UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

**INFORMATION ENGINEERING DEPARTMENT**

**COMPUTER ENGINEERING - AI & ROBOTICS**

# Natural Language Processing 2023/2024

Francesco Crisci 2076739

# Contents

# Chapter 1

# Introduction

## 1.1 General Introduction

**Natural Language Processing** is a field of artificial intelignece that allows machines to read, derive meaning ffrom text, and produce documents. It works in the background of many services, from chatbots through virtual assistants to social media tracking. Such language technologies are already showing major penetration into the information and communication industry.
NLP distinguishes itself from other AI application domains, as for instance computer vision or speech recognition. Text data is fundamentally discrete. But new words can alway be created. Few words are fery frequent, and there is a long tail of rare words. Out-of-Vocabulary words are always being discovered.
Languages have the following characteristics:

- it is **ambiguougs:** units can have different meanings;
- it is **compositional:** meaning of a unit defined as a function of the meaning of its components.
- it is **recursive:** units cna be repeatedly combined.
- they unveils **hidden structure:** local changes in a sentence might have global effects.

#### Ambiguity

word can belong to several categories, like noun, verbs or modal. The word bank, for example, have different meanings: river bank or money bank. The morphological composition, e.g., the word un-do-able is ambiguous between not doable and can be undone.

#### Compositionality

At each level, meaning of a larger unit is provided by some function of the meaning of its immediate components and the way they are combined.

#### Recursion

The rules of the grammar can iterate to generate an infinite number of structures, each with its specific meaning. Recursion is considered the main difference between human and other animals' languages.

#### Hidden structure

Local changees can disrupt te interprettion of a sentence. This sugests the existence of hidden structure.

## 1.2 Search & Learning

Many natural language processing problems can be written mathematically in the form of optimization:

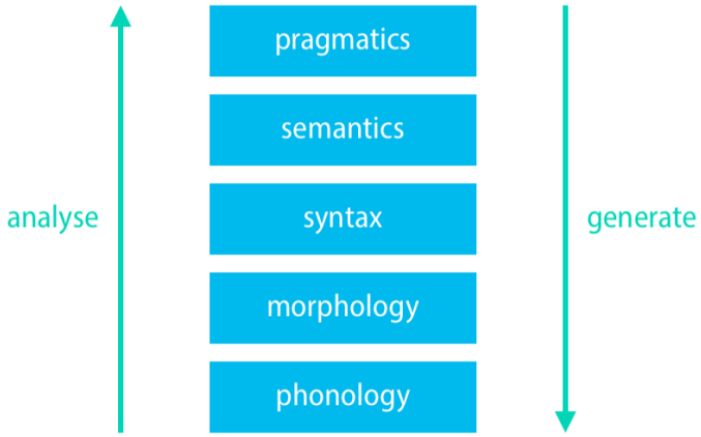
$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}(x)} \Psi(\mathbf{x}, \mathbf{y}; \boldsymbol{\Theta})$$

where x is the input, whicch is an element of a set $\mathcal{X}$; y is the output, which is an element of a set $\mathcal{Y}$(x). $\Psi$ is a scoring function, also called the **model**, which maps from the set $\mathcal{X} \times \mathcal{Y}$ to the real numbers; $\boldsymbol{\Theta}$ is a vector of **parameters** for $\Psi$. $\hat{y}$ is the predicted output, which is chosen to maximize the scoring function.
The **Search** module is responsible for finding the candidate output $\hat{y}$ with the highest score relative to the input x. the **Learning** module is responsible for finding the model parameters $\Theta$ that maximizes the predictive

performance.
**Structured prediction** is an umbrella term for supervised machine learning techniques that involves predicting structured objects, rather than scalar discrete or real values.

## 1.3 Essentials of linguistics

**Natural language** is a structured system for communication, consisting of a vocabulary and a grammar.
Linguistics is the scientific study of language, and in particular the relationship between language form and language meaning. Besides form and meaning, another important subject of study for linguistics is how languages is used in context. **Phonology** studies the rules that organize patterns of sunds in human languages.




