Introduction to Natural Language Processing

From Symbolic Rules to Deep Representations

Paul MINCHELLA

November 2025

Master MIASHS

Contents

Chapter 1

Context and Motivation of Natural Language Processing (NLP)

1.1 Introduction: What is Natural Language Processing?

Natural Language Processing (NLP) is a field at the intersection of linguistics, computer science, and applied mathematics whose objective is to enable machines to *understand*, *generate*, and *reason about* human language. Unlike structured numerical data, linguistic data are inherently ambiguous, contextual, and dynamic. Words can have multiple meanings depending on their usage, and the structure of human discourse is far from deterministic.

NLP therefore aims to design mathematical models capable of capturing both the **syntactic** structure and the **semantic** content of language, in order to perform tasks such as:

- automatic translation,
- information extraction,
- text classification and sentiment analysis,
- dialogue systems and question answering,
- automatic summarization and text generation.

The key challenge is to transform sequences of words – highly complex symbolic structures – into numerical representations suitable for computation and learning. This transformation constitutes the central motivation of modern NLP: *the search for a meaningful vector representation of language.*

1.2 Why NLP Matters

In the last two decades, the amount of unstructured text data has exploded: emails, social media, medical reports, legal documents, scientific literature, and so on. While numerical data are easily manipulated by algorithms, text data require an intermediate level of abstraction – a *representation layer* that connects human meaning and mathematical computation.

NLP allows:

- **Automation of knowledge extraction**: detecting key entities, topics, or relations within large corpora.
- **Decision support**: transforming textual information into features usable by machine learning models.
- **Human-computer interaction**: enabling machines to communicate using natural language.
- **Language understanding**: bridging the gap between statistical learning and cognitive interpretation.

Consequently, NLP is now a cornerstone of artificial intelligence. It powers applications ranging from intelligent search engines to clinical decision systems. The modern era of NLP is characterized by a strong integration with deep learning and probabilistic modeling.

1.3 The Representation Problem: From Symbols to Vectors

The central goal of modern NLP can be expressed as the **representation problem**:

How can one map linguistic entities (words, sentences, documents) to mathematical objects that preserve their semantic relationships?

Why Representations Matter ? A machine learning model can only manipulate numerical data. To process language, we must embed it in a vector space where operations such as distance and similarity make sense. This embedding allows one to compare words, measure analogy, and transfer knowledge across contexts.

Example. If vectors $v_{\text{king}}, v_{\text{man}}, v_{\text{woman}}$ are well structured, we may observe:

$$v_{\text{king}} - v_{\text{man}} + v_{\text{woman}} \approx v_{\text{queen}}.$$

Such relationships show that geometric structure can encode semantic regularities.

Motivation of the Embedding Paradigm. From symbolic rules to probabilistic counts, every historical stage of NLP has aimed – implicitly or explicitly – at building representations:

Era	Representation Type	Goal
Symbolic (1950–1990)	Grammars, trees, logic	Capture structure
Statistical (1990–2010)	Probabilities, frequencies	Capture local dependencies
Neural (2010–...)	Continuous vectors (embeddings)	Capture meaning and context

Modern NLP therefore rests on a simple but profound principle: *Understanding language leads to establish (through learning) useful representations of it.*

The historical trajectory of NLP reflects a gradual move from explicit symbolic manipulation toward implicit representation learning. Each paradigm – symbolic, statistical, neural – has contributed to a deeper understanding of how to model language computationally.

Today, the embedding paradigm dominates: text is transformed into structured numerical spaces where geometry mirrors semantics. This shift not only enables better predictive models but also provides a foundation for reasoning, transfer learning, and interpretability in language understanding.

The following chapters will explore how such representations are learned, how loss functions (especially softmax-based likelihoods) govern this learning, and how probabilistic and energy-based formulations connect NLP to broader principles of statistical modeling.

Chapter 2

A Brief History of Natural Language Processing

2.1 The Symbolic Era (1950–1990)

Overview. The early decades of Natural Language Processing were dominated by a **symbolic** and **rule-based** paradigm. Language was conceived as a system of logical and grammatical rules, inspired by the rise of formal linguistics (Chomsky, 1957) and early computational logic. At that time, researchers believed that if all syntactic and semantic rules of a language could be explicitly defined, a computer could manipulate and understand human language deterministically.

Symbolic NLP systems operated on discrete symbols rather than numerical data. They represented linguistic knowledge using grammars, lexicons, ontologies, and logical formalisms. The main idea was that each linguistic phenomenon (e.g., noun phrase, verb phrase, clause) could be captured by explicit production rules or logical predicates.

Grammatical and Rule-Based Systems. The most influential frameworks included:

- **Context-Free Grammars (CFG):** Introduced by Noam Chomsky, CFGs defined sentences as hierarchical structures built from recursive rewriting rules:

  $$S \to \text{NP VP}, \quad \text{NP} \to \text{Det } N, \quad \text{VP} \to V \text{ NP}.$$

  where:

  - S stands for the **Sentence** symbol – the root of the syntactic tree.
  - NP denotes a **Noun Phrase**, typically consisting of a determiner followed by a noun (e.g., "the cat").
  - VP denotes a **Verb Phrase**, often consisting of a verb followed by a noun phrase (e.g., "chased the mouse").
  - Det represents a **Determiner**, such as "the", "a", or "an".
  - N denotes a **Noun**, such as "cat", "dog", or "house".
  - V denotes a **Verb**, such as "chased", "saw", or "found".

- **Transformational Grammars:** Attempted to explain variations in sentence structure through transformations (e.g., active/passive voice).

- **Logic-Based Representations:** Systems such as *Prolog* encoded linguistic relations and inference mechanisms in first-order logic.

This period saw pioneering applications such as:

- *ELIZA* (Weizenbaum, 1966): a chatbot simulating psychotherapy via pattern matching.
- *SHRDLU* (Winograd, 1972): a system manipulating blocks in a virtual world using natural commands.
- *SYSTRAN*: one of the first machine translation systems, based on handcrafted grammar rules.

Strengths and Limitations. Symbolic systems captured deep syntactic structure and logical reasoning, but they suffered from several severe limitations:

- **Lack of scalability:** Thousands of linguistic rules were required, often inconsistent and difficult to maintain.
- **Fragility:** Rule-based systems failed on slightly unexpected input.
- **No generalization:** The model could not infer patterns from data; all knowledge had to be explicitly encoded.

Consequently, as corpora and computational power increased, the field gradually shifted toward data-driven and probabilistic methods, where patterns were learned automatically from examples rather than manually programmed.

2.2 The Statistical Era (1990–2010)

The Rise of Probability in Language Modeling. In the 1990s, the field underwent a profound paradigm shift: language began to be treated as a **stochastic process**. Instead of relying on handcrafted rules, researchers sought to model the *probability* of word sequences and linguistic structures.

The foundational idea is that any sentence $w_1, w_2, \ldots, w_T$ can be represented by its joint probability:

$$\mathbb{P}(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} \mathbb{P}(w_t \mid w_1, \ldots, w_{t-1}),$$

where each conditional probability expresses how likely a word is, given its preceding context.

N-Gram Models and the Markov Assumption. Because computing dependencies over entire histories is infeasible, the **Markov assumption** was introduced:

$$\mathbb{P}(w_t \mid w_1, \ldots, w_{t-1}) \approx \mathbb{P}(w_t \mid w_{t-n+1}, \ldots, w_{t-1}).$$

This assumption leads to the well-known *n-gram* models (Shannon, 1948; Chen and Goodman, 1999):

$$\mathbb{P}(\text{sentence}) = \mathbb{P}(w_1)\mathbb{P}(w_2 \mid w_1)\mathbb{P}(w_3 \mid w_1, w_2)\ldots,$$

$$\mathbb{P}(w_t \mid w_{t-1}) \text{ (bigram model)},$$

$$\mathbb{P}(w_t \mid w_{t-2}, w_{t-1}) \text{ (trigram model)}.$$

These models were estimated by counting word co-occurrences in large corpora. To avoid zero probabilities for unseen combinations, smoothing methods such as Laplace, Good–Turing, or Kneser–Ney smoothing were introduced.

N-gram models provided a robust probabilistic foundation for speech recognition, text generation, and machine translation, and they became the backbone of statistical NLP for nearly two decades.

Hidden Markov Models (HMM). For tasks where latent structures underlie observable words – such as part-of-speech tagging or speech recognition – **Hidden Markov Models** (HMM) (Rabiner, 1989) were employed. HMMs assume a sequence of hidden states $S_1, S_2, \ldots, S_T$ generating observed tokens $O_1, O_2, \ldots, O_T$:

$$\mathbb{P}(O, S) = \mathbb{P}(S_1) \prod_{t=2}^{T} \mathbb{P}(S_t \mid S_{t-1})\mathbb{P}(O_t \mid S_t).$$

Each state corresponds to a linguistic category (e.g., noun, verb), and emissions model the likelihood of words given their category. Training is achieved using the *Baum–Welch algorithm* (a form of Expectation–Maximization), while decoding typically uses the *Viterbi algorithm* to find the most probable state sequence.

Discriminative and Feature-Based Models. By the early 2000s, the field moved toward **discriminative models** that directly estimate $\mathbb{P}(y \mid x)$ rather than modeling joint distributions $\mathbb{P}(x, y)$. Two landmark approaches emerged:

- **Maximum Entropy Models (Berger et al., 1996):** Derived from the principle of maximum entropy, they represent conditional probabilities as:

$$\mathbb{P}(y \mid x) = \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right),$$

  where f_i are feature functions and λ_i their learned weights.

- **Conditional Random Fields (CRF):** Extended this idea to sequence labeling by incorporating dependencies between output variables.

These models allowed for rich, handcrafted features, integrating lexical, syntactic, and contextual cues into a unified probabilistic framework.

Achievements and Limitations. Statistical NLP represented a crucial breakthrough:

- Models were trained automatically from large corpora.
- Probabilistic reasoning handled ambiguity quantitatively.
- Data-driven evaluation metrics (e.g., perplexity, accuracy) emerged.

However, several challenges persisted:

- Sparse data problems due to the combinatorial explosion of possible word sequences.
- Limited context: n-gram models capture only local dependencies.
- Lack of semantics: words are treated as discrete symbols; "cat" and "dog" remain orthogonal.

These limitations motivated the next revolution: learning continuous, distributed representations of language.

2.3 The Neural Revolution (2010–Present)

From Symbols to Continuous Representations. Around 2010, advances in deep learning radically transformed NLP. Instead of modeling discrete probabilities over symbols, researchers began representing words, sentences, and documents as points in a continuous vector space. This marked the emergence of **word embeddings**, where meaning is encoded geometrically. The key principle:

> Words appearing in similar contexts should have similar vector representations.

This idea connects linguistics (the distributional hypothesis) with neural computation and optimization.

The Word2Vec Breakthrough. The **Word2Vec** framework (Mikolov et al., 2013) proposed two simple yet powerful architectures:

- **Continuous Bag of Words (CBOW):** predicts a target word from its context.
- **Skip-Gram:** predicts context words given a target word.

Both are trained to maximize the conditional likelihood:

$$\mathcal{L} = \sum_t \sum_{-c \le j \le c, j \neq 0} \log \mathbb{P}(w_{t+j} \mid w_t),$$

where the probability is computed via a softmax:

$$\mathbb{P}(w_O \mid w_I) = \frac{\exp(v_{w_O}^\top v_{w_I})}{\sum_w \exp(v_w^\top v_{w_I})}.$$

Through optimization, the embeddings v_w capture latent semantic relationships – such as gender, tense, or analogies – without explicit supervision.

From Word-Level to Contextual Representations. Later architectures extended this concept:

- **GloVe** (Pennington et al., 2014) introduced a global co-occurrence factorization.
- **FastText** (Bojanowski et al., 2017) integrated subword information, improving morphology.
- **ELMo** (Peters et al., 2018) produced contextual embeddings using bidirectional LSTMs.
- **Transformers** (Vaswani et al., 2017) and **BERT** (Devlin et al., 2019) leveraged attention mechanisms to produce deep contextualized embeddings.

The concept of representation evolved from static word vectors to dynamic, context-dependent embeddings capable of capturing polysemy and long-range dependencies.

Main Motivation of the Neural Era. The neural revolution unified language modeling, representation learning, and optimization under a single differentiable framework. The main motivation can be summarized as follows:

> **To understand language, one must learn to represent it in a form suitable for computation and generalization.**

Instead of designing rules or counting frequencies, models now learn representations automatically by minimizing a loss function – typically derived from the log-likelihood of observed text.

This approach bridges linguistics, geometry, and statistics:

- **Linguistics:** meaning arises from context.
- **Geometry:** semantic relations are encoded as distances in vector space.
- **Statistics:** learning corresponds to optimizing probabilistic consistency with data.

Conclusion of the Historical Perspective. The history of NLP reflects a continuous evolution of paradigms:

Era	**Representation**	**Core Idea**	**Main Limitation**
Symbolic (1950–1990)	Logical rules	Handcrafted syntax and semantics	No generalization; rules do not scale to real-world variability.
Statistical (1990–2010)	Probabilistic counts	Learning from data frequencies; use of $\mathbb{P}(w_t \mid w_{t-n+1}, \ldots, w_{t-1})$ to capture local dependencies	No semantics; fails to represent meaning or context.
Neural (2010–Nowadays)	Continuous embeddings	Learning distributed meaning via differentiable representations and optimization	Data- and computation-intensive; interpretability remains limited.

Each stage, despite its limitations, prepared the ground for the next. Today, the focus of NLP has clearly shifted toward **representation learning** – the quest for embeddings that encode linguistic, semantic, and even world knowledge in high-dimensional spaces. This motivation will guide the remainder of the course, where we will study how such representations are learned, optimized, and applied.

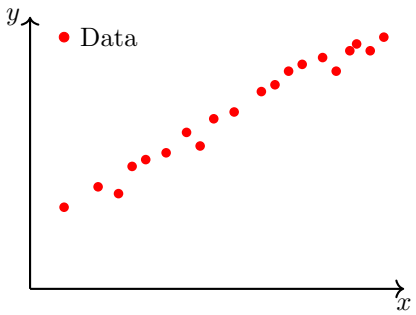
Chapter 3

Understanding the Learning Process of a Model

Before going back into the details of NLP processes, let's recall the fundamental principles of learning a statistical model. Every predictive system in machine learning–from linear regression to deep neural networks–relies on the same conceptual foundation: defining a model, choosing a *loss function* that quantifies its errors, and adjusting parameters to minimize this loss. This chapter introduces these key ideas in a simple yet rigorous way, before applying them to natural language data.

3.1 Understanding a Model Means Understanding Its Loss

Let's suppose you dispose to these data points:



A model can be viewed as a mathematical function

$$f_\theta : \mathcal{X} \to \mathcal{Y},$$

where θ represents its parameters (weights, biases, etc.). Given a dataset $\{(x_i, y_i)\}_{i=1}^n$, the goal of learning is to find θ such that $f_\theta(x_i)$ approximates the true output y_i as closely as possible.

An example. Suppose we have a cloud of points in the plane representing a simple relationship between two variables (x, y). We can approximate this relation with a linear model

$$\hat{y} = \theta_0 + \theta_1 x.$$

The quality of this approximation is measured by the **residual energy**, i.e., the sum of squared differences between predicted and observed values:

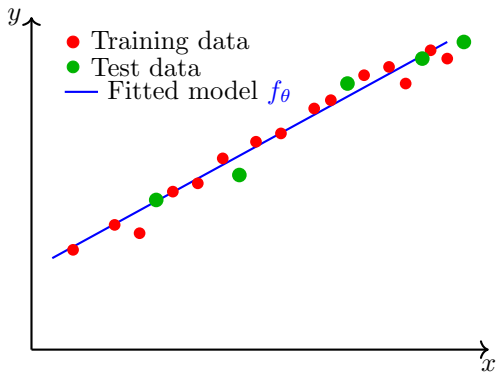
$$\mathcal{L}(\theta_0, \theta_1) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

Minimizing this **loss function** corresponds to finding the line that best fits the data in the least-squares sense.

