 14 different datasets, including semantic relatedness, synonym detection, concept categorization, selectional preferences, and analogy recovery. Their results generally favored prediction-based methods. However, some of these findings were later disputed by Levy and colleagues in their 2015 paper "Improving Distributional Similarity with Lessons Learned from Word Embeddings," which demonstrated that well-tuned count-based methods could achieve competitive performance with prediction-based approaches.

Chapter 6

Compositional Semantics and Sentence Representations

6.1 Introduction to Compositional Semantics

The principle of compositionality is fundamental to understanding how meaning is constructed in natural language. According to this principle, the meaning of each whole phrase can be derived from the meaning of its constituent parts. This compositional nature of language implies that sentence structure itself conveys meaning beyond the individual words. Deep grammars that model semantics alongside syntax typically implement one semantic composition rule for each syntactic rule, creating a systematic mapping between structure and interpretation.

Despite the elegance of compositional principles, semantic composition presents several non-trivial challenges. Similar syntactic structures can yield different meanings, as exemplified by sentences like "it barks," "it rains," and "it snows," where the final two contain pleonastic pronouns that function differently from the subject pronoun in the first sentence. Conversely, different syntactic structures may convey identical meanings, as seen in passive constructions where "Kim ate the apple" and "The apple was eaten by Kim" express the same semantic content.

Not all phrases are interpreted compositionally, with idioms like "red tape" and "kick the bucket" serving as notable exceptions. However, these expressions can sometimes receive compositional interpretations as well, making it impossible to simply exclude them from compositional analysis. Additional meaning can arise through the process of composition itself, as demonstrated by logical metonymy where "fast programmer" versus "fast plane" involve different interpretations of "fast," or where "enjoy a book" implies reading while "enjoy a cup of tea" implies drinking. Meaning transfers and connotations can emerge through composition, particularly in metaphorical expressions such as "I can't buy this story" versus "This sum will buy you a ride on the train." The recursive nature of semantic composition further complicates matters, as evidenced by deeply nested structures like "Of course I care about how you imagined I thought you perceived I wanted you to feel."

Two primary approaches address compositional semantics. First, compositional distributional semantics models composition in a vector space using unsupervised methods to create general-purpose representations. Second, compositional semantics with neural networks employs supervised or self-supervised learning to produce typically task-specific representations.

6.2 Compositional Distributional Semantics

Extending distributional semantics to account for the meaning of phrases and sentences poses a fundamental challenge. Natural languages, given a finite vocabulary, license an infinite number

of sentences, making it impossible to learn vector representations for all possible sentences directly. However, distributional word representations can be leveraged by learning to perform semantic composition in distributional space.

6.2.1 Vector Mixture Models

Vector mixture models, introduced by Mitchell and Lapata in 2010, provide simple but surprisingly effective approaches to semantic composition. Two basic models are the additive and multiplicative models. In the additive model, the composed representation is simply the sum of the constituent word vectors:

$$p = u + v$$

where p represents the phrase vector and u and v represent individual word vectors. In the multiplicative model, the composition is performed through element-wise multiplication:

$$p = u \odot v$$

where $\odot$ denotes element-wise multiplication.

Consider a concrete example with simplified vectors. For words "old," "dog," "cat," "runs," and "barks" with respective vectors $[1, 4, 0]$, "old" is $[1, 0, 0]$, then composing "old dog" additively yields $[1, 4, 0]+[1, 0, 0] = [2, 4, 0]$, while multiplicatively it yields $[1, 4, 0]\odot[1, 0, 0] = [1, 0, 0]$. These models correlate with human similarity judgments about adjective-noun, noun-noun, verb-noun, and noun-verb pairs.

Both the additive and multiplicative models are symmetric and commutative, meaning they do not account for word order or syntactic structure. This limitation is evident in pairs like "John hit the ball" versus "The ball hit John," which receive identical representations despite having different meanings. These models are more suitable for modeling content words and would not apply well to function words such as conjunctions and prepositions, where word order matters significantly in distinguishing phrases like "some dogs," "lice and dogs," versus "lice on dogs."

6.2.2 Lexical Function Models

Lexical function models introduce a crucial distinction between two types of words. Some words, such as nouns, have meanings directly determined by their distributional profile. Other words, such as adjectives and adverbs, act as functions that transform the distributional profiles of other words.

Baroni and Zamparelli proposed in 2010 that nouns are vectors while adjectives are matrices, allowing for more sophisticated composition in semantic space. In this framework, adjectives are modeled as lexical functions applied to nouns. Specifically, adjectives are represented as parameter matrices A_{adj} and nouns as vectors $\mathbf{n}$, with composition performed through linear transformation:

$$\mathbf{p}_{\text{adj-noun}} = A_{\text{adj}} \times \mathbf{n}$$

For a concrete example, consider composing "old dog" where the adjective "OLD" is represented as a matrix:

$$\text{OLD} = \begin{bmatrix} 0.5 & 0 \\ 0.3 & 1 \end{bmatrix}$$

and the noun "dog" as a vector $\mathbf{dog} = [1, 5]^T$. The composition yields:

$$\text{old dog} = \begin{bmatrix} 0.5 & 0 \\ 0.3 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 5.3 \end{bmatrix}$$

Learning these adjective matrices involves several steps. First, a distributional vector $\mathbf{n}$ is obtained for each noun in the lexicon. Second, adjective-noun pairs (a, n) are collected

from a corpus. Third, a distributional vector $\mathbf{p}$ is obtained for each pair (a, n) from the same corpus using a conventional distributional semantic model. The set of tuples $(\mathbf{n}, \mathbf{p})$ represents a dataset D_a for adjective a. Finally, the matrix A_a is learned from D_a using linear regression by minimizing the squared error loss:

$$L = \sum_{(i,j) \in D_a} \|\mathbf{p}_{ij} - A_a \mathbf{n}_j\|^2$$

This approach creates separate training and test sets where training pairs like "old house," "old dog," "old car," "old cat," and "old toy" are used to learn the adjective matrix, which is then evaluated on unseen test pairs like "old elephant" and "old mercedes."

6.3 Compositional Semantics with Neural Networks

Neural network approaches to compositional semantics address two fundamental questions. First, how can a task-specific representation of a sentence be learned with a neural network? Second, how can predictions for a given task be made from that representation? These questions are central to applying neural architectures to natural language understanding tasks.

6.3.1 Task and Dataset: Sentiment Classification

Sentiment classification of movie reviews exemplifies task-specific sentence representation learning. Given a review such as "You'll probably love it," the goal is to classify it into one of five sentiment categories: very negative (0), negative (1), neutral (2), positive (3), or very positive (4). The learned representation must be specialized for sentiment analysis rather than serving as a general-purpose encoding.

The Stanford Sentiment Treebank (SST) serves as the primary dataset for this task, containing approximately 12,000 data points. Each data point includes a one-sentence review with a global sentiment score, a tree structure capturing syntax, and more detailed sentiment scores at the node level, allowing for compositional analysis of sentiment throughout the parse tree.

6.3.2 Bag of Words Model

The Bag of Words (BOW) model represents the simplest approach to sentence-level sentiment classification. This additive model does not account for word order or syntactic structure. Task-specific word representations with fixed dimensionality $d = 5$ are learned, where the dimensions of the vector space are explicit and interpretable, often corresponding to the five sentiment classes.

The BOW model computes sentence representations by summing word embeddings and adding a bias term:

$$\mathbf{o} = \sum_{i=1}^{n} \mathbf{e}_i + \mathbf{b}$$

where $\mathbf{e}_i$ represents the embedding of the i-th word and $\mathbf{b}$ is a bias vector. The predicted class is then:

$$\hat{y} = \arg\max(\mathbf{o})$$

Consider a concrete example with the sentence "this movie is stupid." If the embeddings are "this" = [0.0, 0.1, 0.1, 0.1, 0.0], "movie" = [0.0, 0.1, 0.1, 0.2, 0.1], "is" = [0.0, 0.1, 0.0, 0.0, 0.0], "stupid" = [0.9, 0.5, 0.1, 0.0, 0.0], and "bias" = [0.0, 0.0, 0.0, 0.0, 0.0], then summing these yields [0.9, 0.8, 0.3, 0.3, 0.1], and taking the argmax gives class 0 (very negative).

6.3.3 Continuous Bag of Words

The Continuous Bag of Words (CBOW) model extends the basic BOW approach by introducing a learned linear transformation. Instead of directly using the sum of embeddings, CBOW applies a weight matrix:

$$\mathbf{o} = W\left(\sum_{i=1}^{n} \mathbf{e}_i\right) + \mathbf{b}$$

where W is a learned weight matrix. This single linear layer allows the model to learn task-specific combinations of word features.

6.3.4 Deep Continuous Bag of Words

Deep CBOW introduces multiple layers of non-linear transformations to learn more complex features. The architecture stacks multiple layers with non-linear activation functions:

$$\mathbf{h}_1 = \tanh(W_1\mathbf{e}_{\text{sum}} + \mathbf{b}_1)$$

$$\mathbf{h}_2 = \tanh(W_2\mathbf{h}_1 + \mathbf{b}_2)$$

$$\mathbf{o} = W_3\mathbf{h}_2 + \mathbf{b}_3$$

where $\mathbf{e}_{\text{sum}} = \sum_{i=1}^{n} \mathbf{e}_i$ represents the summed word embeddings. The tanh activation function introduces non-linearity:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

This architecture remains an additive model that does not capture word order or syntax. The dimensions of the vector space are not interpretable, and with multiple layers and non-linear transformations, predictions cannot be easily traced back to specific input features.

6.3.5 Deep CBOW with Pre-trained Embeddings

Rather than learning word embeddings from scratch, pre-trained embeddings from models like Word2Vec or GloVe can be incorporated. These general-purpose word representations, learned from large corpora, can either be kept frozen (not updated during training) or fine-tuned (updated with the task-specific learning signal to become specialized). The architecture remains:

$$\mathbf{h}_1 = \tanh(W_1E\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}_2 = \tanh(W_2\mathbf{h}_1 + \mathbf{b}_2)$$

$$\mathbf{o} = W_3\mathbf{h}_2 + \mathbf{b}_3$$

where E represents the pre-trained embedding matrix and $\mathbf{x}$ is the input representation. Note that a bias term is added whenever multiplying with a weight matrix W.