It is different from **phonetics**, which is concerned with the production, transmission and perception of sounds, without prir knowledge of the language being spoken.
**Morphology** is the study of how words are composed by **morphemes**, which are the smallest meaningful units of language. The structure of a word consists of several morphemes: one root or stem and zero or more affixes. **Inflectional morhpology** means that there is no chang ein the grammatical category. **Derivational morphology** means that there is a change in the grammatical category.
A language can be of the following types:

- **Isolating:** a language in which each word form consists typically of a single morpheme;
- **Analytic:** no inflection to indicate grammatical relationship, may still contain derivational morphemes;
- **Synthetic**: uses inflection or agglutination to express relationships within a sentence.

The term **morphologically rich language** refers to a language in which substantial grammatical information is expressed at word level.
what we have seen in our examples so far is **concatenative morhpology:** morphemes are placed one after the other. Some languages, as for instance semitic languages, are based on **template morhpology.**
**Syntax** studies the rules and constraints that govern how words can be organized into sentences. Differently from formal language theory, NLP systems need to be robust to input that does not follow the rules of grammar.

### Part of speech

A **part of speech** is a ccategory for words that play similar roles within the syntactic structure of a sentence. PoS can be defined:

- **distributionally:** Kim saw the {elephant, movie, mountain, error } before we did.
- **functionally:** verbs=predicates; nouns=arguments; adverbs=modify verbs, etc.

**Open class** tags: noun, verb, adjective, and adverb. New words in the language are usually added to these classes.
**Closed class** tags: determines, prepositions, conjunctions, etc. Closed word classes rarely receive new members.
There are several representations for syntactic structure. Most common are Phrase structure and Dependency tree.

**Phrase structure** is a tree-like representation with leaf nodes representing sentence words and internal nodes representing word grouping called **phrases.**
**Dependency tree** is a tree-like representation where the nodes represent words and punctuation in the sentence, and arc represent grammatical relations between a **head** and a **dependent.** Dependency trees use labels at arcs, representing grammatical relations. Arcs in a dependency tree are often called **dependencies**.
**Semantics** is the study of the meaning of linguistic expressions such as words, phrases and sentences. The focus is on what expressions conventionally/abstractly mean, rather than on what they migght mean in a particular context. The linguistic study of word meaning is called **lexical semantics.** The internal semantic structure of a word refers to the similarity with other words. The external semantic structure of a word refers to the allowability to combine with other words.
**Lexical ambiguity** arises because a word can have different meanings, called word senses.
**Principle of compositionality:** the meaning of a whole expression is a function of the meanings of its parts and of the way they are syntactically combined. Syntax provides the scaffolding for semantic composition: the meaning of a sentence isn't just the amalgamation of the meaning of its component words.
**Meaning representation** Many representations for semantic structure. Most common are logical, prediccate-argument or graph structure.
**Pragmatics** studies the way linguistic expressions with their semantic meanings are used for specific communicative goals. In contrast to semantics, pragmatics explicitly asks the question what an expression means in a given context. An important concept in pragmatics is the speech art, which describes an action performed through language.

#### Discourse analysis

**Discourse analysis** studies written and spoken language relation to its social context. **Discourse** refers to a piece of text with multiple sub-topics and ccoherence relations between them, such as explanation, elaboration andd contrast. There are many formalisms for representing discourse structure, as for example disccourse representation theory and rhetorical structure theory. **Dialogue** is a cooperative kind of discourse, where two or more participants are involved.

## 1.4 Text Language Processing

Cleaning up a text ddata set requires the use of specialized **regular expressions.** Most programmingg languguages have facilities for compiling regular expressions into efficient finite automata and running these automata on input text. Regular expressions come in many variants. We describe here the so-called **extended** regular expressions. We have different categories and operators:

- **Search:** $/d/$ matches d in woodchuck.
- **Sets of characters:** $/[abc]/$ matches a, b or c; $/[\hat{}abc]/$ matches any character other than a, b and c; $/[a-z]/$ matches any character from a to z; $/./$ matches any character; also called **wildcard**.
- **Aliases:** $\backslash d$ stands for any digit; same as $[0-9]$; $\backslash D$ stands for any character other than a digit, same as $[\hat{}0-9]$; $\backslash w$ stands for any alphanumeric or underscore; $\backslash W$ is the converse of $\backslash w$; $\backslash s$ stands for any whitespace character; $\backslash S$ converse of $\backslash s$; $\backslash.$ stands for period; $\backslash n$ stands for newline.
- **Repetition:** $/\ d\{2,5\}/$ matches between 2 and 5 digits; $/\backslash d\{3,\}/$ matches 3 or more digits; $/\backslash d\{4\}\backslash w?/$ matches exactly 4 digits and one optional alphanumeric or underscore; $/[a-z]+/$ matches one or more lowercase letters; $/\backslash s+java\backslash s+/$ matches java with one or more whitespace characters before and after; $/[\hat{}\backslash+]*/$ matches zero or more characters other than + (Kleen star).
- **Anchors:** $/\hat{}/$ matches beginning of input string or line; $/\$/$ matches end of input string or line; $/\backslash b/$ matches word bondary; $/\hat{}The$ matches occurence of The at the beginning of a string. With | you can have dijunctions and groups $/the|any$ matches the or any; $/grupp(y|ies)/$ matches gruppy and gruppies.
- **Substitute and Back-reference:** we can replace matches of a regular expression by a given pattern using the substitute operator $s/word1/word2$; we can use matches of a regular expression through the back-reference operator.

### Words

We need to disstinguish between words and punctuation marks. Punctuation is critical to find sentence boundaries and for identifying some aspects off meaning. There are two ways of talking about words: **types** are the distinct words appearing in a document and **tokens** that are the individual occurences of words in a document. the set of all types in a corpus is the **Vocabulary** V. The vocabulary size $|V|$ is the number of types in the

corpus. The size of the corpus N is the number of tookens, if we ignore punctuation marks. In very large corpora, the relation between N and $|V|$ can be expressed as

$$|V| = kN^{\beta}$$

In very large corpora, the r-th most frequent type has frequency f(r) that scales acccording to

$$f(r) \propto \frac{1}{(r+\beta)^{\alpha}}$$

The **word-form** is the full infleccted or derived form of the word. Eacch word-form is associated with a single **lemma**, the citation form used in dictionaries. alternatively to $|V|$, another measure of the number of words in a corpus is the number of lemmas.

## Corpora

**Corpus:** large collection of text, in computer-readable form. Several dimensions of varition for corpora should be taken into account: language, genre, etc.
**Languag:** it is important to tet NLP algorithms on more than one language. languages lacking large corpora are considered low-resource languages.
**Genre:** text documents might come from newswire, fiction, scientific articles, Wikipedia, etc.
**Time:** language changes over time. for some languages we have good corpora of texts from different historical periods.
**Collection process:** how big is the data and how was it sampled? How was the data pre-processed, and what metadata is available?
**Annotation:** what are the specifics of the used annotation? How was the data nnotated? How where the annotators trained?

## Text Normalization

**Text normalization** is the process of transforming a text into some predefined standard form. This consists of seveeral tasks. There is no all-purpose normalization proccedure: text normalization depends on what type of text is being normalized and what type of NLP task needs to be carried out afterwards. Text normalization is also important for applications other than NLP, such as text mining and WEB search engines.

### Language identification

**Language identification** is the task of detecting the source language for the input text. Several statistical techniques for this task: functional word frequency, N-gram language models, distance measure based on mutual information, etc.

### Spell checker

**Spell checkers** correct grammatical mistakes in text. They use approximate string matching algorithms such as **Levenshtein distance** to find correct spellings.

### Contractions

The following should be managed before further normalization: contracted forms, abbreviations and slangs. The most straightforward techniques is to create a dictionary of contractions and abbreviations with their corresponding expansions.

### Punctuation

**Punctuation** marks in text need to be isolated and treated as if they were separatee words. This is critical for finding sentence boundaries and for identifying some aspects of meaning.

### Tokenization

**Tokenization** is the process of segmenting text into units called tokens. Tokenization tecchniques can be grouped into three families: word, character and subword tokenization. Tokens are then organized into a vocabulary and, depending on the specific NLP application, may later be mapped into natural numbers.

## Word tokenization

Word tokenization is a very common approach for European languages. For english, most of the text is already tokenized after previous steps, with the following important exceptions: special compound names and city names.

## Character tokenization

Major east Asian languages write text without any spaces between words. For most Chinese NLP task, character tokenization works better than word tokenization: each character generally represents a single unit of meaning and word tokenization results in huge vocabulary, with large number of rare words.