Illustration: Train-Test Split and Loss Evaluation. For **supervised learning**, in practice, the available data are divided into two subsets:

- the **training set** (about 80% of the data) used to fit the model,
- the **test set** (about 20%) used to evaluate its generalization ability.

The following diagram illustrates this concept: red points represent the training samples used to estimate the parameters, while green points correspond to unseen test samples. The fitted line minimizes the residual energy on the training set.



The residual energy $\mathcal{L}(\theta)$ thus quantifies how far predictions are from the ground truth. The model that minimizes $\mathcal{L}$ is the one that best explains the data under the chosen representation.

3.2 Learning Challenges and Generalization

A central question in machine learning is not merely how to fit the data, but how to **generalize** beyond it. If the model perfectly fits the training points but performs poorly on unseen data, it is said to **overfit**. Conversely, if it captures only a very rough trend and underperforms even on the training data, it **underfits**.

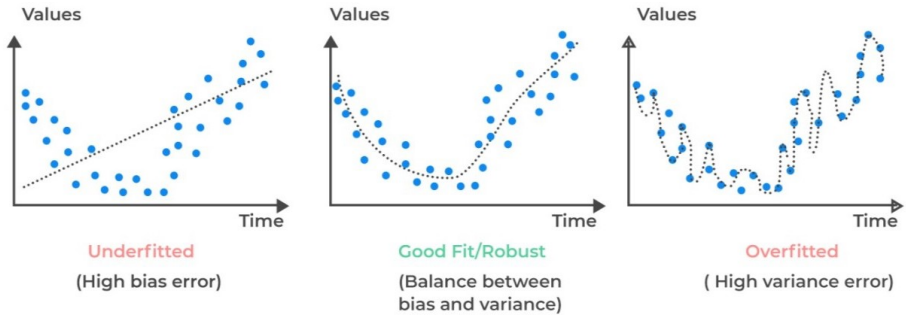


Figure 3.1: Illustration of the bias–variance trade-off. On the left, the model is **underfitted**: it is too simple to capture the underlying structure of the data, leading to a high *bias* error. In the middle, the model achieves a **good fit**, balancing bias and variance for robust generalization. On the right, the model is **overfitted**: it learns noise and fluctuations specific to the training set, resulting in a high *variance* error when applied to new data.

Bias-Variance Trade-off. This balance between fitting and generalizing is often formalized as the **bias–variance trade-off**:

- High bias: the model is too rigid, incapable of capturing complexity.
- High variance: the model is too flexible, memorizing noise.

The goal of statistical learning is to find the "sweet spot" minimizing both bias and variance–a model complex enough to represent structure, but simple enough to generalize.

The Role of the Test Set. The test set provides an unbiased estimate of the model's performance on unseen data. Formally, if $\mathbb{P}$ denotes the underlying data distribution and $\mathcal{L}$ the loss function, we can define:

$$R(\theta) = \mathbb{E}_{(x,y)\sim\mathbb{P}}[\mathcal{L}(f_\theta(x), y)].$$

Since $\mathbb{P}$ is unknown, we estimate this risk empirically on training or test samples:

$$\hat{R}_{\text{train}}(\theta) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \mathcal{L}(f_\theta(x_i), y_i), \qquad \hat{R}_{\text{test}}(\theta) = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \mathcal{L}(f_\theta(x_i), y_i).$$

A good model exhibits low $\hat{R}_{\text{test}}$ as well as low $\hat{R}_{\text{train}}$, demonstrating true predictive power.

3.3 Optimization by Gradient Descent

Once a loss function is defined, the next step is to minimize it. For most models, this is achieved by **gradient descent**, an iterative algorithm that updates the parameters in the opposite direction of the gradient of the loss.

Definition. Let $\theta \in \mathbb{R}^d$ be the vector of parameters and $\mathcal{L}(\theta)$ the differentiable loss. Starting from an initial guess $\theta^{(0)}$, the gradient descent algorithm performs:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \mathcal{L}(\theta^{(t)}),$$

where $\eta > 0$ is the **learning rate**. The process is repeated until convergence (i.e., until $\mathcal{L}$ stops decreasing significantly).

Geometric Intuition. Intuitively, gradient descent can be visualized as a small ball rolling down a landscape of loss values toward the lowest valley.

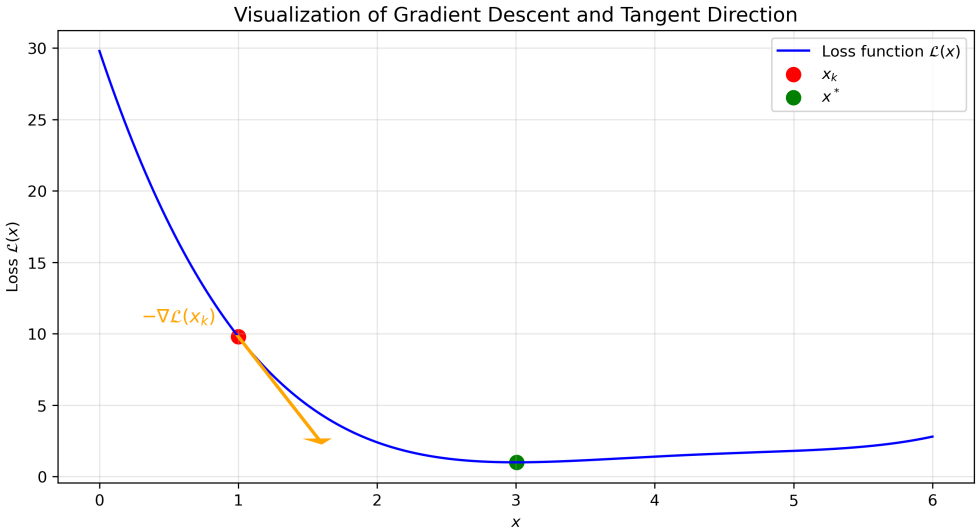


Figure 3.2: Gradient descent illustration: the gradient $-\nabla\ell(x_k)$ (orange arrow) points toward the minimum x^* (green) starting from x_k (red).

Each iteration moves the parameter θ in the direction of steepest descent, gradually approaching the minimum of $\mathcal{L}$. This method is fundamental because it scales naturally to high-dimensional parameter spaces and provides the foundation for training deep neural networks.

Why Gradient Descent Is So Useful. Gradient descent is one of the most fundamental and versatile optimization techniques in machine learning. Its power lies in its generality: it can be applied to highly complex and nonlinear loss functions, even when no analytical (closed-form) solution for the minimum exists. Rather than requiring the explicit computation of the minimum, gradient descent only depends on the *gradient* of the loss, which measures the local direction of steepest ascent or descent.

Another key advantage is scalability. In large datasets or high-dimensional models, the algorithm can be adapted through stochastic variants such as **Stochastic Gradient Descent (SGD)** (Bottou, 2012), **Adam** (Kingma and Ba, 2014), or **RMSProp** (Hinton, 2012), which approximate the gradient efficiently using small data batches.

Finally, gradient descent offers a powerful geometric and physical interpretation: learning can be seen as an *energy minimization process*, where the model iteratively reduces its "potential energy" (the loss) to reach a stable equilibrium corresponding to optimal parameters.

Connection to NLP. Every NLP model – from Word2Vec to BERT – relies on this principle. Training such models consists in defining a loss function (often derived from a softmax or energy-based formulation) and minimizing it by gradient descent. Understanding these basic ideas allows one to interpret the optimization of word embeddings and neural architectures in the chapters that follow.

3.4 The Maximum Likelihood Principle

3.4.1 Motivation and Intuition

One of the fundamental goals of statistical inference is to *estimate unknown parameters* from observed data. The idea behind the **Maximum Likelihood Estimation** (abbreviated **MLE**) is conceptually simple yet extremely powerful: we seek the parameter value that makes the observed data the most "likely" under the assumed model.

In other words, among all possible models, we select the one under which the actually observed data would have had the highest probability of occurring. This approach can be viewed as a form of *empirical coherence*: the chosen model should best explain what has been observed.

Example 3.4.1 (Intuitive Illustration)**.** *Suppose that a variable Y is assumed to follow a normal distribution $\mathcal{N}(\mu, \sigma^2)$, but the parameters μ and σ^2 are unknown. The likelihood principle seeks the values of μ and σ that make the observed sample $(y_1, \dots, y_n)$ most probable under this normality assumption. Hence, the maximum likelihood estimator selects the parameters $(\hat{\mu}, \hat{\sigma})$ that maximize the joint probability of the observed data.*

The principle of maximum likelihood relies on an "inverse" reasoning: we start from the data and infer the model that is most plausible to have generated them. This reasoning is universal: whether we deal with lifetimes, counts, survival probabilities, or risk scores, the same goal remains — to find the parameters that make our observations the most likely.

3.4.2 Mathematical Formalism

Consider an independent and identically distributed (i.i.d.) sample $Y_1, Y_2, \dots, Y_n \sim f(y; \theta)$, where $f(y; \theta)$ is a probability density (or mass) function parameterized by $\theta \in \Theta \subset \mathbb{R}^p$.

Definition 3.4.1 (Likelihood Function)**.** *The likelihood function is defined as*

$$L(\theta) = \prod_{i=1}^{n} f(y_i; \theta),$$

and represents the probability density of observing the sample $(y_1, \dots, y_n)$ given the parameter θ.

Definition 3.4.2 (Maximum Likelihood Estimator)**.** *The maximum likelihood estimator (MLE) is defined as*

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta \in \Theta} L(\theta) = \arg\max_{\theta \in \Theta} \log L(\theta).$$

Remark 1 (Log-Likelihood). In practice, we almost always work with the *log-likelihood*:

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^{n} \log f(y_i; \theta),$$

since the logarithm converts products into sums (which are easier to manipulate) and does not affect the maximizer.

Property 3.4.1 (Maximum Condition)**.** *Under regularity conditions, the MLE satisfies:*

$$\left.\frac{\partial \ell(\theta)}{\partial \theta}\right|_{\theta=\hat{\theta}} = 0, \quad \textit{and} \quad \left.\frac{\partial^2 \ell(\theta)}{\partial \theta^2}\right|_{\theta=\hat{\theta}} < 0,$$

meaning that the gradient of the log-likelihood vanishes at the maximum point, and that the Hessian matrix there is negative definite.

3.4.3 Properties and Statistical Interest

Definition 3.4.3 (Fisher Information Matrix)**.** *For a parametric model $f(y;\theta)$ satisfying regularity conditions, the **Fisher Information Matrix** is defined as*

$$\mathcal{I}(\theta) = \mathbb{E}\left[\left(\frac{\partial}{\partial \theta} \log f(Y;\theta)\right)\left(\frac{\partial}{\partial \theta} \log f(Y;\theta)\right)^{\top}\right] = -\,\mathbb{E}\left[\frac{\partial^2}{\partial \theta^2} \log f(Y;\theta)\right],$$

where the expectation is taken with respect to the true distribution of Y.

Remark 2 (Intuitive Interpretation). The Fisher Information quantifies how much "information" an observable random variable Y carries about an unknown parameter θ. If small changes in θ cause large variations in the likelihood function, then the model is said to contain a *lot* of information about θ. Conversely, if the likelihood changes very little with θ, the Fisher Information is small — meaning that the data are not very informative about that parameter.

Geometrically, the Fisher Information can be interpreted as a measure of **curvature** of the log-likelihood function around its maximum. A sharply peaked log-likelihood corresponds to high information (parameters are well determined), whereas a flat likelihood indicates uncertainty in estimation.

In this sense, $\mathcal{I}(\theta)$ acts like a notion of "variance" in the *parameter space*: it quantifies how sensitive the model is to perturbations in θ, and its inverse often approximates the covariance matrix of the estimator $\hat{\theta}_{\text{MLE}}$.

Property 3.4.2 (Asymptotic Properties)**.** *Under general conditions, the MLE possesses several remarkable asymptotic properties:*

- ***Consistency:*** $\hat{\theta}_{\text{MLE}} \to \theta^\star$ *as* $n \to \infty$*;*
- ***Asymptotic Normality:***

$$\sqrt{n}(\hat{\theta}_{\text{MLE}} - \theta^\star) \xrightarrow{\mathcal{L}} \mathcal{N}\big(0, \mathcal{I}(\theta^\star)^{-1}\big),$$

  where $\mathcal{I}(\theta)$ *denotes the Fisher Information Matrix from Equation (3.4.3);*

- ***Asymptotic Efficiency:*** *among all regular estimators, the MLE achieves the minimum possible variance, as characterized by the Cramér–Rao lower bound.*

Remark 3 (Geometric Interpretation). The maximum likelihood estimator can be interpreted geometrically as finding the point θ whose parametric distribution $f(y;\theta)$ best "fits" the empirical distribution of the observed data. In other words, we adjust the model $f(y;\theta)$ so that its shape aligns as closely as possible with the data distribution. This geometric viewpoint helps to understand why, as n increases, the MLE converges toward the true parameter $\theta^\star$.

Connection with NLP Models

The principle of maximum likelihood is at the very heart of modern NLP models. Language models, from the early n-gram models to recent Transformer-based architectures, all aim to maximize the probability of observed linguistic sequences under a parameterized model.

Formally, given a sentence composed of tokens $(w_1, w_2, \ldots, w_T)$, the model assigns a conditional probability

$$\mathbb{P}_\theta(w_t \mid w_{t-1}, \ldots, w_1),$$

and learning consists in finding the parameter vector θ that maximizes the overall likelihood

$$L(\theta) = \prod_{t=1}^{T} \mathbb{P}_\theta(w_t \mid w_{t-1}, \ldots, w_1).$$

Taking the logarithm gives the log-likelihood:

$$\ell(\theta) = \sum_{t=1}^{T} \log \mathbb{P}_\theta(w_t \mid w_{t-1}, \ldots, w_1).$$

Maximizing this quantity corresponds to minimizing the **cross-entropy loss**, which is the standard objective in neural language modeling.

Thus, models such as **Word2Vec**, **GloVe**, and **BERT** can all be interpreted as optimizing (or approximating) a maximum likelihood objective — where the softmax function plays the role of a normalized exponential ensuring that probabilities sum to one:

$$\mathbb{P}_\theta(w_O \mid w_I) = \frac{\exp(\langle v_{w_O}, v_{w_I} \rangle)}{\sum_{w \in \mathcal{V}} \exp(\langle v_w, v_{w_I} \rangle)}.$$

In this context, the Fisher Information acquires a practical interpretation: it measures the *sensitivity* of the model's output probabilities to small changes in the embedding or network parameters θ. A high Fisher Information indicates that the model parameters are strongly constrained by the data — for instance, when common words such as *"doctor"* and *"hospital"* consistently co-occur, the likelihood landscape becomes sharply curved around their optimal embeddings.

Consequently, understanding the geometry of the likelihood and its curvature (through the Fisher matrix) helps explain why neural NLP models can both generalize and overfit: a very sharp curvature implies strong specialization (low bias, high variance), while a flatter likelihood surface corresponds to more robust and generalizable embeddings.

3.5 Cross-Entropy Loss in Natural Language Processing

In Natural Language Processing, many models are designed to output a probability distribution over a discrete vocabulary. For example, a language model estimates

$$\mathbb{P}(w_t \mid w_1, \ldots, w_{t-1}),$$

the probability of the next word w_t given its context. To train such models, we need a loss function that quantifies how close the predicted distribution is to the true one (the observed word).