6.3.6 Training Neural Networks

Neural networks are trained using Stochastic Gradient Descent (SGD), which iteratively updates model parameters. The training process involves several steps. First, a training example is sampled. Second, a forward pass computes network activations and produces the output vector. Third, the loss is computed by comparing the output vector with the true label using a loss function. Fourth, a backward pass using backpropagation computes the gradient of the loss with respect to learnable parameters (weights and biases). Fifth, a small step is taken in the opposite direction of the gradient to minimize the loss.

For categorical outputs with multiple classes, Cross Entropy loss is used:

$$\text{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log(\hat{y}_i)$$

In the sparse formulation where $\mathbf{y}$ is a one-hot vector with the true class at position k:

$$\text{SparseCE}(k, \hat{\mathbf{y}}) = -\log(\hat{y}_k)$$

For example, given an output vector after softmax $\hat{\mathbf{y}} = [0.0589, 0.0720, 0.0720, 0.7177, 0.0795]$ and target label $\mathbf{y} = [0, 0, 0, 1, 0]$ (class 3), the loss is $\text{SparseCE}(3, \hat{\mathbf{y}}) = -\log(0.7177)$.

The softmax function converts raw output scores (logits) into a probability distribution:

$$\text{softmax}(\mathbf{o})_i = \frac{\exp(o_i)}{\sum_j \exp(o_j)}$$

This ensures that the output values sum to 1.0, making them interpretable as probabilities. For prediction, only the argmax is needed, but for computing model loss during training, softmax combined with cross entropy is essential because argmax is not differentiable.

6.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) provide a powerful framework for handling sequential data. RNNs consist of multiple copies of the same network, with each copy passing a message to its successor. Unlike feedforward networks, RNNs take an input vector $\mathbf{x}$ and produce an output vector $\mathbf{h}$ that is influenced by the entire history of inputs. The internal state $\mathbf{h}$ is updated at every time step, and in the simplest case, this state consists of a single hidden vector.

6.4.1 RNN Architecture and Computation

RNNs model sequential data with one input $\mathbf{x}_t$ per time step t. For a sentence like "the cat sat on the mat," the RNN processes tokens sequentially:

$$\mathbf{h}_1 = f(\mathbf{x}_1, \mathbf{h}_0)$$

$$\mathbf{h}_2 = f(\mathbf{x}_2, \mathbf{h}_1)$$

$$\mathbf{h}_3 = f(\mathbf{x}_3, \mathbf{h}_2)$$

and so forth, where $\mathbf{h}_0$ is typically initialized to zeros.

The transition function f consists of an affine transformation followed by a non-linear activation:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) = \tanh(W\mathbf{x}_t + R\mathbf{h}_{t-1} + \mathbf{b})$$

where W is the weight matrix for the current input, R is the weight matrix for the previous hidden state, and $\mathbf{b}$ is a bias vector. The same matrices W and R are shared across all time steps, providing parameter efficiency and the ability to process sequences of arbitrary length.

When unfolded in time, the RNN reveals its sequential structure where the same transformation is applied at each step. Predictions can be made using argmax during inference, while training involves applying softmax and computing cross entropy loss before backpropagating gradients through time.

6.4.2 The Vanishing Gradient Problem

Simple RNNs suffer from the vanishing gradient problem, making them difficult to train effectively. During backpropagation through time (BPTT), gradients must be propagated backward through many time steps. At each step, gradients pass through multiplications with the recurrent weight matrix R and derivatives of non-linear activation functions like sigmoid or tanh.

Consider the recurrent matrix R containing entries with value $r = 0.5$. For an input at the first time step that needs gradient information from N steps later, the gradient is multiplied by 0.5 repeatedly:

$$\nabla \propto (0.5)^N$$

This leads to exponential decay:

$$\begin{aligned}(0.5)^5 &= 0.03\\ (0.5)^{10} &= 9 \times 10^{-4}\\ (0.5)^{15} &= 3 \times 10^{-5}\\ (0.5)^{20} &= 9 \times 10^{-7}\end{aligned}$$

As the number of time steps increases, gradients become vanishingly small, making it difficult for the network to learn long-range dependencies. The opposite problem, exploding gradients, occurs when matrix entries have values greater than 1, such as $r = 1.5$, causing gradients to grow exponentially large.

6.5 Long Short-Term Memory Networks

Long Short-Term Memory (LSTM) networks, introduced by Hochreiter and Schmidhuber in 1997, represent a special kind of RNN designed to deal with long-term dependencies by alleviating the vanishing gradient problem. LSTMs can capture relationships across long sequences, such as understanding that "I lived in France for a while when I was a kid so I can speak fluent..." should be completed with "French."

6.5.1 LSTM Core Ideas

LSTMs employ three key mechanisms to address the limitations of simple RNNs. First, they maintain a separate memory cell state $\mathbf{c}_t$ distinct from what is outputted, serving as long-term memory. Second, they use gates to control information flow: the forget gate removes irrelevant information, the input gate stores new relevant information from the current input, and the output gate returns a filtered version of the cell state. Third, backpropagation through time operates with partially uninterrupted gradient flow through the cell state.

6.5.2 LSTM Mathematical Formulation

While a simple RNN computes:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) = \tanh(W\mathbf{x}_t + R\mathbf{h}_{t-1} + \mathbf{b})$$

an LSTM computes both hidden state and cell state:

$$(\mathbf{h}_t, \mathbf{c}_t) = \text{LSTM}(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1})$$

The LSTM computation involves four components, each with its own parameters. The input gate determines what new information to store:

$$\mathbf{i}_t = \sigma(W_i\mathbf{x}_t + R_i\mathbf{h}_{t-1} + \mathbf{b}_i)$$

where σ denotes the sigmoid activation function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The forget gate decides what information to discard from the cell state:

$$\mathbf{f}_t = \sigma(W_f \mathbf{x}_t + R_f \mathbf{h}_{t-1} + \mathbf{b}_f)$$

Candidate values are extracted from the previous hidden state and current input:

$$\tilde{\mathbf{c}}_t = \tanh(W_c \mathbf{x}_t + R_c \mathbf{h}_{t-1} + \mathbf{b}_c)$$

The output gate determines what parts of the cell state to output:

$$\mathbf{o}_t = \sigma(W_o \mathbf{x}_t + R_o \mathbf{h}_{t-1} + \mathbf{b}_o)$$

The cell state is updated by first forgetting information according to the forget gate, then adding new candidate values scaled by the input gate:

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

where $\odot$ denotes element-wise multiplication.

Finally, the hidden state is computed by applying the output gate to a transformed version of the cell state:

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

Each gate has different parameters (weight matrices W, R, and bias vectors $\mathbf{b}$), allowing the network to learn specialized behaviors for each gating mechanism.

6.5.3 LSTM Cell State and Information Flow

The cell state $\mathbf{c}_t$ runs straight down the entire chain of LSTM cells with only minor linear interactions, providing a pathway for gradients to flow backward through time without vanishing. The LSTM can add or remove information from the cell state through the carefully regulated gate structures.

The forget gate first determines what to discard by computing a value between 0 and 1 for each element in the cell state, where 0 means "completely forget" and 1 means "completely retain." The candidate cell extracts new values that could potentially be added to the cell state. The input gate then decides which of these candidate values to actually store. The cell state update combines these operations: information marked for forgetting is removed through element-wise multiplication with the forget gate, while new information is added through element-wise multiplication of the input gate with the candidate values. Finally, the output gate decides what parts of the cell state to output by first applying tanh to put cell state values between -1 and 1, then multiplying by the output gate to filter the result.

6.5.4 LSTM Applications

LSTMs have achieved remarkable success across numerous NLP tasks. In language modeling, they capture long-range dependencies in text, as demonstrated by Mikolov et al. in 2010 and Sundermeyer et al. in 2012. For parsing, LSTMs effectively encode syntactic structure, with notable work by Vinyals et al. in 2015, Kiperwasser and Goldberg in 2016, and Dryer et al. in 2016. Machine translation systems by Bahdanau et al. in 2015 leverage LSTMs for sequence-to-sequence modeling. Beyond pure NLP, LSTMs enable image captioning as shown by Bernardi et al. in 2016, and visual question answering as demonstrated by Antol et al. in 2015, along with many other tasks.

6.6 Tree-Structured Models

While Bag of Words models produce order-independent sentence representations and sequence models like RNNs provide order-sensitive functions of word sequences, tree-structured models offer an alternative approach where sentence representations are functions of word representations that are sensitive to the syntactic structure of the sentence.

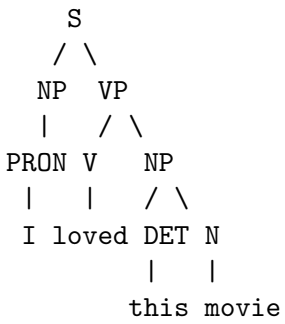
6.6.1 Exploiting Syntactic Structure

Rather than treating input as a flat sequence, tree-structured approaches assume a hierarchical parse tree and apply the principle of compositionality recursively. The meaning vector of a sentence is determined by two factors: the meanings of its constituent words and the rules that combine them according to syntactic structure.

Tree structure proves particularly helpful for disambiguation. Consider "He eats spaghetti with a spoon" versus "He eats spaghetti with meat." Despite similar surface forms, these sentences have different structures. In the first, the prepositional phrase "with a spoon" modifies the verb "eats," while in the second, "with meat" modifies the noun "spaghetti." These structural differences are captured by distinct parse trees.

6.6.2 Constituency Parsing

Constituency parse trees represent hierarchical phrase structure. For the sentence "I loved this movie," a constituency parse might reveal structure like:



where S denotes sentence, NP noun phrase, VP verb phrase, PRON pronoun, V verb, DET determiner, and N noun.