The **cross-entropy loss** fulfills this role. It measures the dissimilarity between two probability distributions: the model's predicted distribution q and the true (empirical) distribution p. Minimizing cross-entropy encourages the model to assign high probability to the correct word while penalizing wrong predictions proportionally to their confidence.

Definition 3.5.1. *Let $p = (p_1, p_2, \ldots, p_K)$ and $q = (q_1, q_2, \ldots, q_K)$ be two discrete probability distributions over a vocabulary of size K. The **cross-entropy** of p relative to q is defined as:*

$$H(p, q) = -\sum_{i=1}^{K} p_i \log q_i.$$

In NLP tasks, the true distribution p is usually a one-hot vector:

$$p_i = \begin{cases} 1 & \text{if } i = i^* \text{ (true target word)}, \\ 0 & \text{otherwise.} \end{cases}$$

Hence, the loss reduces to the negative log-probability of the correct word:

$$H(p, q) = -\log q_{i^*}.$$

In other words, minimizing cross-entropy is equivalent to maximizing the likelihood of the observed word under the model's predicted distribution.

3.5.1 Link to Maximum Likelihood Estimation

Cross-entropy loss naturally arises from the principle of **maximum likelihood estimation (MLE)**. Given a dataset of observed word–context pairs $\{(x_t, w_t)\}$, the model predicts probabilities $q_\theta(w_t \mid x_t)$. The MLE objective is to maximize:

$$\mathcal{L}(\theta) = \sum_t \log q_\theta(w_t \mid x_t),$$

which is equivalent to minimizing the expected cross-entropy between the empirical distribution and the model's distribution:

$$\min_\theta \ \mathbb{E}_{p_{\text{data}}}[-\log q_\theta(w_t \mid x_t)].$$

Thus, the cross-entropy loss is not an arbitrary choice—it is the canonical loss function for probabilistic models trained via MLE.

3.5.2 Why Cross-Entropy is Essential in NLP

Cross-entropy plays a central role in training virtually all modern NLP systems, from classical language models to deep neural architectures such as transformers. Its importance stems from several key properties:

- **Probabilistic consistency:** It directly measures the distance between true and predicted probability distributions.
- **Gradient interpretability:** The derivative of $-\log q_{i^*}$ increases sharply when q_{i^*} is small, encouraging the model to focus on underconfident predictions.

- **Compatibility with softmax:** Most NLP models end with a softmax layer:

$$q_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)},$$

producing a valid probability distribution over words. Cross-entropy and softmax form a natural mathematical pair for classification over large vocabularies.

- **Information-theoretic meaning:** Minimizing cross-entropy corresponds to minimizing the **Kullback–Leibler (KL) divergence** between p and q:

$$H(p, q) = H(p) + D_{\mathrm{KL}}(p\|q),$$

where $H(p)$ is constant with respect to model parameters. Hence, minimizing cross-entropy effectively minimizes $D_{\mathrm{KL}}(p\|q)$.

Example 3.5.1. *Predicting the Next Word Consider the sentence fragment:*

$$\textit{"The doctor works in the"} \quad \Rightarrow \quad ?$$

Suppose the model predicts:

$$q = [\textit{hospital} : 0.7, \textit{office} : 0.2, \textit{school} : 0.1],$$

and the true next word is hospital. Then the cross-entropy loss is:

$$H(p, q) = -\log(0.7) \approx 0.357.$$

If the model incorrectly predicted $q_{hospital} = 0.1$, the loss would rise to $-\log(0.1) = 2.302$, strongly penalizing the error.

Interpretation. Cross-entropy can be understood as a measure of "surprise": it quantifies how unexpected the true outcome is according to the model's predicted distribution. The lower the cross-entropy, the less surprised the model is — and the better it captures the underlying structure of the language.

In this sense, minimizing cross-entropy aligns the model's internal probabilistic geometry with the empirical distribution of natural language. It provides the mathematical foundation for learning meaningful embeddings, predictive text models, and generative systems in NLP.

Chapter 4

The Softmax Function and Energy-Based Models (EBM)

In this chapter, we bridge the mathematical concepts that underpin the training of modern NLP models. We first explore the **softmax function**, which converts raw numerical scores into probabilistic predictions, and then the broader framework of **Energy-Based Models (EBMs)**, which generalizes this principle. Finally, we show how these concepts appear everywhere in Natural Language Processing, from word embeddings to large language models.

4.1 The Softmax Function: Definition and Intuition

Mathematical Definition. Given a vector of real-valued scores (often called *logits*) $z = (z_1, z_2, \ldots, z_K)$, the softmax function transforms these scores into a probability distribution over K categories:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}.$$

This simple normalization has several desirable properties:

- $p_i > 0$ for all i,
- $\sum_i p_i = 1$ (valid probability distribution),
- monotonicity: if $z_i > z_j$, then $p_i > p_j$,
- differentiability, allowing the use of gradient-based optimization.

The softmax acts as a smooth approximation to the *argmax* operator: the larger z_i, the more probable category i becomes, but without abrupt transitions.

Probabilistic Interpretation In probabilistic modeling, the softmax corresponds to the normalized exponential of unnormalized *log-probabilities*:

$$\mathbb{P}(y = i \mid x) = \frac{\exp(f_\theta(x)_i)}{\sum_j \exp(f_\theta(x)_j)}.$$

Here, $f_\theta(x)_i$ denotes the score (logit) assigned by the model to class i for input x. Thus, softmax serves as a natural bridge between **raw model outputs** and **probability theory**.

Statistical Interpretation: Logistic Regression Generalized Softmax generalizes the logistic regression model to multi-class problems. In binary classification, we have:

$$\mathbb{P}(y = 1 \mid x) = \frac{e^{w_1^\top x}}{e^{w_1^\top x} + e^{w_0^\top x}},$$

while the multi-class case extends this to:

$$\mathbb{P}(y = k \mid x) = \frac{e^{w_k^\top x}}{\sum_{j=1}^{K} e^{w_j^\top x}}.$$

From a statistical viewpoint, this form is the **maximum-entropy** distribution consistent with the given expected features. It is the most "uninformative" distribution satisfying linear constraints – hence its theoretical elegance and universality.

Physical Interpretation: The Boltzmann Distribution In statistical physics, the softmax corresponds to the Boltzmann or Gibbs distribution:

$$\mathbb{P}(i) = \frac{e^{-\beta E_i}}{\sum_j e^{-\beta E_j}},$$

where E_i represents the **energy** of state i and β is the inverse temperature. High-probability states correspond to low-energy configurations. The temperature parameter $\tau = 1/\beta$ controls the concentration of probabilities:

$$\text{softmax}_\tau(z_i) = \frac{e^{z_i/\tau}}{\sum_j e^{z_j/\tau}}.$$

As $\tau \to 0$, the distribution becomes a hard maximum; as $\tau \to \infty$, it approaches uniformity.

Geometric Interpretation: Mapping to the Probability Simplex. The softmax function can be viewed geometrically as a smooth projection from $\mathbb{R}^K$ to the $(K-1)$-dimensional probability simplex:

$$\Delta^{K-1} = \{p \in \mathbb{R}^K : p_i \geq 0,\ \sum_i p_i = 1\}.$$

This mapping preserves the ordering of components while compressing differences. It is also translation-invariant:

$$\text{softmax}(z + c\mathbf{1}) = \text{softmax}(z),$$

meaning that adding a constant to all scores does not affect the resulting probabilities. This makes the softmax ideal for neural outputs that depend only on relative differences.

Computational Interpretation. Softmax is differentiable and stable to compute, especially with the *log-sum-exp trick*:

$$\log \sum_i e^{z_i} = z_{\max} + \log \sum_i e^{z_i - z_{\max}}.$$

Its derivative has a simple closed form:

$$\frac{\partial p_i}{\partial z_j} = p_i(\delta_{ij} - p_j),$$

which leads directly to elegant backpropagation rules. This property makes the softmax essential in deep learning pipelines, particularly when combined with the cross-entropy loss:

$$\mathcal{L} = -\sum_i y_i \log p_i.$$

Summary. The softmax simultaneously belongs to multiple disciplines:

- **Probability:** normalizes unscaled scores into valid probabilities.
- **Statistics:** yields maximum-entropy distributions.
- **Physics:** represents Gibbs distributions over energy levels.
- **Geometry:** maps real-valued vectors to the probability simplex.
- **Computation:** provides a stable, differentiable normalization.

Its universality explains why it has become the standard normalization in classification, language modeling, and attention mechanisms across NLP.

4.2 Energy-Based Models (EBM)

4.2.1 Definition

An **Energy-Based Model** (Mnih and Hinton, 2008; LeCun et al., 2006) associates a scalar energy $E_\theta(x, y)$ to every possible pair of input x and output y. The goal is to assign low energy to compatible pairs and high energy to incompatible ones. Formally, the model defines a probability distribution through the Boltzmann normalization:

$$\mathbb{P}_\theta(y \mid x) = \frac{e^{-E_\theta(x,y)}}{Z_\theta(x)}, \quad Z_\theta(x) = \sum_{y'} e^{-E_\theta(x,y')}.$$

Thus, the *softmax* over negative energies converts an arbitrary energy landscape into a valid probability distribution.

Intuition. The energy function defines a landscape where valleys represent plausible configurations. Learning corresponds to **sculpting the energy surface** such that observed data (x, y) fall into deep valleys (low energy), while unobserved or incorrect pairs lie on high plateaus (high energy).

The model can then predict by choosing the configuration of minimal energy:

$$\hat{y} = \arg\min_y E_\theta(x, y).$$

This view unifies many learning paradigms:

- Regression: energy = squared residual $(y - f_\theta(x))^2$.
- Classification: energy = $-f_\theta(x)_y$ (negative score).
- Probabilistic models: energy proportional to $-\log \mathbb{P}_\theta(y \mid x)$.

4.2.2 The Softmax as a Normalized Energy Model

The connection between softmax and EBMs is straightforward. Let $E_\theta(x, y) = -f_\theta(x, y)$ be the negative of the model's score. Then:

$$\mathbb{P}_\theta(y \mid x) = \frac{\exp(f_\theta(x, y))}{\sum_{y'} \exp(f_\theta(x, y'))} = \text{softmax}(f_\theta(x, y)).$$

Thus, every neural classifier using a softmax output layer can be seen as a **normalized energy-based model**. The function $f_\theta(x, y)$ defines the compatibility score between x and y; the softmax transforms it into probabilities.

4.2.3 Unnormalized Energy Models and Contrastive Learning

In some contexts, computing the partition function $Z_\theta(x)$ is infeasible (e.g., when y spans a large vocabulary). In that case, the model may operate in an **unnormalized** form, trained with contrastive objectives such as:

- **Noise Contrastive Estimation (NCE)**,
- **Contrastive Divergence** (used in Restricted Boltzmann Machines),
- **Energy-based Transformers and Contrastive Language Models**.

These approaches approximate the normalization implicitly by learning to assign higher scores to correct (x, y) pairs than to corrupted ones.

Historical Note. The energy-based perspective dates back to the statistical mechanics interpretation of neural networks, particularly the Boltzmann machine. In NLP, the connection between energy and probability became explicit with hierarchical softmax and distributed language models such as *A Scalable Hierarchical Distributed Language Model* by Mnih and Hinton (2008), which introduced efficient approximations for large vocabulary normalization.

4.3 Softmax, Energy Models, and Their Role in NLP

Language Modeling as Energy Minimization. In NLP, predicting the next word w_t given its context c_t is equivalent to assigning an energy to every possible candidate word:

$$E(c_t, w_t) = -f_\theta(c_t, w_t),$$

where f_θ measures compatibility between the context and the word. The model defines:

$$\mathbb{P}(w_t \mid c_t) = \frac{\exp(f_\theta(c_t, w_t))}{\sum_{w'} \exp(f_\theta(c_t, w'))}.$$

The training objective (cross-entropy or negative log-likelihood) encourages high probability for observed words, i.e. low energy for correct pairs.

Word2Vec as an Energy-Based Model The Skip-Gram version of Word2Vec learns embeddings v_w and v_c such that:

$$E(w, c) = -v_w^\top v_c.$$

The training objective minimizes the energy of observed (word, context) pairs while increasing that of random pairs (negative sampling). This process is a form of **contrastive energy learning**, central to representation learning in NLP.

Transformers and Attention as Energy Distributions Even attention mechanisms can be seen as normalized energy models. Given query q_i and keys $\{k_j\}$, the attention weights are computed as:

$$\alpha_{ij} = \frac{\exp(q_i^\top k_j / \sqrt{d})}{\sum_l \exp(q_i^\top k_l / \sqrt{d})}.$$

Here, the compatibility score $q_i^\top k_j$ plays the role of $-E_{ij}$, and the softmax transforms these energies into a probability distribution over possible attention targets. Thus, the same principle – *low energy for high compatibility* – drives attention and contextualization in NLP.

Conceptual Summary. Softmax and Energy-Based Models provide a unified lens for understanding modern NLP:

- The **softmax** defines a probabilistic normalization of scores.
- The **energy function** defines compatibility between inputs and outputs.
- **Training** consists of lowering the energy of observed examples.
- **Inference** corresponds to finding the lowest-energy configuration.

From early distributed models like Word2Vec to large-scale transformers, the logic is the same: *Language understanding emerges from learning an energy landscape where coherent text has low energy, and incoherent text high energy.*

Conclusion. Softmax and EBMs are not isolated mathematical tools; they are the conceptual backbone of all modern neural NLP systems. They connect probability, optimization, and representation learning into a single coherent theory. Understanding them provides the foundation for interpreting how models such as BERT, GPT, or other large language models learn and reason about language.

Chapter 5

Neural Networks: Foundations, Architecture, and Backpropagation

After understanding the principles of model learning and optimization, we now explore the core mathematical structure behind modern NLP systems: the **artificial neural network**. Neural networks are flexible, differentiable architectures capable of approximating highly nonlinear functions, making them the cornerstone of deep learning.

5.1 Statistical Learning Tools for Neural Networks and NLP

Before studying neural networks and gradient-based optimization, it is essential to recall the core differential tools from multivariable calculus. These form the mathematical foundation of backpropagation and optimization algorithms used in modern NLP.

Taylor–Young Expansion. The Taylor–Young theorem generalizes the concept of a local linear (or quadratic) approximation of a smooth function. It allows us to describe how a function varies near a point using its derivatives — a key idea for optimization algorithms.

Theorem 5.1.1 (Taylor–Young Expansion in One Variable)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ be of class $\mathcal{C}^n$ in a neighborhood of $a \in \mathbb{R}$. Then, as $x \to a$,*

$$f(x) = f(a) + \sum_{k=1}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k + o\big((x-a)^n\big).$$

Theorem 5.1.2 (Second-Order Taylor–Young Expansion in Several Variables)**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be of class $\mathcal{C}^2$ in a neighborhood of $a \in \mathbb{R}^d$. For x close to a, set $h = x - a$. Then*

$$f(a+h) = f(a) + \nabla f(a) \cdot h + \tfrac{1}{2} h^\top H_f(a) h + o(\|h\|^2), \quad (h \to 0).$$

Remark 4. Two points here:

- The linear term $\nabla f(a) \cdot h$ corresponds to the tangent hyperplane of f at a.
- The quadratic term $\frac{1}{2} h^\top H_f(a) h$ captures the local curvature near a.

Here, $\nabla f(a)$ is the gradient vector, and $H_f(a)$ is the Hessian matrix:

$$\nabla f(a) = \begin{pmatrix} f_{x_1}(a) \\ \vdots \\ f_{x_d}(a) \end{pmatrix}, \qquad H_f(a) = \left(\frac{\partial^2 f}{\partial x_i \partial x_j}(a) \right)_{1 \le i,j \le d}.$$

Gradient and Level Curves. In two dimensions, the gradient not only measures the direction of fastest change but also reveals geometric properties of the function's level sets.

Proposition 5.1.1 (Orthogonality of the Gradient to Level Sets)**.** *Let $f : \mathbb{R}^2 \to \mathbb{R}$ be of class $\mathcal{C}^1$. For any $k \in \mathbb{R}$, define the level set*

$$L_k = \{(x, y) \in \mathbb{R}^2 : f(x, y) = k\}.$$

If $p \in L_k$ and $\nabla f(p) \neq 0$, then $\nabla f(p)$ is orthogonal to the tangent of L_k at p.

Idea of the proof. Let $\gamma : I \to \mathbb{R}^2$ be a smooth parametrization of L_k with $\gamma(t_0) = p$. Since $f(\gamma(t)) \equiv k$, the derivative is null and:

$$0 = \frac{\mathrm{d}}{\mathrm{d}t}(f \circ \gamma)(t_0) = \nabla f(\gamma(t_0)) \cdot \gamma'(t_0) = \nabla f(p) \cdot \gamma'(t_0).$$

Thus, $\nabla f(p)$ is orthogonal to all tangent vectors $\gamma'(t_0)$ of L_k at p. □

Gradient and Direction of Maximal Growth. Beyond its geometric meaning, the gradient also gives the direction in which a function increases (or decreases) most rapidly — a central idea for optimization.

Proposition 5.1.2 (Gradient as Direction of Steepest Ascent)**.** *For $f : \mathbb{R}^2 \to \mathbb{R}$ of class $\mathcal{C}^1$, and any unit vector $u \in \mathbb{R}^2$, the directional derivative of f at p in the direction u is*

$$D_u f(p) = \nabla f(p) \cdot u.$$

By the Cauchy–Schwarz inequality,

$$|D_u f(p)| \leq \|\nabla f(p)\|.$$

The maximum value $\|\nabla f(p)\|$ is achieved when u is aligned with the gradient:

$$u = \frac{\nabla f(p)}{\|\nabla f(p)\|}.$$

Hence, $\nabla f(p)$ points in the direction of maximal increase of f, while $-\nabla f(p)$ points in the direction of steepest decrease.

Remark 5. The first-order Taylor approximation

$$f(p + tu) = f(p) + t\, D_u f(p) + o(t)$$

shows that for small $t > 0$, $f(p + tu) > f(p)$ if u points in the direction of $\nabla f(p)$.

Chain Rule in Several Variables. The chain rule connects the derivatives of composed functions, allowing gradients to be propagated through layers of transformations — the very principle behind the backpropagation algorithm.

Proposition 5.1.3 (Chain Rule: From 1D to 2D)**.** *Let $f : \mathbb{R}^2 \to \mathbb{R}$ be of class $\mathcal{C}^1$ and $u, v : \mathbb{R} \to \mathbb{R}$ differentiable. Define $g(t) = f(u(t), v(t))$. Then:*

$$\frac{\mathrm{d}g}{\mathrm{d}t}(t) = \frac{\partial f}{\partial x}(u(t), v(t))u'(t) + \frac{\partial f}{\partial y}(u(t), v(t))v'(t).$$

In vector form, if $w(t) = (u(t), v(t))$, then:

$$g'(t) = w'(t) \cdot \nabla f(w(t)).$$

Proposition 5.1.4 (Chain Rule: From 2D to 2D (Jacobian Form))**.** *Let $f : \mathbb{R}^2 \to \mathbb{R}$ and $u, v : \mathbb{R}^2 \to \mathbb{R}$ be of class $\mathcal{C}^1$. Define $h(x, y) = f(u(x, y), v(x, y))$. Then:*

$$\frac{\partial h}{\partial x} = \frac{\partial f}{\partial x}(u, v)u_x + \frac{\partial f}{\partial y}(u, v)v_x,$$
$$\frac{\partial h}{\partial y} = \frac{\partial f}{\partial x}(u, v)u_y + \frac{\partial f}{\partial y}(u, v)v_y.$$

In matrix form, setting

$$W(x, y) = \begin{pmatrix} u(x, y) \\ v(x, y) \end{pmatrix}, \quad J_W(x, y) = \begin{pmatrix} u_x & u_y \\ v_x & v_y \end{pmatrix},$$

the chain rule becomes:

$$\nabla h(x, y) = J_W(x, y)^\top \nabla f\big(W(x, y)\big).$$

Theorem 5.1.3 (General Chain Rule)**.** *Let $f : \mathbb{R}^m \to \mathbb{R}^p$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ be differentiable. Then their composition $h = f \circ g : \mathbb{R}^n \to \mathbb{R}^p$ is differentiable, and its Jacobian satisfies:*

$$J_h(x) = J_g(x)\, J_f\big(g(x)\big),$$

or equivalently,

$$J_h(x) = J_f\big(g(x)\big)\, J_g(x),$$

depending on whether Jacobians are viewed as row or column operators.

Component form: *For each $1 \le i \le p$ and $1 \le j \le n$,*

$$\frac{\partial h_i}{\partial x_j}(x) = \sum_{k=1}^{m} \frac{\partial f_i}{\partial y_k}\big(g(x)\big) \frac{\partial g_k}{\partial x_j}(x).$$

Remark 6. This general chain rule is the mathematical foundation of **backpropagation** in neural networks: it expresses the gradient of a composite function as the product of local derivatives (Jacobians) of each layer.

5.2 Motivation: Why Neural Networks?

5.2.1 From Linear Models to Nonlinear Representations

Classical linear models such as linear regression or logistic regression express predictions as:

$$\hat{y} = w^\top x + b,$$

where $x \in \mathbb{R}^d$ is an input vector and (w, b) are parameters. These models are limited because they can only capture **linear dependencies** between input and output.

However, most real-world relationships – especially in language – are inherently nonlinear: words interact in complex, context-dependent ways. For instance, the meaning of "bank" changes entirely between "river bank" and "bank account".

Neural networks overcome this limitation by introducing intermediate transformations called **hidden layers**, which allow the model to learn hierarchical and nonlinear representations.

Biological Analogy. The concept of neural networks is inspired by the human brain. Each neuron receives input signals, applies a transformation, and transmits an output signal to other neurons. In artificial networks:

- Inputs are numeric features,

- Each neuron applies a weighted sum followed by a nonlinear activation,
- The network as a whole learns a mapping from input to output through composition of layers.

The strength of this analogy lies not in biological accuracy but in its functional abstraction: simple computational units, when combined in large numbers, can approximate arbitrarily complex functions.

5.2.2 Universal Approximation Theorem

A fundamental theoretical result, known as the **Universal Approximation Theorem** (Cybenko, 1989; Hornik, 1991), states that:

> A feed-forward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on compact subsets of $\mathbb{R}^n$, given suitable activation functions.

This theorem formalizes the expressive power of neural networks and justifies their use as general-purpose function approximators in all domains, including natural language.

5.3 Architectural Principles: Building Networks Layer by Layer

5.3.1 The Neuron: Basic Computational Unit

A neuron takes a vector of inputs $x = (x_1, \ldots, x_d)$ and produces an output

$$h = \sigma(w^\top x + b),$$

where:

- w is a weight vector,
- b is a bias term,
- σ is a nonlinear activation function.

The activation function introduces nonlinearity into the model. Common choices include:

$$\sigma(x) = \tanh(x), \quad \sigma(x) = \frac{1}{1+e^{-x}} \text{ (sigmoid)}, \quad \sigma(x) = \max(0, x) \text{ (ReLU)}.$$

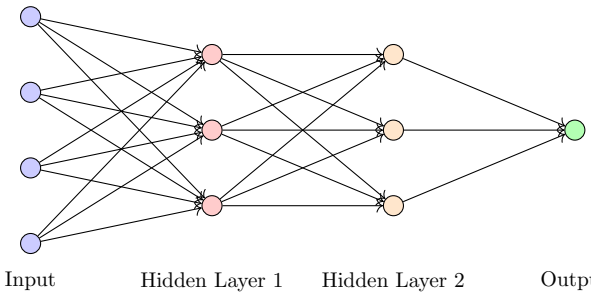
Without σ, the composition of multiple layers would still be linear and offer no advantage over standard regression.

5.3.2 Feedforward Architecture

A **feedforward neural network** (or multilayer perceptron) is constructed by stacking layers:

$$\begin{aligned} h^{(1)} &= \sigma(W^{(1)}x + b^{(1)}), \\ h^{(2)} &= \sigma(W^{(2)}h^{(1)} + b^{(2)}), \\ &\vdots \\ \hat{y} &= W^{(L)}h^{(L-1)} + b^{(L)}. \end{aligned}$$

Each layer performs an affine transformation followed by a nonlinear activation. By composing several layers, the network learns **hierarchical features**: the first layers capture simple patterns, and deeper layers capture more abstract and semantic structures.



This layered structure is the foundation for more advanced architectures: convolutional networks (CNNs), recurrent networks (RNNs), and transformers all extend this core idea.

5.3.3 Nonlinearity and Representation Power

Stacking multiple nonlinear transformations allows the model to represent compositions of features, a key property for NLP:

- Lower layers can encode lexical or syntactic features.
- Higher layers capture semantics, context, and meaning.

This hierarchical abstraction mirrors how humans process language – from sounds and words to sentences and discourse.

5.4 Backpropagation: Learning Through the Chain Rule

5.4.1 Objective: Minimizing a Loss Function

As with all learning systems, a neural network is trained by minimizing a loss function $\mathcal{L}(\theta)$, such as mean squared error or cross-entropy:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i).$$

The parameters $\theta = \{W^{(l)}, b^{(l)}\}_{l=1}^{L}$ are updated iteratively via gradient descent:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}.$$

The Need for Efficient Gradient Computation. For deep networks, $\mathcal{L}$ depends on parameters through multiple nested functions:

$$\mathcal{L} = \ell(W^{(L)} h^{(L-1)} + b^{(L)}, y), \quad \text{where } h^{(L-1)} = \sigma(W^{(L-1)} h^{(L-2)} + b^{(L-1)}), \text{ etc.}$$

Manually computing partial derivatives layer by layer is impractical. The **backpropagation algorithm** solves this by systematically applying the chain rule of calculus in reverse.

5.4.2 Chain Rule Recap

If $y = f(u)$ and $u = g(x)$, then by the chain rule:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}y}{\mathrm{d}u} \cdot \frac{\mathrm{d}u}{\mathrm{d}x}.$$

In a neural network, this principle extends to multivariate compositions:

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \frac{\partial \mathcal{L}}{\partial h^{(l)}} \cdot \frac{\partial h^{(l)}}{\partial W^{(l)}}.$$

5.4.3 Forward and Backward Pass

Training proceeds in two stages:

1. Forward Pass. Compute outputs layer by layer:

$$h^{(l)} = \sigma(W^{(l)} h^{(l-1)} + b^{(l)}).$$

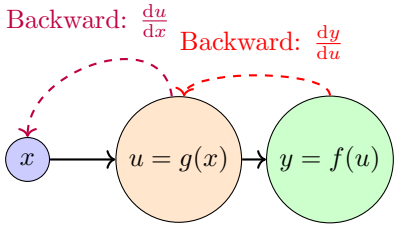
At the end, compute the loss $\mathcal{L}$ from predictions and true labels.

2. Backward Pass. Propagate the error backward using the chain rule:

$$\delta^{(l)} = (\nabla_{h^{(l)}} \mathcal{L}) = ((W^{(l+1)})^\top \delta^{(l+1)}) \odot \sigma'(a^{(l)}),$$

where $a^{(l)} = W^{(l)} h^{(l-1)} + b^{(l)}$ and $\odot$ denotes elementwise multiplication. Gradients with respect to weights are:

$$\nabla_{W^{(l)}} \mathcal{L} = \delta^{(l)} (h^{(l-1)})^\top.$$



Forward and backward computation through chain rule

This mechanism allows efficient computation of all gradients with only about twice the computational cost of a forward pass, regardless of network depth.

Why Backpropagation Is Fundamental.

- It enables **end-to-end learning**: every parameter in the network receives gradient information.
- It scales to millions of parameters through automatic differentiation.
- It unifies all models – convolutional, recurrent, or transformer-based – under a single optimization framework.

Connection to NLP. In NLP, backpropagation allows neural architectures to learn distributed representations of words, sentences, and documents by minimizing a loss (e.g., cross-entropy) over large corpora. Whether training embeddings (Word2Vec), sequence encoders (LSTMs), or transformers (BERT), the process remains the same:

$$\text{Forward pass (prediction)} + \text{Backward pass (gradient update)} \Rightarrow \text{Representation learning.}$$

Summary. Neural networks are powerful because they combine:

- layered nonlinear transformations,
- a unified differentiable structure,
- an efficient optimization algorithm (gradient descent via backpropagation).

This combination makes them the natural tool for modeling language, where relationships between words and meanings are hierarchical, nonlinear, and context-dependent. Understanding their architecture and learning dynamics provides the foundation for studying embeddings, transformers, and all modern NLP systems.

Remark 7 (Differential Form of the Chain Rule in Neural Networks). Let a neural network be a composition of layers

$$f = \phi^{(L)} \circ \phi^{(L-1)} \circ \cdots \circ \phi^{(1)},$$

where each layer is defined by

$$\phi^{(l)}(x^{(l)}) = \sigma^{(l)}\big(W^{(l)}x^{(l)} + b^{(l)}\big),$$

with learnable parameters $(W^{(l)}, b^{(l)})$ and activation function $\sigma^{(l)}$.

Then, the differential of the whole network with respect to its input satisfies

$$Df(x^{(1)}) = D\phi^{(L)}_{x^{(L)}} \circ D\phi^{(L-1)}_{x^{(L-1)}} \circ \cdots \circ D\phi^{(1)}_{x^{(1)}},$$

where $x^{(l+1)} = \phi^{(l)}(x^{(l)})$.

When differentiating with respect to parameters,

$$Df_{W^{(l)}} = D\phi^{(L)}_{x^{(L)}} \circ \cdots \circ D\phi^{(l+1)}_{x^{(l+1)}} \circ D_{W^{(l)}}\phi^{(l)}(x^{(l)}),$$

and similarly for $b^{(l)}$.

This highlights the essence of **backpropagation**: the global gradient is obtained by successive compositions of local differentials — each layer passes its gradient to the previous one, weighted by the Jacobian of its local transformation.

Chapter 6

The Importance of Representing Words in a Vector Space

The ability to represent linguistic units numerically lies at the heart of modern Natural Language Processing. Before constructing semantic embeddings or geometric structures, one must define what the model manipulates: **tokens**. This chapter begins by introducing the concept of tokens and vocabulary, and shows how words are mapped to numerical representations suitable for learning.

6.1 From Tokens to Vocabulary and Embedding Matrices

6.1.1 Definition of a Token

In any NLP pipeline, the first step is to decompose text into basic units called **tokens**. A token is an atomic element of the input sequence. Depending on the task, tokens can represent:

- individual words (e.g., "hospital", "patient", "diagnosis");
- subwords or morphemes (e.g., "play" and "-ing" in "playing");
- or even individual characters or punctuation marks.

Tokenization converts raw text into a discrete sequence:

$$\text{"The patient recovered."} \longrightarrow [\text{"The"}, \text{"patient"}, \text{"recovered"}, \text{"."}].$$

These tokens become the fundamental input units for all subsequent computations.