6.6.3 Recurrent versus Tree Recursive Neural Networks

RNNs process sequences left-to-right, computing representations like:

$$\mathbf{h}_1 = f(\text{loved})$$

$$\mathbf{h}_2 = f(\text{this}, \mathbf{h}_1)$$

$$\mathbf{h}_3 = f(\text{movie}, \mathbf{h}_2)$$

This sequential processing cannot capture phrases without prefix context and often over-weights the last words in the final vector representation.

Tree recursive neural networks instead follow the parse tree structure, requiring a parse tree for each sentence. Composition proceeds bottom-up through the tree, combining child representations to form parent representations. For "loved this movie," the tree structure would first combine "this" and "movie" to form the noun phrase representation, then combine "loved" with this noun phrase to form the complete sentence representation. Each node in the tree takes representations from its children as input and produces a parent representation as output.

6.7 Tree LSTM

Tree LSTMs, proposed independently by three research groups in 2015, generalize the LSTM architecture to tree structures. Tai, Socher, and Manning introduced improved semantic representations from tree-structured Long Short-Term Memory networks at ACL 2015, describing both Child-Sum and N-ary Tree LSTM variants. Le and Zuidema explored compositional distributional semantics with long short-term memory at SEM 2015, while Zhu, Sobihani, and Guo presented long short-term memory over recursive structures at ICML 2015.

6.7.1 Child-Sum Tree LSTM

The Child-Sum Tree LSTM sums over all children of a node, making it suitable for any number of children. This variant has three key properties: it does not impose any ordering on children, it works with a variable number of children, and it shares gate weights between all children. The formulation for a node j with children in set C_j is:

$$\tilde{\mathbf{h}}_j = \sum_{k \in C_j} \mathbf{h}_k$$

$$\mathbf{i}_j = \sigma(W_i \mathbf{x}_j + U_i \tilde{\mathbf{h}}_j + \mathbf{b}_i)$$

$$\mathbf{f}_{jk} = \sigma(W_f \mathbf{x}_j + U_f \mathbf{h}_k + \mathbf{b}_f)$$

$$\mathbf{o}_j = \sigma(W_o \mathbf{x}_j + U_o \tilde{\mathbf{h}}_j + \mathbf{b}_o)$$

$$\tilde{\mathbf{c}}_j = \tanh(W_c \mathbf{x}_j + U_c \tilde{\mathbf{h}}_j + \mathbf{b}_c)$$

$$\mathbf{c}_j = \mathbf{i}_j \odot \tilde{\mathbf{c}}_j + \sum_{k \in C_j} \mathbf{f}_{jk} \odot \mathbf{c}_k$$

$$\mathbf{h}_j = \mathbf{o}_j \odot \tanh(\mathbf{c}_j)$$

Note that children's outputs and memory cells are summed, and there is no fixed ordering of children. This makes Child-Sum Tree LSTM particularly suitable for encoding dependency trees.

6.7.2 N-ary Tree LSTM

The N-ary Tree LSTM uses separate parameter matrices for each child position, providing fine-grained control over how information propagates. This variant has three key properties: each node must have at most N ordered children (commonly binary trees), it provides fine-grained control on information propagation, and the forget gate can be parameterized with N matrices (one per child position) so that siblings can affect each other. For a node j with exactly N children indexed by $l \in \{1, \ldots, N\}$:

$$\mathbf{i}_j = \sigma\left(W_i \mathbf{x}_j + \sum_{l=1}^{N} U_i^{(l)} \mathbf{h}_{j_l} + \mathbf{b}_i\right)$$

$$\mathbf{f}_{jl} = \sigma\left(W_f \mathbf{x}_j + \sum_{k=1}^{N} U_f^{(l,k)} \mathbf{h}_{j_k} + \mathbf{b}_f\right)$$

$$\mathbf{o}_j = \sigma\left(W_o \mathbf{x}_j + \sum_{l=1}^{N} U_o^{(l)} \mathbf{h}_{j_l} + \mathbf{b}_o\right)$$

$$\tilde{\mathbf{c}}_j = \tanh\left(W_c \mathbf{x}_j + \sum_{l=1}^{N} U_c^{(l)} \mathbf{h}_{j_l} + \mathbf{b}_c\right)$$

$$\mathbf{c}_j = \mathbf{i}_j \odot \tilde{\mathbf{c}}_j + \sum_{l=1}^{N} \mathbf{f}_{jl} \odot \mathbf{c}_{jl}$$

$$\mathbf{h}_j = \mathbf{o}_j \odot \tanh(\mathbf{c}_j)$$

The N-ary formulation is particularly useful for encoding constituency trees and is implemented in practical applications. Standard LSTMs can be considered a special case of Tree-LSTMs where each node has exactly one child.

6.7.3 Transition Sequences for Tree Construction

Binary trees can be described using shift-reduce transition sequences, providing a method for constructing trees incrementally. For the sentence "I loved this movie," a transition sequence might be "S S R R R" where S denotes shift and R denotes reduce.

The construction process begins with an empty stack and a buffer (queue) containing all words. When encountering a shift (S) operation, the leftmost word is taken from the buffer and pushed onto the stack. When encountering a reduce (R) operation, the top two items are popped from the stack, combined using a Tree LSTM to create a new node with both hidden state $\mathbf{h}$ and cell state $\mathbf{c}$, and this new node is pushed back onto the stack.

For example, processing "I loved this movie" with transitions "S S R R R":

1. Start: stack = [], buffer = [I, loved, this, movie]
2. After S: stack = [I], buffer = [loved, this, movie]
3. After S: stack = [I, loved], buffer = [this, movie]
4. After R: stack = [Tree(I, loved)], buffer = [this, movie]
5. After S: stack = [Tree(I, loved), this], buffer = [movie]
6. After S: stack = [Tree(I, loved), this, movie], buffer = []
7. After R: stack = [Tree(I, loved), Tree(this, movie)], buffer = []
8. After R: stack = [Tree(Tree(I, loved), Tree(this, movie))], buffer = []

The final element on the stack is the root node, whose representation is used for classification.

6.7.4 Mini-batching with Tree LSTMs

Processing multiple sentences simultaneously requires careful handling of variable-length transition sequences. Consider two sentences: "I loved this movie" with transitions "S S S S R R R" and "It was boring" with transitions "S S S R R". To process these in a mini-batch, padding (PAD) is added to the shorter sequence, and operations are synchronized across examples. At each step, the transition dictates whether to shift a word onto the stack or reduce the top two stack elements using the Tree LSTM. This batched processing enables efficient training while respecting the distinct tree structure of each sentence.

6.7.5 Tree LSTM Variants Comparison

Child-Sum Tree-LSTM sums over hidden representations of all children with no imposed ordering, making it suitable for variable numbers of children. It shares parameters between children and is particularly appropriate for dependency trees where child ordering is not meaningful.

N-ary Tree-LSTM discriminates between child positions using a weighted sum with a fixed maximum branching factor (typically binary trees). It uses different parameters for each child

position, providing more expressive power for capturing asymmetric relationships. This variant is particularly suitable for constituency trees where left and right children have distinct syntactic roles.

6.7.6 Input Representations

In a Tree LSTM over a constituency tree, leaf nodes take the corresponding word vectors as input. These word embeddings can be randomly initialized and learned during training, or they can be pre-trained representations from models like Word2Vec or GloVe. Internal nodes do not have direct word inputs; instead, their representations are computed entirely from their children's hidden and cell states.

6.8 Summary

Compositional approaches to sentence representation span a spectrum from simple distributional methods to sophisticated neural architectures. Bag of Words models (BOW, CBOW, Deep CBOW) can encode sentences of arbitrary length but lose word order information. Sequence models like RNNs and LSTMs are sensitive to word order, with LSTMs addressing the vanishing gradient problem through gating mechanisms. The LSTM architecture employs input, forget, and output gates to control information flow and maintain long-term dependencies through the cell state. Tree-based models, including Child-Sum and N-ary Tree LSTMs, generalize the LSTM structure to hierarchical trees, exploiting compositionality according to syntactic structure. While tree models require parse trees as input, they provide representations that respect linguistic structure and enable fine-grained compositional analysis. Each approach offers different trade-offs between computational efficiency, structural sensitivity, and representational capacity, making them suitable for different natural language understanding tasks.

Chapter 7

Discourse Processing and Document Representations

This chapter covers the fundamental concepts of discourse processing in natural language processing, including discourse structure, document representation learning, and coreference resolution. The material explores how language extends beyond individual sentences to form coherent documents and how computational models can capture these higher-level linguistic structures.

7.1 Discourse Structure and Rhetorical Relations

Discourse structure refers to the organization and relationships between sentences and larger units of text within a document. Most documents exhibit implicit or explicit structure that varies by genre and purpose. Scientific papers typically follow conventional organizational patterns that differ across disciplines, while news stories often begin with a summary sentence. Understanding this structure is crucial for comprehensive text understanding.

7.1.1 Rhetorical Relations

Rhetorical relations describe the implicit relationships between sentences or clauses in discourse. Consider the sentence pair "Max fell. John pushed him." This can be interpreted in multiple ways depending on the underlying discourse relation. Under an EXPLANATION interpretation, the reading becomes "Max fell because John pushed him," where the second sentence explains the cause of the first event. Alternatively, under a NARRATION interpretation, it becomes "Max fell and then John pushed him," describing a temporal sequence of events.

These implicit relationships are called discourse relations or rhetorical relations. The words "because" and "and then" serve as cue phrases that make these relations explicit. However, in natural discourse, these relations often remain implicit, requiring inference from context.

The analysis of text using rhetorical relations typically produces a binary branching structure. This structure distinguishes between the nucleus, which represents the main phrase or central information, and the satellite, which represents subsidiary information. Relations like EXPLANATION and JUSTIFICATION exhibit this nucleus-satellite structure. For example, in "Max fell because John pushed him," the main clause "Max fell" serves as the nucleus while "because John pushed him" provides explanatory satellite information. Other relations assign equal weight to both components, as seen in NARRATION relations where "Max fell and Kim kept running" presents two events of comparable importance.