The Vocabulary Set. Once tokenized, all distinct tokens in a corpus form the **vocabulary**:

$$\mathcal{V} = \{v_1, v_2, \ldots, v_N\}.$$

Each element v_i corresponds to a unique word (or token type). The vocabulary size $N = |\mathcal{V}|$ can range from a few hundred words in a small dataset to several hundred thousand in large corpora.

In practice, models often include two special tokens:

- `[UNK]` – representing unknown or rare words not seen during training;
- `[PAD]` – used for padding sequences to equal length.

This finite vocabulary acts as a discrete lookup table from which all word representations are derived.

From Words to Vectors: The Embedding Matrix. To process text numerically, each word in the vocabulary is associated with a dense vector of fixed dimension d. Formally, we define the **embedding matrix**:

$$W = [\, w_1 \; w_2 \; \ldots \; w_N \,]^\top \in \mathbb{R}^{N \times d},$$

where each row $w_i \in \mathbb{R}^d$ represents the embedding of the corresponding token $v_i \in \mathcal{V}$.

Initially, these embeddings are typically initialized randomly:

$$w_i^{(0)} \sim \mathcal{U}(-\epsilon, \epsilon),$$

where ϵ controls the initialization range. During training, they are refined through optimization so as to capture the contextual meaning of each word.

6.1.2 Learning the Embedding Matrix via the Softmax Objective

In neural language models (see previous chapters on softmax and energy-based models), each word embedding participates in the computation of a conditional probability distribution:

$$\mathbb{P}(w_t \mid \text{context}) = \frac{\exp({v_{w_t}}^\top h_t)}{\sum_{w \in \mathcal{V}} \exp({v_w}^\top h_t)},$$

where:

- h_t is the hidden representation of the context (output of the neural network),
- v_w is the output embedding corresponding to word w,
- W contains all embeddings to be learned.

The network parameters – including the embeddings – are optimized by minimizing a loss such as the negative log-likelihood:

$$\mathcal{L} = -\sum_t \log \mathbb{P}(w_t \mid \text{context}).$$

Through this process, words that occur in similar contexts receive similar embeddings. This transformation converts symbolic co-occurrence relationships into continuous geometric proximity.

Why an Embedding Matrix? Using an embedding matrix offers multiple advantages:

- It enables efficient lookup: given a token index i, we directly access its vector $w_i = W[i]$.
- It reduces sparsity: instead of one-hot encoding of size N, we work in a dense space of dimension $d \ll N$.
- It provides differentiable representations that can be updated by gradient descent.

Hence, W serves as a learnable dictionary that bridges discrete symbolic input and continuous mathematical computation.

Training Dynamics and Interpretation. At initialization, all embeddings are arbitrary points in $\mathbb{R}^d$. As training progresses, the gradient of the loss function propagates back through the softmax layer and updates W:

$$W \leftarrow W - \eta \, \nabla_W \mathcal{L},$$

where η is the learning rate. This process gradually organizes the embeddings so that semantically similar words cluster together in space.

After convergence, W contains the learned word representations that encode syntax, semantics, and even higher-order linguistic patterns. These embeddings can then be used as inputs to other models – such as classifiers, sequence encoders, or transformers – forming the foundation of modern NLP architectures.

Summary. To summarize:

- Text is first decomposed into tokens forming a finite vocabulary $\mathcal{V}$.
- Each token $v \in \mathcal{V}$ is associated with a vector $w_v \in \mathbb{R}^d$.
- All embeddings form a learnable matrix $W \in \mathbb{R}^{N \times d}$.
- The embeddings are optimized through backpropagation and softmax-based loss functions.

Once learned, these vectors transform language into geometry – allowing algebraic, probabilistic, and semantic reasoning in continuous space. The following sections explore how this vectorization of language encodes meaning, enables analogy, and defines quantitative measures of semantic similarity.

6.2 From Discrete Symbols to Continuous Geometry

Language is fundamentally discrete: words are symbolic entities, each with its own identity but no inherent numerical structure. In their raw form, words cannot be directly manipulated by mathematical models – there is no notion of "distance" or "similarity" between symbols such as *cat*, *dog*, or *car*.

To overcome this limitation, modern NLP represents each word w as a vector $v_w \in \mathbb{R}^d$, embedding it into a continuous vector space known as the **embedding space**. This mapping enables geometric reasoning about meaning: words that appear in similar contexts (and thus have related meanings) are positioned close to one another in this space.

The resulting geometry allows us to perform algebraic operations on words and to use powerful mathematical tools – distances, angles, and linear transformations – to reason about language quantitatively.

6.3 Semantic Structure and Linear Regularities

A remarkable property of embedding spaces is that semantic and syntactic relationships often correspond to simple linear transformations. For instance:

$$v_{\text{king}} - v_{\text{man}} + v_{\text{woman}} \approx v_{\text{queen}}.$$

This relationship suggests that embeddings do not merely capture co-occurrence statistics; they organize words along meaningful geometric dimensions.

If the embedding model captures semantics effectively, then:

- The difference $v_{\text{king}} - v_{\text{man}}$ points to a "royalty" direction in the space.
- The difference $v_{\text{woman}} - v_{\text{man}}$ encodes a "gender" direction.
- Analogous relationships (e.g., Paris − France + Italy ≈ Rome) reveal country–capital relations.

In this sense, the vector space decomposes into **subspaces** that reflect conceptual axes – gender, number, tense, profession, royalty, and others. The structure of the space therefore mirrors the latent structure of human semantics.

6.4 Geometric Intuition: Embeddings as Coordinates of Meaning

Each embedding dimension can be interpreted as a latent feature that contributes to meaning. Although individual coordinates are not directly interpretable, their collective configuration encodes similarities, analogies, and functional relationships.

Visually, one can imagine embeddings forming clusters in high-dimensional space:

- Words referring to animals (*dog*, *cat*, *lion*) occupy one region.
- Occupations (*doctor*, *nurse*, *teacher*) cluster elsewhere.

- Abstract concepts (*love*, *hate*, *joy*) lie on different semantic manifolds.

The topology of this space – the distances and angles between vectors – reveals how meanings relate or diverge.

6.5 Measuring Similarity Between Word Embeddings

Once words are represented as vectors, their similarity can be quantified geometrically. Two of the most common measures are the **Euclidean distance** and the **cosine similarity**.

Euclidean Distance. The Euclidean distance between two embeddings $v_i, v_j \in \mathbb{R}^d$ is defined as:

$$d_{\text{Euc}}(v_i, v_j) = \|v_i - v_j\|_2 = \sqrt{\sum_{k=1}^{d} (v_{i,k} - v_{j,k})^2}.$$

This metric measures absolute spatial proximity. However, it can be sensitive to vector norms – longer vectors dominate even if their directions are similar.

Cosine Similarity. A more robust and widely used measure in NLP is the **cosine similarity**, derived from the scalar product:

$$\text{cosine\_sim}(v_i, v_j) = \frac{v_i \cdot v_j}{\|v_i\| \, \|v_j\|} = \cos(\theta_{ij}),$$

where θ_{ij} is the angle between the two vectors.

The cosine similarity focuses on orientation rather than magnitude. Two words are considered semantically similar if their embeddings point in the same direction, regardless of vector length.

- If $\cos(\theta_{ij}) = 1$, the words are perfectly aligned (identical in meaning).
- If $\cos(\theta_{ij}) = 0$, they are orthogonal (unrelated).
- If $\cos(\theta_{ij}) = -1$, they are opposites (antonymous or conceptually distant).

Why Cosine Similarity is Preferred in NLP. In high-dimensional spaces, the absolute position (norm) of vectors can vary due to training dynamics or normalization choices. Cosine similarity neutralizes this variation and focuses purely on **semantic direction**. This makes it a natural metric for comparing embeddings learned through softmax-based or energy-based models.

Furthermore, since embeddings are often optimized by maximizing inner products (e.g., $v_w^\top v_c$ in Word2Vec), cosine similarity directly reflects the optimization objective used during training.

Semantic Subspaces and Linear Analogies. The existence of linear analogies in embedding space suggests that certain directions correspond to semantic properties shared across words. Formally, let U_{royalty} denote the subspace associated with royal titles semantic. Then:

$$v_{\text{king}} - v_{\text{man}} \in U_{\text{royalty}}, \qquad v_{\text{queen}} - v_{\text{woman}} \in U_{\text{royalty}},$$

and thus these differences are approximately parallel. This geometric regularity implies that semantic relations can be treated as linear transformations – allowing the model to generalize across linguistic phenomena.

Conceptual Summary. Representing words in a vector space provides a powerful bridge between linguistics and geometry:

- Words become mathematical objects subject to linear algebraic operations.
- Semantic similarity is encoded as geometric proximity.

- Conceptual relations correspond to vector directions and subspaces.

These properties explain why embedding-based models revolutionized NLP. They translate discrete symbolic systems into continuous geometric structures where meaning can be learned, measured, and manipulated quantitatively. This embedding perspective remains central to modern representation learning, from Word2Vec to BERT and beyond.

Chapter 7

Learning Word Embeddings and Key NLP Models

The success of modern Natural Language Processing rests upon the ability to represent linguistic entities – words, phrases, or sentences – as numerical vectors that capture their meaning and contextual usage. This chapter presents the major milestones in the development of word and sentence embedding models, from shallow architectures such as *Word2Vec* to deep contextual models such as *BERT*. Each section focuses on the core principle, mathematical formulation, and motivation of the model.

7.1 Word2Vec (Mikolov et al., 2013)

Introduced by Mikolov et al. (2013), the **Word2Vec** framework marked a fundamental shift in the way Natural Language Processing (NLP) models learn from text. It demonstrated that semantic relationships between words can emerge purely from statistical co-occurrence, using a simple neural architecture and an elegant probabilistic objective. This section introduces the theoretical principles of Word2Vec, its two main training schemes, and the key intuitions behind why it works.

Motivation: Predicting Context from Words. At the core of Word2Vec lies the hypothesis that *semantic similarity arises from contextual similarity.* Words that appear in similar linguistic contexts tend to have similar meanings. Formally, given a sequence of words $(w_1, w_2, \ldots, w_T)$ extracted from a large corpus, the objective is to learn a function

$$f : \mathcal{V} \longrightarrow \mathbb{R}^p,$$

that maps each word $w \in \mathcal{V}$ from the vocabulary $\mathcal{V}$ (of size N) to a p-dimensional vector $f(w)$, called its **embedding**. The embedding space $\mathbb{R}^p$ is expected to capture syntactic and semantic regularities purely through statistical co-occurrence information.

7.1.1 The Continuous Bag-of-Words (CBOW) Model

The **Continuous Bag-of-Words (CBOW)** architecture predicts a **target word** w_t given its surrounding context words

$$(w_{t-C}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+C}),$$

where C is the context window size. The model treats the context as an unordered "bag" of words and uses the average of their embeddings to infer the central word.

Example 7.1.1. *For the sentence*

$$(\textit{the, cat, sat, on, the, mat}),$$

and a window size $C = 2$, the context for the target "sat" is:

$$\{\textit{the, cat, on, the}\}.$$

The CBOW model uses these context words to predict the central word "sat".

Architecture and Objective Function. Each context word w_{t+j} is represented by its one-hot vector $x_{t+j} \in \{0,1\}^N$. Let $V \in \mathbb{R}^{p\times N}$ be the **input embedding matrix**. The embedding of a word w is

$$v_w = Vx_w.$$

The hidden layer computes the average embedding of all context words:

$$h_t = \frac{1}{2C} \sum_{\substack{-C\le j\le C \\ j\neq 0}} v_{w_{t+j}}.$$

This h_t represents the context.

Output and Probability Distribution. Let $U \in \mathbb{R}^{p\times N}$ be the **output embedding matrix**. The network output before normalization is:

$$z = U^\top h_t, \quad z_i = u_i^\top h_t.$$

A softmax layer converts scores into probabilities:

$$\mathbb{P}(w_t = i \mid \text{context}) = \frac{\exp(u_i^\top h_t)}{\sum_{k=1}^N \exp(u_k^\top h_t)}.$$

Definition 7.1.1 (Loss Function)**.** *The CBOW model is trained by minimizing the negative log-likelihood of the true central word:*

$$\mathcal{L}_{CBOW} = -\sum_{t=1}^{T} \log \mathbb{P}(w_t \mid w_{t-C}, \ldots, w_{t+C}).$$

Interpretation and Properties. Words that occur in similar contexts yield similar hidden representations h_t. Thus, the embeddings v_w implicitly encode **distributional similarity**: the model learns to assign close vectors to words sharing comparable linguistic environments. The hidden representation

$$h_t = \sum_{\substack{-C\le j\le C \\ j\neq 0}} v_{w_{t+j}}$$

acts as a continuous analogue of a discrete word-count histogram. Each embedding $v_{w_{t+j}}$ contributes proportionally to the presence of its corresponding context word, which makes CBOW a smooth, differentiable version of the classical bag-of-words model.

Remark 8 (Loss Function as Negative Log-Likelihood). For every target position t in the corpus, the model predicts the probability $\mathbb{P}(w_t \mid \text{context})$ of the central word given its surrounding words. The quantity to minimize is the **negative log-likelihood** (or equivalently, the **anti-log-likelihood**):

$$\mathcal{L}_t = -\log \mathbb{P}(w_t \mid w_{t-C}, \ldots, w_{t+C}).$$

This formulation is natural because maximizing the likelihood corresponds to maximizing the model's confidence in the observed data. Taking the negative logarithm turns this maximization into a convex minimization problem, and penalizes more strongly low-probability (i.e., poorly predicted) words.

Key Insight. CBOW ignores word order entirely. Its predictive power arises solely from local co-occurrence patterns.

Optimization and Variants. The parameters (V, U) are optimized via stochastic gradient descent:

$$\arg\min_{V,U} \left\{ -\sum_{t=1}^{T} \log \frac{\exp(u_{w_t}^\top h_t)}{\sum_{i=1}^{N} \exp(u_i^\top h_t)} \right\}.$$

Each variable has an interpretation:

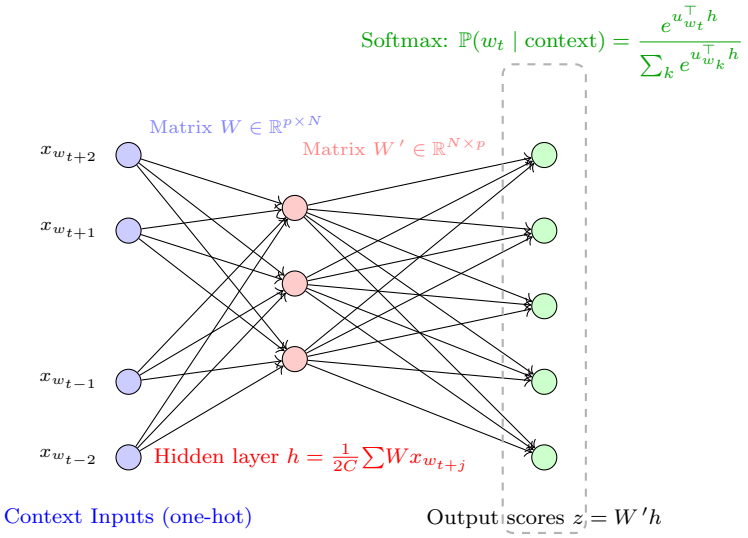
- $\mathcal{V}$: the vocabulary, with size $N = |\mathcal{V}|$.
- T: total number of tokens in the corpus.
- w_t: target (central) word at position t.
- u_{w_t}: output embedding vector (column of U).
- $v_{w_{t+j}}$: input embedding vector (column of V).
- C: context window size.
- h_t: averaged embedding of context words.

Each gradient update adjusts embeddings so that true target words receive higher probabilities given their contexts.

Remark 9 (Practical Extensions). Several modifications improve efficiency and quality:

- **Negative Sampling:** replaces full softmax with a sampled logistic loss.
- **Sub-sampling:** reduces over-representation of very frequent words.
- **Hierarchical Softmax:** replaces flat softmax with a binary tree factorization for faster training.

Summary of CBOW Architecture



Summary: Word2Vec CBOW Model

The **Continuous Bag of Words (CBOW)** model learns to predict a central target word from its surrounding context, providing distributed representations of words in a continuous vector space.

Context words: $(w_{t-C}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+C})$,

Input embeddings: $h_{t+j} = W x_{t+j}$,

Mean context vector: $h = \dfrac{1}{2C} \displaystyle\sum_{j \neq 0} h_{t+j}$,

Output scores: $z = W'h$,

Predicted probability: $\mathbb{P}(w_t \mid w_{t-C}, \ldots, w_{t+C}) = \dfrac{\exp(u_{w_t}^\top h)}{\sum_{i=1}^{N} \exp(u_i^\top h)}$,

Loss: $\mathcal{L} = -\log \mathbb{P}(w_t \mid \text{context})$.

Model parameters:

- $W \in \mathbb{R}^{p \times N}$: input embedding matrix,
- $W' \in \mathbb{R}^{N \times p}$: output embedding matrix,
- p: embedding dimension ($p \ll N$).

Objective:

$$\min_{W, W'} \; -\sum_{t=1}^{T} \log \mathbb{P}(w_t \mid w_{t-C}, \ldots, w_{t+C}).$$

Key insights:

- Context embeddings are averaged to form a single semantic representation.
- The model captures global co-occurrence structure efficiently.
- Negative sampling and hierarchical softmax enable large-scale training.
- CBOW and Skip-Gram share the same embedding matrices and probabilistic foundations.

7.1.2 Model Architecture: The Skip-Gram Formulation

Among the two variants of Word2Vec – Continuous Bag of Words (CBOW) and Skip-Gram – we now focus here on the Skip-Gram model, as it offers a clean probabilistic formulation.

Let C denote the context window size. At position t in the corpus, the model observes a target word w_t and attempts to predict each context word w_{t+j}, with $j \in \{-C, \ldots, -1, 1, \ldots, C\}$.

Formally, the model maximizes the conditional probability

$$\mathbb{P}(w_{t+j} \mid w_t),$$

that is, the probability of observing a context word w_{t+j} given the central word w_t.

Vector Representation and Network Structure Each word $w \in \mathcal{V}$ is represented by a **one-hot vector**

$$x_w = (0, \ldots, 0, 1, 0, \ldots, 0)^\top \in \{0, 1\}^N,$$

where the entry corresponding to w equals 1, and all others are 0.

The network consists of two linear layers:

$$x_w \xrightarrow{W} h \xrightarrow{W'} z,$$

where:

- $W \in \mathbb{R}^{p \times N}$ is the **input embedding matrix**;
- $W' \in \mathbb{R}^{N \times p}$ is the **output embedding matrix**;
- $h = Wx_w \in \mathbb{R}^p$ is the hidden representation (the embedding of the input word);
- $z = W'h \in \mathbb{R}^N$ contains the unnormalized scores for each possible output word.

Since x_w is one-hot, selecting x_{w_t} simply extracts the t-th column of W, that is,

$$h = v_{w_t} = W_{\bullet,t},$$

where v_{w_t} denotes the embedding vector of the word w_t.

From Scores to Probabilities The output vector z is transformed into a probability distribution over all vocabulary words using the Softmax function:

$$\mathbb{P}(w_i \mid w_t) = \frac{\exp(z_i)}{\sum_{k=1}^{N} \exp(z_k)} = \frac{\exp(u_{w_i}^\top v_{w_t})}{\sum_{k=1}^{N} \exp(u_{w_k}^\top v_{w_t})},$$

where u_{w_i} denotes the i-th row of W', i.e., the **output embedding** of word w_i.

Thus, the network defines a conditional probability model:

$$\mathbb{P}(w_{t+j} \mid w_t) = \text{Softmax}(W'Wx_{w_t}).$$

Objective Function For a given pair (w_t, w_{t+j}), we define the loss as the negative log-likelihood of observing the true context word:

$$\mathcal{L}(w_t, w_{t+j}) = -\log \mathbb{P}(w_{t+j} \mid w_t) = -\log \frac{\exp(u_{w_{t+j}}^\top v_{w_t})}{\sum_{k=1}^{N} \exp(u_{w_k}^\top v_{w_t})}.$$

Over the entire corpus, the total loss is the expected negative log-likelihood:

$$\mathcal{L}_{\text{total}} = -\sum_{t=1}^{T} \sum_{\substack{j=-C \\ j \neq 0}}^{C} \log \mathbb{P}(w_{t+j} \mid w_t).$$

Minimizing this loss over all parameters $\{W, W'\}$ using stochastic gradient descent yields embeddings v_{w_t} that capture semantic regularities through co-occurrence prediction.

Dimensionality and Interpretation The dimension p of the hidden layer defines the dimension of the embedding space $\mathbb{R}^p$. It is typically much smaller than the vocabulary size N, i.e.,

$$p \ll N.$$

Hence, the matrices W and W' act as low-rank factorizations of the word-context co-occurrence structure. After training, the column vectors of W (or, equivalently, the row vectors of W') are used as the learned word embeddings.

7.1.3 Computational Considerations: Approximating the Softmax

The softmax normalization over all N words is computationally expensive. Two efficient approximations are commonly used in practice:

1. **Hierarchical Softmax:** factorizes $\mathbb{P}(w_i \mid w_t)$ through a binary tree representation, reducing the complexity from $\mathcal{O}(N)$ to $\mathcal{O}(\log N)$.

2. **Negative Sampling:** replaces the full softmax objective with a logistic regression objective distinguishing true (w_t, w_{t+j}) pairs from randomly sampled "negative" pairs. For each observed pair (w_t, w_{t+j}), the model is trained to:

   - assign a high score $u_{w_{t+j}}^\top v_{w_t}$ when the words co-occur in the corpus;
   - assign low scores to randomly sampled word pairs (w_t, w_k) drawn from a *noise distribution* P_n.

   The resulting local objective is:

$$\mathcal{L}_{\text{NS}}(w_t, w_{t+j}) = -\log \sigma(u_{w_{t+j}}^\top v_{w_t}) - \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n} \left[\log \sigma(-u_{w_k}^\top v_{w_t}) \right],$$

   where:

   - $\sigma(x) = (1 + e^{-x})^{-1}$ is the sigmoid function;
   - K is the number of negative samples per positive pair;
   - P_n is a **noise distribution over the vocabulary** used to sample negative examples.

   The choice of P_n crucially affects both training efficiency and embedding quality. Empirical results show that drawing negatives in proportion to the word frequencies $\text{f}(w_i)$ raised to the $3/4$ power,

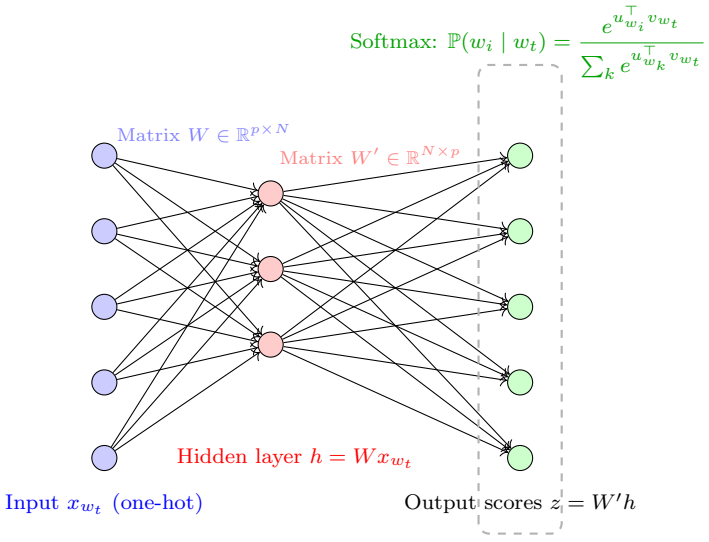
$$P_n(w_i) = \frac{\text{f}(w_i)^{3/4}}{\sum_{j=1}^{N} \text{f}(w_j)^{3/4}},$$

   yields superior performance compared to uniform or purely frequency-based sampling (Goldberg and Levy, 2014). This exponent balances between:

   - down-weighting extremely frequent words (which otherwise dominate the training process),
   - and still allowing them to appear often enough as negatives to shape meaningful contrasts.

   Negative sampling thus allows Word2Vec to learn high-quality embeddings efficiently even for large vocabularies.

Summary of Skip-Gram Architecture



Summary: Word2Vec Skip-Gram Model

The **Word2Vec Skip-Gram model** learns distributed representations of words by predicting context words from a central target word within a fixed window. It optimizes a probabilistic objective derived from co-occurrence prediction.

$$
\begin{aligned}
&\textbf{Input: } x_{w_t} \in \{0,1\}^N, \quad \text{(one-hot encoding of target word)} \\
&\textbf{Hidden representation: } h = W x_{w_t} \in \mathbb{R}^p, \quad \text{(embedding lookup)} \\
&\textbf{Output scores: } z = W' h \in \mathbb{R}^N, \\
&\textbf{Predicted probabilities: } \mathbb{P}(w_i \mid w_t) = \frac{\exp(u_{w_i}^\top v_{w_t})}{\sum_{k=1}^N \exp(u_{w_k}^\top v_{w_t})}, \\
&\textbf{Loss: } \mathcal{L}(w_t, w_{t+j}) = -\log \mathbb{P}(w_{t+j} \mid w_t).
\end{aligned}
$$

Model parameters:

- $W \in \mathbb{R}^{p \times N}$: input embedding matrix (columns = word embeddings),
- $W' \in \mathbb{R}^{N \times p}$: output embedding matrix (rows = context embeddings),
- p: embedding dimension, typically $p \ll N$.

Training objective:

$$
\min_{W, W'} \ \mathcal{L}_{\text{total}} = -\sum_{t=1}^{T} \sum_{\substack{j=-C \\ j \neq 0}}^{C} \log \mathbb{P}(w_{t+j} \mid w_t).
$$

Interpretation:

- The model performs a low-rank factorization of the word–context co-occurrence matrix.
- Semantic similarity arises as a by-product of predicting contextual neighbors.
- After training, each column of W serves as a dense semantic embedding of a word.

Practical optimizations:

- The full softmax is computationally heavy ($\mathcal{O}(N)$).
- Efficient approximations:
  - *Hierarchical Softmax*: $\mathcal{O}(\log N)$.
  - *Negative Sampling*: replaces full normalization with logistic objectives over sampled negatives.

In this framework, embeddings are not explicitly constrained to encode semantics; rather, semantic regularities *emerge* as a by-product of optimizing a purely probabilistic prediction task.

From Random Matrix to Semantic Geometry. Let $W \in \mathbb{R}^{|\mathcal{V}| \times p}$ denote the embedding matrix at initialization, whose entries are drawn randomly. During training with the Skip-Gram Negative Sampling objective, each occurrence of a word–context pair (w_I, w_O) increases their inner product $v_{w_I}^\top v_{w_O}$, whereas each sampled negative pair decreases it. Over many stochastic gradient updates, the embeddings evolve toward a configuration that reflects empirical co-occurrence statistics. As shown by Levy and Goldberg (2014), at convergence the optimal inner product satisfies approximately:

$$
v_{w_I}^\top v_{w_O} \ \approx \ \log \frac{\mathbb{P}(w_O \mid w_I)}{\mathbb{P}(w_O)} \ = \ \mathrm{PMI}(w_I, w_O),
$$

where PMI denotes the so-called **Pointwise Mutual Information** between the input word w_I and the output (context) word w_O. Hence, the Word2Vec model implicitly performs a low-rank factorization of the shifted PMI matrix of word co-occurrences, with the optimization carried out by stochastic gradient descent rather than explicit matrix decomposition. This equivalence elegantly bridges neural and statistical approaches: Word2Vec can be viewed as a predictive neural reformulation of traditional distributional semantics.

7.2 GloVe (Pennington et al., 2014)

7.2.1 Motivation: Combining Local and Global Statistics

While Word2Vec learns from local context windows, **GloVe** (*Global Vectors for Word Representation*) (Pennington et al., 2014) incorporates both local and global corpus statistics. The intuition is that word meaning is better captured when embeddings encode the ratios of co-occurrence probabilities between words.

7.2.2 Mathematical Formulation

Definition 7.2.1 (Co-occurrence Matrix)**.** *Let $\mathcal{V}$ be the vocabulary of size N. GloVe constructs a symmetric word–context matrix $X \in \mathbb{R}^{N\times N}$ where*

$$X_{ij} = \textit{number of times word } j \textit{ appears in the context of word } i.$$

Each entry counts how often j occurs within a fixed-size window centered on i.

Example 7.2.1. *Consider the sentence "The cat sat on the mat." with a window size of 2. For the target word "cat", the context is $\{The, sat\}$. Hence, the corresponding co-occurrence entries increase: $X_{cat,The} \uparrow$ and $X_{cat,sat} \uparrow$.*

The total number of context co-occurrences involving word i is

$$X_i = \sum_{k=1}^{N} X_{ik}.$$

From these, we define the empirical co-occurrence probabilities:

$$P_{ij} = \frac{X_{ij}}{X_i},$$

which represents the probability that word j appears near word i.

Remark 10 (Key Intuition). The ratio of co-occurrence probabilities

$$\frac{P_{ik}}{P_{jk}}$$

encodes semantic relationships between words i and j. For example, if k is the word "solid", then $\frac{P_{\text{ice,solid}}}{P_{\text{steam,solid}}}$ is large, while $\frac{P_{\text{ice,gas}}}{P_{\text{steam,gas}}}$ is small. This suggests that "solid" relates more strongly to "ice" than to "steam", whereas "gas" relates more strongly to "steam". Hence, semantic meaning can be understood through these ratios rather than absolute frequencies.

Proposition 7.2.1 (Core Modeling Principle). *Let X_{ij} denote the number of times word j appears in the context of word i. GloVe aims to find embeddings $w_i, \tilde{w}_j \in \mathbb{R}^p$ and biases $b_i, \tilde{b}_j \in \mathbb{R}$ such that*

$$w_i^\top \tilde{w}_j + b_i + \tilde{b}_j \approx \log X_{ij}.$$

The model is trained by minimizing the following weighted least-squares cost function:

$$\mathcal{L} = \sum_{i,j} f(X_{ij}) \left(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2.$$

Remark 11 (Interpretation). The dot product $w_i^\top \tilde{w}_j$ measures the compatibility between words i and j, while $\log X_{ij}$ quantifies the strength of their empirical association. The equality $w_i^\top \tilde{w}_j \approx \log X_{ij}$ therefore establishes a linear correspondence between geometric similarity and statistical co-occurrence, providing a direct bridge between distributional statistics and vector geometry.

7.2.3 From Ratios to Linear Geometry

If word meaning is expressed through co-occurrence ratios, we may posit a functional relation

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}.$$

To embed this in a vector space, F should depend on differences of word vectors, so that directions capture semantic relations. A natural formulation is:

$$F(w_i, w_j, \tilde{w}_k) = \exp\big((\tilde{w}_k)^\top (w_i - w_j)\big).$$

Taking logarithms gives:

$$(\tilde{w}_k)^\top (w_i - w_j) = \log P_{ik} - \log P_{jk}.$$

Rearranging terms leads directly to the approximation:

$$w_i^\top \tilde{w}_j + b_i + \tilde{b}_j = \log X_{ij}.$$

Hence, the GloVe model can be viewed as a **linearization of co-occurrence ratios**, where each dot product between word vectors predicts the logarithm of their co-occurrence frequency.