7.1.2 Discourse Coherence

Coherence refers to the property of discourse that makes it interpretable and meaningful as a connected whole. Discourses require connectivity to be coherent. The sentence pair "Kim got into her car. Sandy likes apples" lacks apparent coherence when presented in isolation. However, coherence can be established through appropriate context, as in "Kim got into her car. Sandy likes apples, so Kim thought she'd go to the farm shop and see if she could get some." This extended version creates connections that make the discourse interpretable.

Coherence assumptions significantly influence interpretation, particularly in pronoun resolution. Consider "John likes Bill. He gave him a nice Christmas present." Under an EXPLANATION interpretation, where the second sentence explains why John likes Bill, the pronoun "he" most naturally refers to Bill, who performed the gift-giving action that explains John's fondness. Under a JUSTIFICATION interpretation, where the second sentence supplies evidence for the first claim, "he" more likely refers to John, whose action of giving a gift demonstrates his affection for Bill.

"because"

"thus"

7.1.3 Factors Influencing Discourse Interpretation

Multiple factors contribute to how discourse structure and relations are interpreted. Cue phrases such as "because" and "and" provide explicit signals of discourse relations. Punctuation and text structure also play crucial roles in determining interpretation. Compare "Max fell (John pushed him) and Kim laughed" with "Max fell, John pushed him and Kim laughed." The parenthetical structure in the first version suggests a different grouping and interpretation than the comma-separated structure in the second.

Real world content knowledge influences discourse interpretation. In "Max fell. John pushed him as he lay on the ground," world knowledge about the physical sequence of events constrains interpretation. Tense and aspect provide temporal information that affects discourse structure. "Max fell. John had pushed him" uses the past perfect tense to indicate that John's pushing preceded Max's falling, while "Max was falling. John pushed him" uses progressive aspect to indicate that the pushing occurred during the falling action.

7.1.4 Discourse Parsing

Discourse parsing involves identifying the discourse structure and relations within a text. This represents a challenging problem in natural language processing. Much research focuses on the more tractable task of labeling relations between pairs of sentences or clauses rather than constructing complete discourse parse trees.

Two main approaches have been developed for discourse relation classification. Classification with hand-engineered features employs features such as punctuation patterns, cue phrases, syntactic structure, and lexical information to train classifiers that predict discourse relations between text segments. Neural models take a different approach by taking two sentences as input, training a sentence encoder to learn representations, and using these representations to predict the discourse relation between the sentences.

An alternative to explicit discourse parsing involves learning document representations through task-specific objectives, allowing the model to implicitly capture discourse structure without explicit relation labeling.

7.2 Learning Document Representations

Document representation learning addresses the challenge of encoding entire documents into fixed-dimensional vectors that capture their semantic content. This capability enables vari-

ous document classification tasks including text categorization by topic, sentiment analysis, authorship attribution, spam and phishing email filtering, and misinformation detection.

7.2.1 Bidirectional LSTM Architecture

The Bidirectional Long Short-Term Memory network extends the standard LSTM architecture to process sequences in both forward and backward directions. The forward LSTM computes hidden states according to $\overrightarrow{h}_t = \text{LSTM}_{\text{forward}}(\overrightarrow{h}_{t-1}, x_t)$, where x_t represents the input at time step t and $\overrightarrow{h}_{t-1}$ is the previous forward hidden state. Simultaneously, the backward LSTM computes $\overleftarrow{h}_t = \text{LSTM}_{\text{backward}}(\overleftarrow{h}_{t+1}, x_t)$, processing the sequence in reverse order. The final hidden state at each time step concatenates both directions: $h_t = [\overrightarrow{h}_t, \overleftarrow{h}_t]$. This bidirectional processing allows the model to incorporate context from both past and future tokens when computing representations.

7.2.2 Sentence Representation Strategies

Multiple strategies exist for deriving a fixed-dimensional sentence representation from the sequence of hidden states produced by an LSTM or BiLSTM. The simplest approach uses h_L, the final hidden state of the LSTM after processing all L tokens in the sentence. This representation theoretically encodes information about the entire sequence through the recurrent connections.

Mean-pooling computes an average of LSTM hidden states across all time steps, giving equal weight to each position in the sequence. This approach can be expressed as $h_{\text{mean}} = \frac{1}{L}\sum_{t=1}^{L} h_t$. Max-pooling instead takes the maximum value in each dimension across all hidden states, selecting the most salient features regardless of their position in the sequence.

Attention mechanisms provide a more sophisticated approach by computing a weighted sum of hidden states where the weights are learned by the model. The attention mechanism learns a weight vector w_α and computes its dot product with each hidden state transformed by a feed-forward neural network: $\alpha_t = w_\alpha \cdot \text{FFNN}_\alpha(h_t)$. These scores are normalized into a probability distribution using the softmax function: $a_t = \frac{e^{\alpha_t}}{\sum_{k=1}^{L} e^{\alpha_k}}$. The final sentence representation is computed as a weighted sum: $h_{\text{ATT}} = \sum_{t=1}^{L} a_t \cdot h_t$. This allows the model to dynamically focus on different parts of the sentence based on their relevance to the task.

7.2.3 Document Representation Architectures

Two primary architectural approaches exist for building document representations from word sequences. The flat approach feeds the entire document to an LSTM word by word, treating the document as a single long sequence. This architecture can incorporate word-level attention to identify which words contribute most to the document representation. The model processes tokens sequentially, potentially losing fine-grained sentence structure but maintaining direct connections between all words.

The hierarchical approach explicitly models document structure by first computing sentence representations and then combining them into a document representation. This two-level architecture processes each sentence independently with a sentence encoder, typically an LSTM or BiLSTM, to produce sentence-level representations. A second LSTM or attention mechanism then operates over these sentence representations to compute the final document representation. The entire architecture is trained end-to-end with a document-level objective such as classification loss.

7.2.4 Hierarchical Attention Networks

The Hierarchical Attention Network architecture, proposed by Yang et al. in 2016, implements a fully hierarchical approach with attention at both word and sentence levels. The architecture

begins with pretrained word embeddings as input, providing initial semantic representations for each token. An LSTM sentence encoder processes each sentence, computing hidden states for each word. Word-level attention operates over these hidden states to construct sentence representations, allowing the model to identify which words within each sentence are most important.

The sentence representations then serve as input to an LSTM document encoder. Sentence-level attention computes weights over the sentence representations to build the final document representation, enabling the model to determine which sentences contribute most to the document-level prediction. The attention weights at both levels are learned jointly during training with the document-level objective function.

The attention mechanism makes models interpretable by revealing which words and sentences the model considers most important for its predictions. In sentiment analysis of restaurant reviews, the model learns to focus on sentiment-bearing words like "delicious" and "amazing" within sentences, and to weight sentences containing strong sentiment expressions more heavily when constructing the document representation. For topic classification tasks, the attention distribution shifts to focus on topic-indicative terms and sentences containing key topical information.

The hierarchical architecture with dual attention mechanisms allows the model to capture document structure more naturally than flat approaches. Word-level attention identifies salient terms within each sentence's local context, while sentence-level attention captures the importance of different sentences to the overall document meaning. This structure aligns with the natural hierarchical organization of written text and enables effective learning on document classification tasks including sentiment analysis and text categorization.

7.3 Referring Expressions and Coreference

Referring expressions are linguistic elements that identify entities in the discourse. Understanding how language refers to entities and how multiple expressions can refer to the same entity is fundamental to discourse comprehension. Consider the text "Niall Ferguson is prolific, well-paid and a snappy dresser. Stephen Moss hated him — at least until he spent an hour being charmed in the historian's Oxford study."

The referent is the real-world entity that a piece of text or speech refers to, in this case the actual person Professor Ferguson. Referring expressions are the linguistic elements used to perform reference, including "Niall Ferguson," "him," "he," and "the historian." The antecedent is the text that initially introduces or evokes a referent, here "Niall Ferguson." Anaphora describes the phenomenon where a referring expression refers back to an antecedent. Cataphora is the less common phenomenon where pronouns appear before their referents in the text.

Definite descriptions like "a snappy dresser" and "the historian" also serve as referring expressions. These phrases presuppose the existence of entities and provide descriptions that allow readers to identify the intended referent. The interpretation of "a snappy dresser" as referring to Niall Ferguson depends on understanding that it functions as a predicate nominal describing the subject of the sentence.

7.3.1 Pronoun Resolution

Pronoun resolution is the task of identifying the referents of pronouns in text. Anaphora resolution generally considers cases where pronouns refer to antecedent noun phrases within the same discourse. In the example text, "him" refers to Niall Ferguson, while "he" is ambiguous and could refer to either Stephen Moss or Niall Ferguson depending on the interpretation of the discourse structure.

Pronoun resolution represents a crucial component of natural language understanding because the correct interpretation of pronouns is often essential for comprehending text meaning. Misidentifying pronoun referents can lead to fundamental misunderstandings of who performed what actions and how events relate to each other.

7.4 Algorithms for Coreference Resolution

Coreference resolution determines which referring expressions in a text refer to the same real-world entities. This task extends beyond pronoun resolution to identify all coreferent mentions, including proper names, definite descriptions, and other noun phrases.

7.4.1 Supervised Classification Approach

One approach to coreference resolution frames it as supervised binary classification. Training instances consist of potential pronoun-antecedent pairings, with the class being TRUE if the pairing is coreferent and FALSE otherwise. The training data must be labeled with correct coreference pairings. For each pronoun, candidate antecedents typically include all noun phrases in the current sentence and the preceding five sentences.

For example, in "Niall Ferguson is prolific, well-paid and a snappy dresser. Stephen Moss hated him — at least until he spent an hour being charmed in the historian's Oxford study," candidate antecedents for "him" would include "Niall Ferguson," "Stephen Moss," and potentially other noun phrases depending on preceding context. The classifier learns to predict which candidates are true antecedents based on features of the pronoun-antecedent pair.