7.2.4 Weighting Function and Frequency Balance

Definition 7.2.2 (Weighting Function). *The function $f(X_{ij})$ controls the relative influence of each co-occurrence pair in the loss:*

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

Typical parameter values are $\alpha = \frac{3}{4}$ and $x_{max} \approx 100$ (Pennington et al., 2014).

Remark 12 (Purpose of $f(x)$).

- Very frequent pairs (large X_{ij}) are informative but should not dominate training.
- Rare pairs (small X_{ij}) are often noisy and must be down-weighted.
- The function $f(x)$ provides a smooth balance between both extremes, ensuring stable and efficient learning across frequency scales.

7.2.5 Interpretation and Theoretical Connection

GloVe thus performs a **log-linear factorization** of the co-occurrence matrix X. The model combines the predictive nature of Word2Vec (which relies on local context windows) with the statistical grounding of matrix factorization methods (which exploit the entire corpus distribution).

A short summary. In three points:

- GloVe unifies local and global statistical information.
- It learns embeddings whose dot products reproduce the logarithm of word co-occurrences.
- The weighting function $f(x)$ regularizes the importance of frequency, balancing precision and stability.

Let X_{ij} denote the number of times word j appears in the context of word i, and $P_{ij} = X_{ij}/X_i$ the conditional co-occurrence probability. The model seeks embeddings $w_i, \tilde{w}_j$ such that their dot product approximates the logarithm of co-occurrence:

$$w_i^\top \tilde{w}_j + b_i + \tilde{b}_j \approx \log X_{ij}.$$

The loss function is a weighted least-squares objective:

$$\mathcal{L} = \sum_{i,j} f(X_{ij}) \big(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \big)^2,$$

where $f(x)$ downweights rare and overly frequent co-occurrences.

Interpretation. GloVe can be seen as a **log-linear factorization** of the co-occurrence matrix. It merges the efficiency of count-based methods with the predictive power of Word2Vec. The model captures linear semantic regularities (e.g., king − man + woman ≈ queen) as geometric properties of the embedding space.

7.3 FastText (Bojanowski et al., 2017)

7.3.1 Motivation: Incorporating Morphology

While Word2Vec and GloVe treat words as atomic units, they struggle with rare or unseen words. **FastText** (Bojanowski et al., 2017) addresses this by integrating subword information, making it more robust to morphology and spelling variations.

7.3.2 Architecture and Training

Each word is represented as the sum of its character n-gram embeddings. For example, the word "apple" with $n = 3$ would include subwords: "¡ap", "app", "ppl", "ple", "le¿". The final word vector is:

$$v_{\text{apple}} = \sum_{g \in G_{\text{apple}}} z_g,$$

where G_{apple} is the set of n-grams and z_g their embeddings. The same skip-gram objective as in Word2Vec is then used.

Advantages. This approach:

- captures morphological and orthographic structure;
- handles rare or unseen words gracefully;
- produces more meaningful embeddings for languages with rich morphology.

FastText thus extends the predictive embedding paradigm to the subword level, improving linguistic coverage and generalization.

7.4 ELMo (Peters et al., 2018)

7.4.1 Motivation: Contextual Embeddings

Previous models assign one static vector per word, regardless of context. However, the meaning of a word often depends on its sentence. **ELMo** (*Embeddings from Language Models*) (Peters et al., 2018) introduced the notion of **contextual embeddings**, where each token's representation depends on its surrounding words.

7.4.2 Bidirectional LSTM Language Model

ELMo is trained as a **bidirectional language model**:

$$\mathbb{P}(w_1, \ldots, w_T) = \prod_{t=1}^{T} \mathbb{P}(w_t \mid w_{<t}) + \prod_{t=1}^{T} \mathbb{P}(w_t \mid w_{>t}),$$

where the first term is modeled by a forward LSTM and the second by a backward LSTM. Each layer of the LSTM produces hidden states encoding different levels of abstraction. The final ELMo embedding for word t is a weighted combination:

$$\text{ELMo}_t = \gamma \sum_{l=0}^{L} s_l h_{t,l},$$

where $h_{t,l}$ is the hidden state at layer l, s_l are learned scalar weights, and γ a scaling factor.

Impact. ELMo demonstrated that deep contextualized embeddings significantly improve performance across NLP tasks such as sentiment analysis, question answering, and named entity recognition. It marked the transition from static to dynamic representations of language.

7.5 BERT (Devlin et al., 2019)

Motivation: Deep Contextualization and Bidirectionality. **BERT** (*Bidirectional Encoder Representations from Transformers*) (Devlin et al., 2019) represents the culmination of the pretraining paradigm: learning rich, general-purpose language representations that can be fine-tuned for downstream tasks. BERT leverages the **Transformer encoder** architecture (Vaswani et al., 2017), enabling bidirectional attention over text.

7.5.1 Model Architecture

BERT is built from multiple Transformer encoder layers. Each layer applies:

$$\text{Self-Attention: } \text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V,$$

followed by feedforward transformations and normalization. Unlike unidirectional models, BERT attends to both left and right context simultaneously.

7.5.2 Training Objectives

BERT introduces two key pretraining tasks:

Masked Language Modeling (MLM): A random subset of tokens (typically 15%) is replaced by a special [MASK] token. The model must predict the original word:

$$\mathcal{L}_{\text{MLM}} = -\sum_{t \in M} \log \mathbb{P}(w_t \mid w_{\setminus t}).$$

Next Sentence Prediction (NSP): Given two sentences A and B, the model predicts whether B follows A in the original text. This encourages learning of inter-sentence relations.

Impact and Legacy. BERT's deep bidirectional architecture set a new standard in NLP:

- It provided universal pretrained representations transferable to multiple tasks.
- It demonstrated that large-scale unsupervised pretraining followed by fine-tuning could outperform task-specific models.
- It laid the foundation for successor models such as RoBERTa, ALBERT, and GPT series.

BERT thus unified earlier ideas – predictive modeling (Word2Vec), contextualization (ELMo), and attention mechanisms (Transformers) – into a single coherent framework that defines modern NLP.

Conclusion of the Chapter

From Word2Vec to BERT, the evolution of word embeddings reflects a continuous shift:

- from local to global representations (Word2Vec $\rightarrow$ GloVe),
- from static to contextual (ELMo),
- from shallow to deeply pretrained architectures (BERT).

Each step integrated new insights from probability, optimization, and representation learning, progressively bridging linguistic meaning and geometric structure. These models collectively form the conceptual backbone of contemporary NLP.

Chapter 8

Smooth Inverse Frequency: A Simple but Robust Sentence Representations

In Arora et al. (2017), the author proposed a remarkably elegant and theoretically motivated method for computing sentence embeddings. The method, called the **Smooth Inverse Frequency (SIF)** model, provides a simple unsupervised approach to generate sentence-level representations v_s from pre-trained word embeddings v_w. Despite its simplicity, SIF was shown to outperform more complex neural architectures such as LSTMs and RNNs at the time, while being computationally efficient and interpretable.

8.1 From Word Embeddings to Sentence Representations

Motivation. Word embeddings such as Word2Vec or GloVe provide vector representations $v_w \in \mathbb{R}^d$ that capture the semantics of individual words. However, many NLP tasks (e.g., sentiment analysis, document classification, or similarity measurement) require representations at the sentence or paragraph level.

A naive approach is to take the simple average of all word embeddings in a sentence:

$$v_s = \frac{1}{|s|} \sum_{w \in s} v_w.$$

While intuitive, this unweighted mean ignores the varying informativeness of words – frequent words such as "the", "and", or "is" dominate the representation despite carrying little semantic content.

To address this limitation, Arora et al. introduced a weighting scheme that reduces the influence of frequent words while preserving sentence-level meaning.

Smooth Inverse Frequency (SIF) Weighting. The SIF weighting function assigns each word a weight inversely proportional to its frequency in the corpus:

$$\text{weight}(w) = \frac{a}{p(w) + a},$$

where:

- $p(w)$ is the unigram probability (empirical frequency) of the word w,
- a is a small smoothing hyperparameter, typically $a = 10^{-3}$.

This weighting attenuates the impact of very frequent words and enhances the contribution of rare, content-bearing ones. The final SIF sentence embedding is given by:

$$v_s = \frac{1}{|s|} \sum_{w \in s} \frac{a}{p(w) + a} v_w. \tag{8.1}$$

Equation (8.1) defines a weighted average of the constituent word embeddings, yielding a continuous vector representation of the sentence.

8.2 The Generative Model Behind SIF

Latent Discourse Vector. Arora et al. grounded SIF in a probabilistic generative framework for text. They postulated that every sentence (or short discourse) is governed by a latent **discourse vector** $c_t \in \mathbb{R}^d$ that slowly evolves across positions t within the sentence. This vector represents the semantic topic or intent driving the word generation process.

Each word w is associated with a fixed embedding v_w, independent of t. The word emitted at position t is then generated from the log-linear probability:

$$\mathbb{P}(w_t \mid c_t) \propto \exp(\langle c_t, v_w \rangle), \tag{8.2}$$

a formulation first explored in the log-linear language models of Mnih and Hinton (2008), and known to approximate well models such as Word2Vec and GloVe (Arora et al., 2016).

Maximum a Posteriori (MAP) Estimation of Sentence Embeddings. Given a sentence $s = (w_1, \ldots, w_T)$, the goal is to infer its latent discourse vector v_s that maximizes the posterior probability:

$$\widehat{v}_s^{\text{MAP}} = \arg\max_{v_s} \Big\{ \log \mathbb{P}(v_s) + \sum_{w \in s} \log \mathbb{P}(w \mid v_s) \Big\}. \tag{8.3}$$

Assuming a uniform prior on $\mathbb{P}(v_s)$, the MAP estimator is equivalent to the Maximum Likelihood Estimator (MLE). Thus, v_s is the vector that maximizes the likelihood of the observed words under the generative model (8.2).

Discourse Smoothing and Background Context. The model also incorporates a "smoothing" mechanism to account for general words that appear independently of the specific context. It assumes that at each position t, the effective discourse vector is a convex combination:

$$\widetilde{c}_s = \beta c_0 + (1 - \beta) c_s, \tag{8.4}$$

where c_0 represents a background context capturing common discourse directions, and $\beta \in [0, 1]$ controls the smoothing strength.

The resulting probabilistic model for word emission is:

$$\mathbb{P}(w \mid c_s) = \alpha\, p(w) + (1 - \alpha)\, \frac{\exp(\langle \widetilde{c}_s, v_w \rangle)}{Z_{\widetilde{c}_s}}, \tag{8.5}$$

where:

- $p(w)$ is the empirical unigram frequency of w,
- α and β are barycentric hyperparameters,
- $Z_{\tilde{c}_s} = \sum_{w \in \mathcal{V}} \exp(\langle \tilde{c}_s, v_w \rangle)$ is the partition function, assumed constant under the hypothesis that word embeddings are uniformly distributed on the unit sphere.

Equation (8.5) expresses the word probability as a balance between:

- a frequency-based component ($\alpha p(w)$), and
- a contextual component determined by the softmax probability.

Linearization and Derivation of SIF. Taking the log-likelihood of (8.5) and applying a first-order Taylor approximation yields:

$$f_w(\widetilde{c}_s) = \log\left[\alpha p(w) + (1-\alpha)\frac{\exp(\langle v_w, \widetilde{c}_s\rangle)}{Z}\right] \approx \text{const} + \frac{a}{p(w)+a}\,\langle v_w, \widetilde{c}_s\rangle,$$

where $a = \frac{1-\alpha}{\alpha Z}$. Maximizing this approximate objective leads to:

$$v_s \propto \sum_{w\in s} \frac{a}{p(w)+a}\, v_w.$$

This gives exactly the weighted mean defined in Equation (8.1).

8.3 The Final SIF Embedding and Its Interpretation

Weighted Mean Representation. The final SIF embedding is thus:

$$v_s = \frac{1}{|s|}\sum_{w\in s} \frac{a}{p(w)+a}\, v_w. \tag{8.6}$$

It can be interpreted as a smoothed, frequency-aware average of word embeddings. Frequent words with high $p(w)$ are downweighted, ensuring that rare, informative words dominate the sentence representation.

Removing the Common Component. Arora et al. also proposed an additional refinement: subtracting the first principal component of all sentence embeddings in the corpus. This step removes the global "common discourse direction" corresponding to generic or stop words (e.g., "the", "is", "and"), further improving discriminative power:

$$v_s^{\text{final}} = v_s - (u_1^\top v_s)u_1,$$

where u_1 is the top principal component.

Empirical Success. Despite its simplicity, the SIF method performs remarkably well on tasks such as semantic textual similarity and document classification. Its success can be attributed to:

- Theoretical grounding in probabilistic modeling.
- Simplicity of implementation – only pre-trained word embeddings and corpus frequencies are needed.
- Robustness – no supervised fine-tuning required.

In practice, SIF embeddings have served as a strong baseline and an interpretable alternative to more complex contextual models.

8.4 Connection to Energy-Based and Softmax Models

The log-linear form (8.2) connects SIF embeddings to the family of **energy-based models (EBM)** discussed earlier. The probability of emitting a word is expressed as:

$$\mathbb{P}(w \mid c_s) \propto \exp(-E(w, c_s)),$$

where the energy function is defined by the negative inner product:

$$E(w, c_s) = -\langle v_w, c_s\rangle.$$

Minimizing energy corresponds to aligning v_w and c_s, just as maximizing likelihood encourages high dot products between context and word embeddings. The SIF model can thus be viewed as a linearized and frequency-smoothed version of an EBM.

8.5 Summary and Impact

The SIF method occupies a special place in the evolution of sentence embeddings:

- It is theoretically motivated by a probabilistic generative model.
- It produces compact, interpretable representations using simple weighted averages.
- It introduced the principle of removing common discourse components, later reused in contextual embedding architectures.

Key insight: complex language behavior can often be captured by simple, mathematically principled structures when the model is properly regularized and interpreted geometrically.

The work of Arora et al. (2017) thus bridges two worlds: on one side, the statistical physics view of word generation via energy landscapes; on the other, the geometric view of meaning as position in a semantic space. This synthesis established one of the most elegant formulations of sentence embedding in modern NLP.

8.6 The [CLS] Token: A Learned Alternative to SIF for Sentence Embeddings

Motivation: From Averaging to Learned Representations. The SIF method introduced a simple, theoretically grounded way to obtain sentence embeddings by computing a weighted average of static word vectors. However, with the rise of deep contextual models such as BERT (Devlin et al., 2019), a new paradigm emerged: **learning** a specific vector representation for an entire sequence through attention-based mechanisms rather than hand-crafted averaging.

The [CLS] token (*classification token*) embodies this paradigm shift. It provides a learned, model-internal representation of a sentence or document, trained end-to-end through supervision or pretraining tasks.

Definition and Role in Transformers. In Transformer-based models, each input sequence is first tokenized and augmented with a special token [CLS] placed at the beginning:

$$\texttt{[CLS]}\ w_1\ w_2\ \ldots\ w_T.$$

This token does not correspond to any real word – it is an artificial symbol whose sole purpose is to aggregate information from the entire sequence.

Through the Transformer's self-attention layers (Vaswani et al., 2017), every token (including [CLS]) attends to every other token. At each layer, the [CLS] representation is updated by weighted combinations of all other tokens' contextualized embeddings. After L layers, the final hidden state of the [CLS] token, usually denoted $h_{\texttt{[CLS]}}$, serves as a **sentence-level embedding** summarizing the full input.

$$v_s^{\text{CLS}} = h_{\texttt{[CLS]}}^{(L)}. \tag{8.7}$$

This vector plays the same conceptual role as the SIF embedding v_s – a compact numerical representation of the sentence – but is obtained through end-to-end training rather than heuristic averaging.