7.4.2 Linguistic Constraints on Coreference

Several linguistic constraints restrict possible coreference relationships. Agreement in number and gender requires that pronouns match their antecedents on these features. "A little girl is at the door — see what she wants, please?" demonstrates gender agreement, where "she" must refer to a feminine antecedent. "My dog has hurt his foot — he is in a lot of pain" shows both gender and number agreement requirements.

Reflexive pronouns must be coreferential with a preceding argument of the same verb according to binding theory. In "John cut himself shaving," the reflexive pronoun "himself" must refer to the subject "John" rather than some other entity in the discourse. This constraint is syntactically determined and holds regardless of semantic plausibility.

Pleonastic pronouns are semantically empty and do not refer to any entity. "It is snowing" uses "it" as a pleonastic subject required by English syntax but lacking referential content. "It is not easy to think of good examples" similarly uses a pleonastic "it" that serves as a syntactic placeholder for the clausal subject "to think of good examples." Coreference resolution systems must identify and exclude pleonastic pronouns from consideration.

7.4.3 Factors Affecting Coreference Resolution

Beyond hard constraints, several factors influence which antecedent is most likely for a given pronoun. Recency preference indicates that more recent antecedents are generally preferred because they are more accessible in working memory. In "Kim has a big car. Sandy has a smaller one. Lee likes to drive it," the pronoun "it" most naturally refers to the more recent "one" (Sandy's car) rather than Kim's car.

Grammatical role follows a preference hierarchy where subjects are preferred over objects, which are preferred over other grammatical roles. In "Fred went to the shopping centre with Bill. He bought a CD," the pronoun "he" most naturally refers to Fred, the subject of the previous sentence, rather than Bill, who appears in a prepositional phrase.

Repeated mention effects indicate that entities mentioned multiple times in the discourse are more accessible and thus preferred as antecedents. This factor captures the intuition that discourse tends to maintain focus on particular entities across multiple sentences.

Parallelism preferences favor antecedents that share the same grammatical role as the pronoun in structurally parallel sentences. "Bill went with Fred to the lecture. Kim went with him to the bar" exemplifies this effect. The parallel structure suggests "him" refers to Fred because Fred occupies the parallel position in the first sentence to the pronoun's position in the second sentence.

Coherence effects arise from the discourse relations inferred between sentences. In "Bill likes Fred. He has a great sense of humour," the interpretation of "he" depends on which discourse relation is inferred. If the second sentence explains why Bill likes Fred, then "he" likely refers to Fred, whose sense of humor is the reason for Bill's fondness. If instead the second sentence provides additional information about Bill, then "he" refers to Bill.

7.4.4 Features for Classification

Supervised coreference resolution systems employ various features to represent pronoun-antecedent pairs. The cataphoric feature is binary, taking the value true if the pronoun appears before the antecedent in the text, which is unusual and typically marks special constructions. Number agreement is a binary feature indicating whether the pronoun is compatible with the antecedent in number. Gender agreement similarly indicates gender compatibility.

The same-verb feature is binary and indicates whether the pronoun and candidate antecedent are arguments of the same verb, which is relevant for reflexive binding and syntactic constraints. Sentence distance is a discrete feature measuring how many sentences separate the pronoun from the candidate antecedent, capturing the recency effect.

Grammatical role of the potential antecedent is represented as a discrete feature with values such as subject, object, or other, capturing the grammatical role hierarchy. The parallel feature is binary and indicates whether the antecedent and pronoun share the same grammatical role in their respective sentences. Linguistic form of the antecedent is a discrete feature distinguishing proper names, definite descriptions, indefinite descriptions, and pronouns, as these different types have different accessibility properties.

7.4.5 Limitations of Simple Classification

The pairwise classification approach has significant limitations. It cannot implement repeated mention effects because each classification decision is made independently without access to information about how many times an entity has been mentioned. Similarly, the model cannot use information from previously resolved coreference links, preventing it from building a coherent discourse model.

True coreference resolution requires reasoning about real-world entities that correspond to clusters of referring expressions rather than simply making pairwise decisions. An entity might be mentioned multiple times using different expressions, and these mentions form a coreference chain that should be identified as a coherent cluster. The pairwise approach lacks this global view of entity clusters.

7.4.6 Neural End-to-End Coreference Resolution

The neural end-to-end approach proposed by Lee et al. in 2017 addresses limitations of pairwise classification by implementing a mention-ranking paradigm. The model assigns each span i an antecedent y_i from the set $Y_i = \{1, \ldots, i-1, \epsilon\}$, where the empty token ϵ indicates that span i is either non-referential or discourse-new (first mention of an entity).

The model considers all text spans up to a certain length as potential mentions rather than relying on a separate mention detection system. This end-to-end approach allows the model to jointly learn mention detection and coreference resolution. For each pair of spans i and j, the model assigns a score $s(i, j)$ for their coreference link and computes a probability distribution over possible antecedents: $P(y_i) = \frac{e^{s(i,y_i)}}{\sum_{y' \in Y_i} e^{s(i,y')}}$.

The score $s(i, j)$ decomposes into three components: $s(i, j) = m(i) + m(j) + c(i, j)$, where $m(i)$ represents whether span i is a mention, $m(j)$ represents whether span j is a mention, and $c(i, j)$ represents whether j is a valid antecedent of i. The score $s(i, \epsilon)$ is set to zero, meaning the model predicts the antecedent with the highest positive score or abstains from making a prediction.

The scoring functions are computed from learned span representations. Each span i is represented by a vector $g_i = [h_{\text{START}(i)}, h_{\text{END}(i)}, h_{\text{ATT}(i)}, \phi(i)]$, which concatenates the hidden states at the start and end of the span, an attention-weighted representation of the span, and a feature vector $\phi(i)$ encoding the span length. The hidden states come from a bidirectional LSTM that processes the entire document.

The mention score is computed as $m(i) = w_m \cdot \text{FFNN}_m(g_i)$, where w_m is a learned weight vector and FFNN_m is a feed-forward neural network. The coreference score is computed as $c(i, j) = w_c \cdot \text{FFNN}_c([g_i, g_j, g_i \odot g_j, \phi(i, j)])$, where $\odot$ denotes element-wise multiplication and $\phi(i, j)$ encodes features such as the distance between spans in the text.

The span representation g_i includes an attention-weighted head representation $h_{\text{ATT}(i)}$ computed over the tokens within the span. This allows the model to identify which token within the span is most important for coreference decisions, typically focusing on the syntactic head of the noun phrase.

The model is trained end-to-end to maximize the marginal log-likelihood of all correct antecedent assignments. During inference, spans are typically pruned based on their mention scores to maintain computational efficiency, keeping only the most promising candidate mentions for coreference resolution. The model then predicts coreference links by selecting the highest-scoring antecedent for each mention or declaring it discourse-new if no antecedent receives a sufficiently high score.

This neural approach achieves strong performance while avoiding the need for hand-engineered features or separate mention detection systems. The attention mechanisms at both the span level (for identifying heads) and the coreference scoring level (for weighing different factors) provide some interpretability by showing which aspects of the input the model considers most relevant for its predictions. However, the model can make errors in challenging cases involving world knowledge, long-distance dependencies, or subtle discourse effects that require deep semantic understanding.

Chapter 8

Neural Sequence Modelling

8.1 Introduction to Sequence Modelling

Neural sequence modelling forms the backbone of many natural language processing tasks. These tasks involve conditioning on input text and predicting output sequences. Applications span diverse areas including part-of-speech tagging, named-entity recognition, machine translation, text summarisation, entity retrieval, and information extraction. While deploying systems for these tasks requires substantial expert knowledge about the task itself, datasets, and design decisions, most solutions share a common core architecture based on neural models of sequence prediction.

8.2 Sequence-to-Sequence Framework

The fundamental problem addresses modelling a specific relationship between pairs of sequences. We consider an input sequence x drawn from an input space $\mathcal{X}$ and an output sequence y drawn from an output space $\mathcal{Y}$. The relationship between these sequences is modelled directionally as $x \to y$ in a non-deterministic manner. Throughout this formulation, capital letters denote random variables (such as Y), lowercase letters represent their assignments (such as y), and calligraphic letters indicate sample spaces (such as $\mathcal{Y}$). The notation Y_j denotes a step in a random sequence, while $Y_{<j}$ represents a prefix sequence up until but not including Y_j. The distribution of Y is written as P_Y, and the distribution of Y given $X = x$ is $P_{Y|X=x}$. The probability of observing $Y = y$ given $X = x$ is expressed as $P(Y = y|X = x)$.

8.2.1 Probabilistic Modelling Framework

The output y is treated as an observation of a random variable Y, which is drawn conditionally given an observation x of the random variable X. The probability with which we observe $Y = y$ conditioned on $X = x$ is given by a parametric function with parameters θ:

$$P(Y = y|X = x) = f(y|x; \theta)$$

The primary modelling task involves designing this probability mass function. Once the pmf is specified, the subsequent steps address parameter estimation and using the model for predictions. Designing a pmf requires specifying the parametric family and selecting appropriate parameter values. The focus initially rests on specifying the parametric family, with the understanding that gradient-based optimisation techniques will be employed for parameter estimation.

8.3 Parameterisation of Conditional Probability Distributions

Parameterising conditional probability distributions for structured inputs and outputs presents two fundamental challenges. First, the input space $\mathcal{X}$ is typically very large and often infinite. Second, the output space $\mathcal{Y}$ is similarly large, being either infinite or growing combinatorially with the size of the input x.

8.3.1 Structured Input with Unstructured Output

Consider a simplified scenario where $\mathcal{Y} = \{1, \ldots, K\}$. To prescribe a conditional probability distribution for $Y|X = x$, we need K probabilities for any given $x \in \mathcal{X}$. For a single input x, storing K probability values is manageable. However, providing these probabilities for every possible $x \in \mathcal{X}$, including those never encountered during training, requires more sophisticated approaches.

8.3.2 Log-Linear Conditional Probability Distributions

Log-linear or logistic conditional probability distributions provide an elegant solution to this parameterisation challenge. The approach involves three key steps. First, map the input x to a fixed number of features $h(x) \in \mathbb{R}^D$. Second, map these features to K scores, also known as logits, for example through a linear transformation $Wh(x) + b$. Third, constrain the outputs to the probability simplex using appropriate normalisation. This construction maps any input x that can be featurised to a Categorical probability mass function. Crucially, regardless of how large $\mathcal{X}$ is, the parameterisation requires only $K \times D + K$ parameters.

8.3.3 Encoding Functions

The ability to encode an arbitrary input x into a D-dimensional space is essential for this parameterisation strategy. Prior to 2010, these functions were handcrafted feature functions designed by domain experts. In modern approaches, these encoding functions are part of the parameterisation itself. Neural networks are employed to represent the input and map it to output probability values.

8.3.4 Neural Text Classification Architecture

A neural text classifier for K-way classification can be formalised as follows. The conditional distribution is specified as $Y|X = x \sim \text{Cat}(g(x; \theta))$, where g is a neural network parameterised by θ. The encoder-decoder architecture proceeds as follows. Let $I = |x|$ denote the length of the input sequence. For each position $i = 1, \ldots, I$, compute the embedding $e_i = \text{embed}_D(x_i; \theta_{\text{inp}})$. Pass these embeddings through an LSTM to obtain hidden representations $r_{1:I} = \text{LSTM}_H(e_{1:I}; \theta_{\text{enc}})$. Apply average pooling to aggregate these representations $h = \text{avgpool}(r_{1:I})$. Compute output scores through a linear transformation $s = \text{linear}_K(h; \theta_{\text{out}})$. Finally, obtain probabilities via the softmax function $g(x; \theta) = \text{softmax}(s)$. The classification rule selects the class with maximum probability $\arg\max_{k \in [K]} g_k(x; \theta)$. The parameters θ collectively include the embedding matrix, the LSTM parameters, and the final linear transformation.

8.3.5 Structured Output Spaces

When the output space is structured, we exploit a decomposition into parts where each part is drawn from a relatively small sample space. For instance, in part-of-speech tagging, a tag sequence can be decomposed into individual word categories. Given an input $x = \langle I, am, going, home \rangle$, the corresponding output might be $y = \langle \text{PRP}, \text{VBP}, \text{VBG}, \text{NN} \rangle$, where each tag is generated sequentially.

Similarly, in machine translation, a translation decomposes into a sequence of target words. For the English input $x = \langle How, are, you, doing, ? \rangle$, the Dutch translation might be $y = \langle Hoe, gaat, het, ? \rangle$, with each target word produced one at a time.

8.3.6 General Autoregressive Factorisation

For a general input-output pair $x = \langle x_1, \ldots, x_I \rangle$ and $y = \langle y_1, \ldots, y_J \rangle$, the probability of the output sequence given the input decomposes via the chain rule:

$$P(Y = y|X = x) = \prod_{j=1}^{J} P(Y_j = y_j|X = x, Y_{<j} = y_{<j})$$

In this decomposition, conditioned on increasingly complex context, each part is drawn from a small sample space. The context for the j-th random variable comprises the input $X = x$ and the prefix $Y_{<j} = y_{<j}$. Models factorised in this manner are termed autoregressive.

8.3.7 Autoregressive Sequence Prediction

A general model of sequence prediction is obtained through repeated application of a shared text classifier. At each step j, the conditional distribution is specified as:

$$Y_j|X = x, Y_{<j} = y_{<j} \sim \text{Cat}(g(x, y_{<j}; \theta))$$

The function g maps from an input x and a partial output $y_{<j}$ to the probabilities of possible outcomes for the j-th step. Typically, the sample space remains constant across steps, such as the space of all tags or all words in the vocabulary.

8.3.8 Neural Part-of-Speech Tagger

A neural part-of-speech tagger reuses the same neural network $g(\cdot; \theta)$ repeatedly, as many times as there are steps in the input sequence. Each application maps from a growing context consisting of x and $y_{<j}$ to a probability distribution over the space of tags. The statistical model is formalised as:

$$Y_j|X = x, Y_{<j} = y_{<j} \sim \text{Cat}(g(x, y_{<j}; \theta))$$

The encoder-decoder architecture proceeds as follows. Let $I = |x|$ denote the input length. Compute word embeddings $e_i = \text{embed}_D(x_i; \theta_{\text{words}})$ for $i = 1, \ldots, I$. Encode the input sequence using a bidirectional LSTM $c_{1:I} = \text{BiLSTM}_H(e_{1:I}; \theta_{\text{enc}})$. For decoding at step j, embed the previous tag $t_{j-1} = \text{embed}_D(y_{j-1}; \theta_{\text{tags}})$. Update the decoder hidden state through $h_j = \text{rnnstep}_H(h_{j-1}, [c_j, t_{j-1}]; \theta_{\text{dec}})$. Compute output scores $s_j = \text{linear}_K(h_j; \theta_{\text{out}})$. Obtain probabilities via $g(x, y_{<j}; \theta) = \text{softmax}(s_j)$. The parameters θ include the embedding matrices for words and tags, the parameters of the BiLSTM encoder and the LSTM decoder, and the final linear transformation.

8.3.9 Neural Machine Translation

In neural translation, the same neural network $g(\cdot; \theta)$ is reused repeatedly, mapping from a growing context to a probability distribution over the vocabulary. Unlike tagging, where output length matches input length, in translation the output length is not predetermined. The process continues until a special terminating symbol is observed or generated. The statistical model is:

$$Y_j|X = x, Y_{<j} = y_{<j} \sim \text{Cat}(g(x, y_{<j}; \theta))$$

where the sample space now consists of all words in the target vocabulary.

8.3.10 Abstractive Text Summarisation

Text summarisation aims to generate a short version of a document or collection of documents containing the most important information. Extractive approaches identify and extract important sentences from the input, while abstractive approaches interpret the content and generate original summaries. Neural abstractive summarisation models, similar to translation, condition on input documents x and the prefix of already generated tokens $y_{<j}$ to parameterise a probability distribution over possible tokens for the j-th position. Advanced architectures can incorporate mechanisms that bias the model toward preferred solutions, such as occasionally performing extractive summarisation through pointer-generator networks.

8.4 Parameter Estimation

8.4.1 Data and Task Formalisation

The training data consists of a collection of pairs (x, y) where both x and y are sequences of outcomes from small discrete sets. The statistical task involves observing x and predicting a conditional distribution over all possible sequences. The natural language processing task maps a sequence x to a sequence y, often through $\arg\max_{y \in \mathcal{Y}} P(Y = y | X = x)$.

8.4.2 Maximum Likelihood Estimation

The statistical model specifies that the function g maps from x to a chain rule factorisation of the conditional distribution $Y|X = x$:

$$Y_j | X = x, Y_{<j} = y_{<j} \sim \text{Cat}(g(x, y_{<j}; \theta))$$

where θ collectively refers to all trainable parameters. The statistical objective is maximum likelihood of the model given a dataset of observations $\mathcal{D}$:

$$\mathcal{L}(\theta|\mathcal{D}) = \sum_{(x,y)\in\mathcal{D}} \log P(Y = y|X = x) = \sum_{(x,y)\in\mathcal{D}} \sum_{j=1}^{|y|} \log g_{y_j}(x, y_{<j}; \theta)$$

For concave $\mathcal{L}$ or convex negative log-likelihood, the algorithm finds θ such that $\nabla_\theta \mathcal{L}(\theta|\mathcal{D}) = 0$.

8.4.3 Gradient-Based Parameter Estimation

The optimisation problem $\nabla_\theta \mathcal{L}(\theta|\mathcal{D}) = 0$ lacks a closed-form solution. However, an optimum can be found via a fixed-point iteration $\theta \leftarrow \theta + \gamma \nabla_\theta \mathcal{L}(\theta|\mathcal{D})$ for learning rate $\gamma > 0$.

8.4.4 Stochastic Optimisation for Large Datasets

When the dataset is massive, computing $\nabla_\theta \mathcal{L}(\theta|\mathcal{D})$ requires time and memory that grow linearly with data size. Stochastic optimisation provides a solution by converging in finite time even with gradient estimates, provided they are unbiased and careful learning rate schedules are employed. If B is a random subset or mini-batch of $\mathcal{D}$, it holds that:

$$\nabla_\theta \mathcal{L}(\theta|\mathcal{D}) = \mathbb{E}_{B\sim\mathcal{D}}[\nabla_\theta \mathcal{L}(\theta|B)]$$

Thus iterative steps can be taken based on small random subsets of the data.

8.4.5 Loss Function Terminology

The negative of the log-likelihood is commonly referred to as the cross entropy loss. Alternative names include categorical cross entropy loss or softmax loss, though the latter is somewhat misleading since softmax is a vector-valued function rather than a loss. The terms categorical cross entropy and cross entropy are clear and commonly used in context.

8.5 Prediction and Inference

8.5.1 Decision Making Framework

After training, given an input x, the model outputs a representation of the entire probability distribution $P_{Y|X=x}$ over all of $\mathcal{Y}$. The prediction task involves mapping from this distribution to a single output y. This is often formulated as a search or discrete optimisation problem.

8.5.2 Most Probable Output

A common decision algorithm searches for the candidate output c assigned highest probability:

$$y^* = \arg\max_{c \in \mathcal{Y}} f(c|x;\theta)$$

This can equivalently be performed in log space as $\arg\max_{c\in\mathcal{Y}} \log f(c|x;\theta)$. For most models, including autoregressive models, this optimisation is intractable. Common approximations include greedy decoding, which iteratively applies argmax at each step, and beam search decoding, which maintains a fixed number of high-scoring candidates throughout the generation process.