Learning the [CLS] Representation. The [CLS] token is learned implicitly through the model's pretraining objectives. During BERT's training, the network optimizes two main losses:

- **Masked Language Modeling (MLM):** Predicting masked words in the input sequence encourages each token (including [CLS]) to gather global contextual information.
- **Next Sentence Prediction (NSP):** A binary classification task is applied directly on the [CLS] vector, asking whether two sentences appear consecutively in the corpus.

The second objective explicitly forces the [CLS] embedding to encode inter-sentential semantics. Thus, the [CLS] token learns to represent the entire sentence in a way that is discriminative for higher-level tasks.

Fine-Tuning and Task-Specific Adaptation. Once pretrained, the `[CLS]` embedding can be reused for downstream tasks. For example, in sentiment classification, one simply appends a linear classifier on top of $h_{\texttt{[CLS]}}$:

$$\hat{y} = \text{softmax}(W h_{\texttt{[CLS]}} + b),$$

and fine-tunes the entire model (including embeddings) using cross-entropy loss. Through this fine-tuning process, the model adapts the `[CLS]` representation to encode task-relevant features such as polarity, topic, or entailment relations.

Comparison with SIF Embeddings. Although both SIF and `[CLS]` provide sentence-level representations, their underlying philosophies differ profoundly:

Aspect	**SIF (Arora, 2017)**	**`[CLS]` Token (BERT, 2019)**
Type of model	Unsupervised, linear	Deep contextual, transformer-based
Representation principle	Weighted average of word embeddings	Learned contextual embedding via self-attention
Information aggregation	Static, independent of context beyond word frequencies	Dynamic, attends to all tokens jointly
Training	No fine-tuning required (post hoc)	Learned end-to-end during pre-training and fine-tuning
Interpretability	Simple, transparent geometric averaging	Opaque but expressive and adaptive
Computation cost	Very low (no neural inference)	High (requires transformer inference)

Geometric and Statistical Interpretation. From a geometric viewpoint, the SIF embedding v_s lies in the convex hull of the word vectors – it is an explicit barycenter of the sentence. In contrast, the `[CLS]` representation v_s^{CLS} is an **implicit learned barycenter**: its coordinates are not directly defined by the words' embeddings but by the network's internal attention weights and learned transformations.

Both aim to capture the "center of meaning" of a sentence, but:

- SIF relies on external frequency statistics $p(w)$,
- `[CLS]` relies on self-attention to infer which tokens are most relevant.

Thus, the `[CLS]` vector can be viewed as a **neural generalization of the SIF concept**, where weights are learned instead of predefined.

Conceptual Summary. The introduction of the `[CLS]` token represents a conceptual transition in NLP sentence modeling:

- from fixed, interpretable averages (SIF) to adaptive, data-driven representations (BERT);
- from external corpus-based weighting to intrinsic, self-attention weighting;
- from shallow geometric formulations to deep learned contextual embeddings.

Both approaches remain valuable:

- **SIF** illustrates how sentence meaning can emerge from linear combinations of static word vectors, grounded in probabilistic theory.
- `[CLS]` demonstrates how modern transformers internalize this aggregation process through optimization and attention.

In summary, the SIF embedding and the `[CLS]` token embody two complementary paradigms for representing sentences – one mathematically transparent, the other empirically powerful – together forming a comprehensive understanding of sentence-level representation learning in NLP.

Chapter 9

Named Entity Recognition

9.1 Named Entity Recognition (NER): Identifying Meaningful Entities in Text

9.1.1 Motivation: From Words to Real-World Concepts

While language models capture statistical and semantic patterns in text, many NLP applications require **explicit identification of real-world entities**. For instance, consider the sentence:

"Dr. Smith works at the University of Lyon and studied in Paris."

A human immediately recognizes that:

- *Dr. Smith* refers to a **person**,
- *University of Lyon* refers to an **organization**,
- *Paris* refers to a **location**.

The goal of **Named Entity Recognition (NER)** is to teach a computer to perform this same identification automatically.

In short, NER bridges the gap between raw text and structured information: it transforms unstructured linguistic data into machine-readable knowledge.

9.1.2 Definition and Task Formulation

Named Entity Recognition (NER) is a core NLP task consisting in:

> *Detecting and classifying words or phrases that refer to specific entities (people, organizations, places, dates, etc.) within a text.*

Formally, given a sequence of tokens

$$x = (x_1, x_2, \ldots, x_T),$$

the model must predict a sequence of corresponding entity labels

$$y = (y_1, y_2, \ldots, y_T),$$

where each y_t belongs to a finite set of entity types such as:

PER (Person), ORG (Organization), LOC (Location), DATE, O (Other).

A common labeling format is the **BIO scheme** (Begin, Inside, Outside):

- `B-PER` marks the beginning of a person name,
- `I-PER` marks subsequent tokens inside the same entity,
- `O` marks tokens that do not belong to any entity.

Example:

"Barack Obama was born in Hawaii" ⇒ [`B-PER, I-PER, O, O, O, B-LOC`].

9.1.3 Why NER Matters in NLP

NER is a foundational step in many high-level NLP systems. It enables algorithms to extract structured knowledge from raw text and reason about entities rather than isolated words.

Typical applications include:

- **Information extraction:** automatically identifying key actors, places, or organizations in documents.
- **Question answering:** linking entities in questions (e.g., "Who founded OpenAI?") to knowledge bases.
- **Search and recommendation:** enriching queries by recognizing names, brands, or products.
- **Biomedical and legal NLP:** detecting patient names, drug mentions, gene symbols, or case identifiers.

9.1.4 Relation to Other NLP Tasks

From a modeling perspective, NER is a **sequence labeling problem**. It shares conceptual similarities with:

- **Part-of-speech (POS) tagging** – assigning grammatical categories (noun, verb, etc.),
- **Chunking or syntactic parsing** – grouping tokens into phrases,
- **Sentiment tagging** – associating emotional or evaluative labels to tokens or spans.

However, unlike these tasks, NER is explicitly **semantic**: its goal is to connect words to entities that exist in the world.

9.1.5 From Rules to Deep Learning

Historically, NER systems relied on:

- hand-crafted linguistic rules and dictionaries (symbolic era),
- probabilistic models such as Hidden Markov Models (HMMs) or Conditional Random Fields (CRFs),
- and more recently, deep neural architectures based on **BiLSTMs** or **Transformers** (e.g., BERT-NER).

Modern approaches treat NER as a neural sequence tagging task:

$$h_t = f_\theta(x_t, h_{t-1}) \quad \Rightarrow \quad p(y_t \mid x) = \text{softmax}(W h_t + b),$$

where f_θ may be a recurrent or transformer-based encoder. The model is trained using the **cross-entropy loss** over all token positions:

$$\mathcal{L} = -\sum_{t=1}^{T} \log p(y_t^{\text{true}} \mid x).$$

9.1.6 Interpretation

NER provides a tangible example of how **machine learning models learn to link text to meaning**. It requires understanding not only the words themselves, but also their relationships and syntactic roles within context.

In practical NLP pipelines, NER often serves as a first stage of *semantic grounding*: it converts linguistic surface forms into interpretable entities, enabling higher-level reasoning, information retrieval, and knowledge graph construction.

9.1.7 Summary

- **Goal:** Identify and classify real-world entities in text.
- **Input:** Sequence of tokens.
- **Output:** Sequence of entity labels (e.g., PER, ORG, LOC).
- **Method:** Sequence tagging model trained with cross-entropy.
- **Applications:** Information extraction, question answering, biomedical text mining.

NER exemplifies the transition from purely syntactic processing to genuine semantic understanding in NLP, bridging linguistic patterns and structured world knowledge.

9.2 Modern Neural Architectures for Named Entity Recognition

9.2.1 From Sequential Models to Contextual Encoders

Traditional sequence labeling models such as Hidden Markov Models (HMMs) and Conditional Random Fields (CRFs) relied on handcrafted features and limited contextual windows. Modern NER systems instead learn distributed representations of tokens and exploit long-range dependencies via deep neural networks.

The typical neural NER pipeline follows three stages:

Input tokens $\longrightarrow$ Contextual encoder (BiLSTM or Transformer) $\longrightarrow$ Label decoder (Softmax or CRF).

Each component plays a distinct mathematical role in modeling linguistic and semantic dependencies.

9.2.2 Bidirectional LSTM-CRF Model

One of the most influential architectures for NER is the **BiLSTM–CRF** model (**??**). It combines recurrent contextual encoding with structured prediction at the output layer.

Step 1. Embedding Layer. Each token x_t is mapped to a dense embedding vector $e_t \in \mathbb{R}^d$. This embedding may come from:

- pretrained word vectors (Word2Vec, GloVe, FastText),
- character-level CNNs or LSTMs capturing morphology,
- or contextual embeddings (e.g., from BERT or ELMo).

Step 2. Contextual Encoding via BiLSTM. A **bidirectional LSTM** processes the sequence in both directions to incorporate past and future context:

$$\overrightarrow{h_t} = \mathrm{LSTM}_{\mathrm{fwd}}(e_t, \overrightarrow{h_{t-1}}), \qquad \overleftarrow{h_t} = \mathrm{LSTM}_{\mathrm{bwd}}(e_t, \overleftarrow{h_{t+1}}),$$

and the concatenated hidden state

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$$

forms a context-sensitive representation of token x_t.

Step 3. Linear Transformation to Emission Scores. Each contextual vector h_t is projected into a space of label scores:

$$s_t = W h_t + b, \quad s_t \in \mathbb{R}^{|\mathcal{Y}|},$$

where $|\mathcal{Y}|$ is the number of possible entity tags (e.g., `B-PER`, `I-ORG`, `O`, etc.).

Step 4. Sequence-Level Decoding with CRF. Rather than predicting each label independently (via softmax), the CRF layer models correlations between neighboring labels. For example, it prevents invalid transitions such as `I-PER` immediately following `O`.

Let $s_t(y_t)$ denote the emission score for label y_t at position t, and let A_{y_{t-1},y_t} be a learned transition score between consecutive labels. The total score of a label sequence $y = (y_1, \ldots, y_T)$ is:

$$\text{Score}(x, y) = \sum_{t=1}^{T} \big(A_{y_{t-1},y_t} + s_t(y_t)\big).$$

The CRF defines a probability distribution over all possible label sequences:

$$\mathbb{P}(y \mid x) = \frac{\exp(\text{Score}(x, y))}{\sum_{y'} \exp(\text{Score}(x, y'))}.$$

Training maximizes the log-likelihood of the correct label sequence:

$$\mathcal{L}(\theta) = -\log \mathbb{P}(y^{\text{true}} \mid x).$$

At inference time, the most likely label sequence is found using the **Viterbi algorithm**:

$$\hat{y} = \arg\max_{y} \text{Score}(x, y).$$

Interpretation. The BiLSTM captures the contextual dependencies between words, while the CRF enforces global consistency between output labels. This combination yields robust, interpretable predictions and remains competitive even in the transformer era.

9.2.3 Transformer-Based NER: The BERT Paradigm

With the advent of pretrained transformer encoders such as BERT (Devlin et al., 2019), the NER task shifted toward contextualized representation learning. Unlike BiLSTMs, transformers rely on **self-attention** mechanisms that capture dependencies between all tokens simultaneously.

Contextual Encoding via BERT. Given an input sentence $x = (x_1, \ldots, x_T)$, BERT produces contextual embeddings:

$$H = [h_1, h_2, \ldots, h_T] = \text{TransformerEncoder}(x),$$

where each $h_t \in \mathbb{R}^d$ encodes bidirectional context across the entire sentence.

Token-Level Classification. A linear layer projects each contextual vector h_t into label logits:

$$z_t = W h_t + b,$$

and a softmax transforms these logits into probabilities:

$$p(y_t \mid x) = \text{softmax}(z_t).$$

The training objective is the token-level cross-entropy loss:

$$\mathcal{L} = -\sum_{t=1}^{T} \log p(y_t^{\text{true}} \mid x).$$

Extensions: BERT-CRF and Span-Based Models. Many modern NER systems extend BERT with a CRF layer or span-based classifiers:

- **BERT-CRF:** replaces independent softmax outputs with a CRF decoder, improving label consistency.
- **Span-based models:** predict entity spans directly by scoring pairs of start and end tokens, particularly effective for overlapping or nested entities.

Advantages of Transformer-Based NER.

- **Global attention:** captures long-range dependencies beyond local context windows.
- **Pretraining transfer:** benefits from massive self-supervised learning on large corpora.
- **Contextual polysemy handling:** the same word (e.g., *Apple*) receives different embeddings depending on context (company vs fruit).

9.2.4 Comparison and Conceptual Summary

Model	Key Idea	Mathematical Core
BiLSTM–CRF	Sequential encoding with structured output	$\mathbb{P}(y \mid x) \propto \exp(\text{Score}(x, y))$
BERT–Softmax	Transformer encoder + token-wise prediction	$p(y_t \mid x) = \text{softmax}(Wh_t + b)$
BERT–CRF	Global structure on top of BERT embeddings	Combines contextual attention with CRF sequence decoding, enabling end-to-end sequence labeling with global consistency.

9.2.5 Conclusion

Modern NER models exemplify the fusion of statistical structure and deep representation learning. They combine:

- **Geometric representations** of words (embeddings),
- **Contextual encoders** capturing dependencies across the sequence,
- **Structured decoders** enforcing label consistency.

In mathematical terms, NER can be viewed as the optimization of a conditional probability model over sequences:

$$\hat{y} = \arg\max_{y} \mathbb{P}_\theta(y \mid x),$$

where $\mathbb{P}_\theta$ is parameterized by deep neural functions (BiLSTM or Transformer). This framework unifies symbolic structure and neural geometry — making NER one of the most illustrative applications of deep learning in language understanding.

Bibliography

Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. A latent variable model approach to pmi-based word embeddings. *Transactions of the Association for Computational Linguistics*, 2016.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *International Conference on Learning Representations (ICLR)*, 2017. Published as a conference paper at ICLR 2017.

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. In *Computational Linguistics*, volume 22, pages 39–71. MIT Press, 1996.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

Léon Bottou. Stochastic gradient descent tricks. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2012. doi: 10.1007/978-3-642-35289-8_25. URL https://www.microsoft.com/en-us/research/wp-content/uploads/2012/01/tricks-2012.pdf.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394, 1999.

Noam Chomsky. *Syntactic Structures*. Mouton, The Hague, 1957.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989. doi: 10.1007/BF02551274.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method, 2014. URL https://arxiv.org/abs/1402.3722.

Geoffrey Hinton. Neural networks for machine learning, lecture 6a: Overview of mini-batch gradient descent, 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Coursera Lecture, University of Toronto.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2): 251–257, 1991. doi: 10.1016/0893-6080(91)90009-T.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. URL https://arxiv.org/abs/1412.6980.

Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu Jie Huang. A tutorial on energy-based learning. In Joost Bakker and Bernhard Schölkopf, editors, *Predicting Structured Data*. MIT Press, 2006.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2177–2185, Cambridge, MA, USA, 2014. MIT Press.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013. arXiv preprint arXiv:1301.3781.

Andriy Mnih and Geoffrey E. Hinton. A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 21, pages 1081–1088, 2008.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *Proceedings of EMNLP*, pages 1532–1543, 2014.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, pages 2227–2237, 2018.

Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3): 379–423, 1948.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of NeurIPS*, pages 5998–6008, 2017.

Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

Terry Winograd. *Understanding Natural Language*. Academic Press, New York, 1972.