8.5.3 Expected Utility Maximisation

An alternative decision-theoretic framework defines a utility function $u(y, c; x)$ quantifying the utility of prediction c when y is the true output for input x. A rational decision maker acts by maximising expected utility under the model:

$$y^* = \arg\max_{c \in \mathcal{Y}} \mathbb{E}[u(Y, c; x)]$$

Expected utility can be approximated via Monte Carlo sampling:

$$\mathbb{E}[u(Y, c; x)] \approx \frac{1}{K} \sum_{k=1}^{K} u(y^{(k)}, c; x)$$

where $y^{(k)} \sim P_{Y|X=x}$.

8.5.4 Candidate Enumeration Methods

Various methods exist for enumerating candidates during inference. Greedy decoding selects the highest probability token at each step. Beam search maintains multiple hypotheses throughout generation. Ancestral sampling generates samples by drawing from the predicted distributions. Top-p and top-k sampling restrict sampling to high-probability tokens. Sampling without replacement provides diverse outputs by avoiding duplicate sequences.

8.5.5 Evaluation Methodologies

Model evaluation proceeds along two dimensions. Statistical evaluation assesses how well the model fits the data through perplexity and statistics of model samples. Task-driven evaluation measures whether the model supports good decisions on benchmarks. For short generations such as question answering, entity linking, and information extraction, exact match, precision, recall, and F1 scores are commonly employed. For longer generations, string similarity metrics including BLEU, ROUGE, METEOR, and BEER are used. Semantic similarity metrics such as COMET and BLEURT provide learned evaluation functions.

8.6 Design Choices

8.6.1 Neural Architectures

Many architectural choices exist for encoders and decoders. Convolutional neural networks provide one option for sequence modelling. Graph convolutional networks enable incorporation of structured information. Transformers have emerged as the predominant architecture of choice in contemporary systems.

8.6.2 Transformer Architecture

Transformers are based on stacks of parallel self-attention heads and feed-forward networks. For any Transformer layer, the outputs $h_1^{(k+1)}, \ldots, h_\ell^{(k+1)}$ are such that each output $h_j^{(k+1)}$ depends on no other output $h_i^{(k+1)}$ and on at most all of the layer's inputs $h_1^{(k)}, \ldots, h_\ell^{(k)}$. In the encoder configuration, $h_j^{(k+1)}$ depends on all inputs $h_{1:\ell}^{(k)}$. In the decoder configuration, $h_j^{(k+1)}$ depends on inputs $h_{<j}^{(k)}$ prior to position j to ensure autoregressiveness.

8.6.3 Comparison with Recurrent Neural Networks

Recurrent neural networks exhibit fundamentally different properties. In a BiLSTM encoder, $h_j^{(k+1)}$ depends directly on $h_{j-1}^{(k+1)}$ and $h_{j+1}^{(k+1)}$, and hence recursively on all other outputs. In an RNN decoder, $h_j^{(k+1)}$ depends directly on $h_{j-1}^{(k+1)}$ and hence recursively on all $h_{<j}^{(k+1)}$, ensuring autoregressiveness.

RNNs are stateful while Transformers are stateless, with important implications for computation. During training, all inputs past and future are known. For Transformers, this enables parallelism by computing all outputs simultaneously. The recursive nature of RNNs imposes sequential processing. During testing, regardless of architecture, only past generated inputs are available for observation, making sequential computation unavoidable. These architectural differences also impact learning dynamics related to the chain rule of derivatives.

8.6.4 Alternative Factorisation Approaches

Numerous alternatives to the chain rule factorisation exist. Latent variable models employ marginalisation to overcome factorisation assumptions. The marginalised probability is:

$$P_{Y|X}(y|x) = \int \mathcal{N}(z|0, I) f(y|x, z; \theta)\, dz$$

where the autoregressive factorisation applies to $f(y|x, z; \theta)$. Non-autoregressive models eliminate sequential dependencies entirely. Conditional random fields for sequence labelling tasks factorise with strong bigram-like assumptions in an undirected manner, with probability

proportional to $\prod_{j=1}^{\ell} \exp(g(x, y_{j-1}, y_j; \theta))$. Hybrid approaches combine latent variable models with strong conditional independences. Energy-based models regress to a score without factorisation and normalise:

$$P_{Y|X}(y|x) = \frac{\exp(g(x, y; \theta))}{\sum_{y'} \exp(g(x, y'; \theta))}$$

8.6.5 Challenges Beyond Chain Rule

Strong conditional independence assumptions are often unrealistic for natural language. Marginals and normalising constants are typically intractable to compute, complicating both learning and prediction. The probability of observed data necessary for maximum likelihood training becomes intractable, and search and sampling become very difficult in energy-based models. These advanced topics are covered in depth in subsequent deep learning courses.

8.7 Conditional Inference with Missing Variables

8.7.1 Problem Formulation

Consider a typical sequence-to-sequence model parameterising a chain rule factorisation of the joint distribution of output random sequence Y given outcome $X = x$. For an output sequence of length J, the probability of any outcome $\langle y_1, \ldots, y_J \rangle$ is:

$$P(Y = \langle y_1, \ldots, y_J \rangle | X = x) = \prod_{j=1}^{J} P(Y_j = y_j | X = x, Y_{<j} = y_{<j})$$

The challenge arises when generating outcomes for Y_k given assignments of all other variables. For example, consider $X = x$ and observations $\langle Y_1 = \text{the}, Y_3 = \text{dog}, Y_4 = \text{EOS} \rangle$ with missing Y_2.

8.7.2 Solution via Conditional Probability

Let O denote the set of observed output random variables and o their observed values. Let U denote the set of unobserved variables. The probability that U takes value u given observations is:

$$P(U = u | O = o, X = x) = \frac{P(O = o, U = u | X = x)}{P(O = o | X = x)}$$

The numerator is the joint distribution directly accessible from the model. The denominator is the marginal of the numerator, marginalising over all possible assignments of U. For a vocabulary V, this requires computing $\sum_{t \in V} P(Y_1 = \text{the}, Y_2 = t, Y_3 = \text{dog}, Y_4 = \text{EOS} | X = x)$.

8.7.3 Computational Complexity

For an output sequence of length J and vocabulary size V, the computational complexity of evaluating the conditional probability of an assignment of Y_k given all other variables is $O(JV)$. Probabilities conditioning on past context are simple since they are directly predicted by the neural network. Probabilities conditioning on future context are much more difficult, requiring assessment of the joint probability for every possible assignment of the unobserved variable. Each joint probability requires J calls to the neural network $g(\cdot; \theta)$, one per token in the sequence. This computation must be performed V times, once per possible value of Y_k, yielding total complexity $O(JV)$.

Chapter 9

Large Language Models

9.1 The Transition to General-Purpose Models

The landscape of natural language processing underwent a fundamental transformation beginning around 2014 when deep learning methods began to dominate the field. This shift resulted in substantial performance improvements across numerous tasks, marking what many researchers consider a paradigm shift in how NLP systems are designed and deployed. While neural networks themselves had existed for decades prior to this revolution, the critical innovation was not the architecture itself but rather the way these networks were applied. The key conceptual breakthrough involved learning intermediate meaning representations directly in the process of end-to-end training for specific tasks, moving away from hand-crafted features that had characterized earlier approaches.

This evolution culminated in a departure from task-specific modeling toward general-purpose neural network sentence encoders. Rather than training separate models for each individual task, the field embraced the development of reusable encoders that could be applied across diverse NLP applications. Such general-purpose models accept input text, process it through a reusable encoder to produce representations for each sentence, and then pass these representations to task-specific models that generate the final outputs. This architecture offers two primary advantages. First, it improves performance by producing rich semantic representations that downstream NLP tasks can leverage. Second, it enhances data efficiency by providing robust models of sentence representation for language understanding tasks that may lack sufficient training data.

These general-purpose sentence encoders are expected to capture a wide range of linguistic phenomena. At the lexical level, they model word meanings and perform disambiguation based on context. They encode word order information and capture aspects of syntactic structure. At the semantic level, they learn compositional meaning construction and represent idiomatic or non-compositional phrase meanings. Additionally, they can encode connotation and social meaning, providing nuanced understanding beyond literal interpretation.

9.2 ELMo: Deep Contextualized Representations

The Embeddings from Language Models architecture, introduced by Peters et al. in 2018, represented an early instantiation of the general-purpose model paradigm. ELMo employs a bidirectional Long Short-Term Memory network as its core encoder, specifically utilizing a two-layer BiLSTM architecture. The model is trained with a language modeling objective that jointly maximizes the log likelihood of both forward and backward directions, enabling it to capture context from both sides of each word simultaneously.

During the pretraining phase, the BiLSTM processes input sequences and develops internal representations at multiple layers. When applied to downstream tasks, ELMo produces word

representations as a weighted sum of the hidden representations across all layers of the network. Importantly, these weights are not fixed but are learned specifically for each task during fine-tuning, allowing the model to emphasize different aspects of the representations depending on the requirements of the particular application.

The contributions of ELMo to the field were manifold. The contextualized nature of its word representations provided a level of disambiguation not achievable with static embeddings, as each token's representation depends on its surrounding context. The deep architecture enabled the model to capture linguistic information at various levels of abstraction, with lower layers encoding more syntactic information and higher layers capturing semantic content. ELMo delivered substantial performance improvements across many NLP benchmarks and initiated a paradigm shift toward sentence encoder pretraining. It also began the tradition of naming language models after Sesame Street characters, a convention that would continue with subsequent models.

9.3 BERT: Bidirectional Transformers

The introduction of BERT by Devlin et al. in 2019 marked another significant milestone in the development of large language models. BERT, which stands for Bidirectional Encoder Representations from Transformers, adopted the Transformer architecture rather than recurrent networks, enabling more efficient parallelization during training and better modeling of long-range dependencies.

9.3.1 Architecture

BERT consists of stacked Transformer encoder blocks, each comprising multi-head self-attention mechanisms followed by feed-forward neural networks. The architecture includes residual connections that facilitate gradient flow during backpropagation. The BASE variant of BERT contains 12 Transformer layers with 12 attention heads per layer, totaling approximately 110 million parameters. The LARGE variant scales up to 24 Transformer layers with 16 attention heads, containing roughly 340 million parameters. Notably, BERT does not employ weight sharing across layers, allowing each layer to specialize in capturing different aspects of linguistic structure.

9.3.2 Input Representation

BERT introduces special tokens to structure its inputs. The `[CLS]` token is prepended to every input sequence and its final hidden state serves as an aggregate representation of the entire sequence for classification tasks. The `[SEP]` token marks boundaries between distinct segments within a single input, enabling the model to process sentence pairs.

The complete input representation for any token is constructed as the sum of three distinct embedding components. Token embeddings provide the basic word-level representation, potentially using subword tokenization such as WordPiece to handle out-of-vocabulary items. Position embeddings encode the absolute position of each token in the sequence, allowing the model to capture word order information despite the position-invariant nature of the attention mechanism. Segment embeddings distinguish between different segments in multi-segment inputs, with all tokens from segment A receiving one embedding vector (denoted E_A) and all tokens from segment B receiving another (denoted E_B). This tripartite representation scheme enables BERT to simultaneously encode content, position, and segment information.

9.3.3 Pretraining Objectives

BERT employs two complementary pretraining tasks that enable bidirectional context modeling. The first task, masked language modeling, addresses the limitation of standard conditional language models that can only model context in one direction at a time. BERT randomly masks 15% of the input tokens and trains the model to predict these masked tokens based on bidirectional context. This approach draws inspiration from the cloze task used in psycholinguistics and educational assessment. By masking tokens during training, BERT forces the model to develop representations that integrate information from both left and right contexts simultaneously.

The second pretraining task, next sentence prediction, models relationships between sentences. During training, the model receives pairs of sentences where 50% of the pairs consist of consecutive sentences from the corpus and the remaining 50% consist of randomly paired sentences. The model must predict whether the second sentence actually follows the first in the original text. This objective encourages BERT to learn discourse-level relationships between sentences, which proves valuable for tasks such as question answering and natural language inference that require understanding connections between multiple sentences.

The overall pretraining loss combines these two objectives, summing the mean masked language modeling likelihood and the mean next sentence prediction likelihood. BERT was pretrained on large-scale corpora including the BooksCorpus, containing approximately 800 million words, and English Wikipedia, contributing roughly 2.5 billion words. This extensive pretraining on diverse text enables the model to develop broad linguistic knowledge.

9.3.4 Fine-tuning and Impact

After pretraining, BERT can be adapted to downstream tasks through fine-tuning. The same architectural structure used during pretraining is maintained, with only the output layers being task-specific. All parameters of the pretrained model are updated during fine-tuning rather than being frozen, allowing the model to adapt its representations to the specific requirements of each task. For instance, in question answering tasks using the SQuAD dataset, the question and paragraph are provided as two segments, and the model predicts start and end positions for the answer span. For named entity recognition, token-level predictions are made using the final hidden states. For sentence pair classification tasks such as those in MNLI, the final hidden state of the `[CLS]` token is passed through a classification layer.

The contributions of BERT to natural language processing were substantial. It advanced the state of the art across a wide range of NLP benchmarks and demonstrated the critical importance of bidirectional pretraining for language understanding. BERT reduced the need for highly specialized task-specific architectures, as the same pretrained encoder could be adapted to diverse tasks through relatively simple fine-tuning procedures. The model has become the most widely-used NLP model in research and practice, accumulating over 150,000 citations. Subsequent analysis by Tenney et al. in 2019 revealed that BERT rediscovers the classical NLP pipeline, with a linguistic hierarchy emerging across its layers where lower layers encode syntactic information and higher layers capture semantic and discourse-level content.

9.4 Generative Language Models: The GPT Family

While BERT focused on encoding and understanding language, the GPT family of models developed by OpenAI emphasized generation capabilities. Introduced by Radford et al. in 2019, the Generative Pretrained Transformer series (GPT, GPT-2, GPT-3, GPT-4, and subsequent versions) employed left-to-right language modeling rather than bidirectional encoding. This architectural choice enables these models to generate coherent text autoregressively, predicting each token based on all preceding tokens.

GPT models utilize the Transformer architecture with decoder blocks rather than encoder blocks. The original GPT was comparable in size to BERT BASE, but subsequent versions scaled dramatically in both parameter count and training data. A key insight motivating the GPT approach is that many tasks are already implicitly described within naturally occurring text data. For example, web text contains numerous instances of translation tasks, reading comprehension examples, and other linguistic phenomena presented with natural language instructions. The hypothesis underlying GPT development is that language models can learn to perform tasks directly from these natural language instructions found in web text, functioning as unsupervised multitask learners.

9.4.1 Instruction Tuning and Dialogue Optimization

The development of InstructGPT and ChatGPT represented refinements of the base GPT models to make them more controllable and useful for interactive applications. InstructGPT was specifically trained to follow instructions provided in prompts and generate detailed, relevant responses. ChatGPT further optimized the model for dialogue, enabling more natural conversational interactions where the model can ask follow-up questions, acknowledge mistakes, challenge incorrect premises, and reject inappropriate requests.

9.4.2 Reinforcement Learning from Human Feedback

A critical challenge in training generative language models is defining what constitutes "good" text, as quality criteria vary substantially depending on task and context. Creative writing may value novelty and surprise, while informational responses require factual accuracy. Additionally, model outputs should be safe, unbiased, and polite, considerations that extend beyond the basic language modeling objective of predicting the next word. Evaluating and balancing these factors requires a nuanced approach that standard maximum likelihood training cannot provide.

Reinforcement Learning from Human Feedback addresses this challenge through a three-stage process. The procedure begins with a pretrained language model that has been trained on large-scale text corpora. The second stage involves gathering human preference data and training a reward model. Specifically, annotators are provided with a set of prompts, and multiple language models generate different continuations for these prompts. Human evaluators then rank these continuations, producing a scalar score that numerically represents human preferences for different outputs. A reward model is trained to predict these preference scores, effectively learning to approximate human judgment.

The third stage fine-tunes the language model using reinforcement learning to optimize for the learned reward. At each iteration of this process, the language model receives a prompt x and generates a continuation y. The concatenation of prompt and continuation is passed to the reward model, which outputs a reward score r_θ. The language model is then updated to maximize this reward score across the current batch of training data. Critically, the optimization includes regularization to ensure that the per-token probability distributions do not diverge too drastically from those of the original language model, preventing the model from exploiting the reward model in ways that produce high scores but poor actual outputs.

9.5 Applications and Extensions

9.5.1 Multitask Learning and Zero-Shot Generalization

Instruction-tuned language models demonstrate remarkable capabilities for zero-shot task generalization. The T0 model, presented by Sanh et al. at ICLR 2022, exemplifies this approach through multitask prompted training. The model is trained on a mixture of NLP datasets spanning different task types, with each dataset associated with multiple prompt templates. These

templates reformulate the task in natural language, converting structured inputs and outputs into textual format. After training on this diverse mixture of prompted tasks, the model can generalize to new tasks specified through natural language prompts without requiring task-specific fine-tuning.

9.5.2 Multilingual Language Models

Multilingual large language models pursue the goal of developing a single model that captures universal language structures and can reason across all known languages. Nearly all languages exhibit certain commonalities, such as distinguishing between nouns and verbs and separating function words from content words. Multilingual models aim to identify and exploit these commonalities at lexical, syntactic, and semantic levels while simultaneously balancing language-agnostic and language-specific information.

The fundamental intuition underlying multilingual models is that phrases with similar meanings should obtain similar representations regardless of the language in which they are expressed. This principle must be realized without compromising monolingual semantic relationships within each language. Various architectures have been developed to achieve multilingual representation, ranging from five-layer LSTM models such as LASER to mid-size Transformer models like Multilingual BERT, XLM, and XLM-R, and large-scale Transformer models including BLOOM, XGLM, mT5, Aya, and CommandR. These models can encode up to 110 languages within a single unified architecture.

Pretraining approaches for multilingual models combine monolingual and cross-lingual objectives. Monolingual pretraining typically employs masked language modeling or generative language modeling on text in each language independently. Cross-lingual pretraining introduces tasks such as sentence translation or translation language modeling, where the model must predict masked tokens in one language using context from another language, forcing it to align representations across languages.

The typical application workflow for multilingual models involves three stages. First, a multilingual language model is pretrained on large corpora spanning many languages. Second, the model is fine-tuned on one or more high-resource languages for a specific task, producing a task-specific model. Third, this model is applied in zero-shot or few-shot transfer to other languages, particularly low-resource languages where task-specific training data may be scarce or nonexistent. The success of such transfer varies considerably, with high performance observed for typologically similar languages but substantially lower performance for typologically distant and low-resource languages.

9.6 Outstanding Challenges

Despite remarkable progress in large language model development, numerous challenges remain unresolved and represent active areas of research. Interpretability remains a fundamental concern, as understanding what linguistic and conceptual knowledge these models encode and how they process information continues to be difficult. Better learning algorithms are needed, particularly for continual learning scenarios where models must acquire new knowledge without catastrophically forgetting previously learned information.

Low-resource languages remain underserved by current models, as the data-hungry nature of large-scale pretraining disadvantages languages with limited digital corpora. The balance between generalization and memorization represents another critical challenge, as models must learn general patterns while avoiding simple memorization of training data that could lead to privacy concerns and poor generalization. Common sense reasoning and the development of coherent world models remain areas where current language models show limitations, often producing outputs that are fluent but factually incorrect or logically inconsistent.

Finally, ethics and alignment constitute perhaps the most pressing challenges for the field. Ensuring that large language models behave in accordance with human values, do not perpetuate or amplify societal biases, and can be safely deployed in high-stakes applications requires ongoing research spanning technical machine learning, ethics, and policy considerations. These topics continue to drive research in advanced natural language processing courses and represent the frontier of current investigation in the field.