## Subword tokenization

Many NLP systems need to deal with **unknown words**, that is, words that are not in the vocabulary of the system. To deal with the problem of unknown words, modern tokenizers automatically induce sets of tokens that include tokens smaller than words, called **subwords**. Subword tokenization reduces vocabulary size, and has become the most common tokenization method for large language modelling and neural models in general. Subword tokenization is inspired by algorithms originally developed in information theory as a simple and fast form of data compression alternative to Lempel-Ziv-Welch.
Subword tokenization schemes consist off three diffferent algorithms:

- the **token learner** takes a raw training corpuss and induces a set of tokens, called **vocabulary**.
- the **token segmenter** takes a vvocabulary and a raw sentence, and segments the sentence into the tokens in hte vocabulary.
- the **token merger** takes a token sequence and reconstructs the original sentence.

Three algorithms are widely used for subword tokenization: byte-pair encoding, unigram and WordPiece tokenization.

### BPE: learner

The BPE **token learner** is usually run inside words, not merging acoss word boundaries. To this end, use a special end-of-word marker. The algorithm iterates through the following steps:

- begin with a vocabulary composed by all individual characters.
- choose the two symbols A, B that are most frequently adjacent.
- add a new merged symbol AB to the vocabulary.
- replace every adjacent A, B in the corpus with AB.

Stop when the voocabulary reaches size k, a hyperparameter.

### BPE: Encoder

Two versions of **BPE token segmenter**: apply merge rules in frequency order all over the data set, or, for each word, left-to-right, match longest token from vocabulary. Encoding is computationally expensive. Many systems use some form of caching: pre-tokenize all the words and save how a word should be tokenized in a dictionary; when an unknown word is seen: applu the encoder to tokenize the word and add the tokenization to the dictionary for future reference.

### BPE: Decoder

**BPE token merger:** to decode, we have to concatenate all the tokens together to get the whole word and use the end-of-word marker to solve possible ambiguities.

## WordPiece

**WordPiece** is a subword tokenization algorithm used by the large language model BERT. Like BPe, WordPiecce starts from the initial alphabet and learns merge rules. The main difference is the way pair A, B is selected to be merged:

$$\frac{f(A,B)}{f(A) \times f(B)}$$

The algorithm prioritizes the merging of pairs where the individual parts are less frequent in the vocabulary.
Text normalization also includes **sentence segmentation:** breaking up a text into individual sentences. This can be undone using cues like periods, question marks, or exclamation points.
**Lower casing** is very useful to standardize words and before stop word removal. Most programming languages have facilities for string lowercasing.
**Stop word removal** includes getting rid of common articles, pronouns, prepositions and coordinations. Stop word removal heavily depends on the task at hand, since it can wipe out relevant information.

**Stemming** refers to the process of slicing a word with the intention of removving affixes. Stemming is problematic in th elinguistic perspective, since it sometimes produces words that are not in the langage, or else words that have a different meaning.
**Lemmatization** has the objeccctive of reduccing a word to its base form, also called **lemma**, therefore grouping together different forms of the same word. Lemmatization and Stemming are mutually exclusive, and the former is much more resource-intensice than the latter.

# Chapter 2

# Word Embeddings

What is the meaning of a word? The linguistic study of word meaning is called **lexical semantics**. A model of word meaning should allow us to relate different words and draw inferences to address meaning-related tasks. A word-form can have multiple meanings; each meaning is called a **word sense**, or sometimes a **synet**. **Word sense disambiguation** (WSD) is the task of determining which sense of a word is being used in a particular context.
Lexical semantic relationship between words are important components of word meaning. Two words are **synonyms** if they have a common word sense. Two words are **similar** if they have similar meanings. Two words are **related** if they refer to related concepts. Two word are **antonyms** if they define a binary opposition. one word is an **hyponim** of another if the firs has a more specific sense. notions of **hypernym** or **hyperonym** are define symmetrically. Words can have **affective** meanings implying positive or negative connotations/evaluation.

## 2.1 Distributional semantics

**Distributional semantics** develops methods for quantifying semantic similarities between words based on their distributional properties, meaning their **neighboring words**. The basic idea lays in the so-called **distributional hypothesis**: linguistic items with similar distribution have similar meanings. The basic approach is to collect distributional information in high-dimensional vectors, and to define distributional/semantic similarity in terms of vector similarity. Similar words are mapped into "close enough" vectors.
Lexical semantic relationship, represented as word vector differences, aare generally preserved.


Figure 2.1: Example: Vectors from sentiment analysis application, projected into 2-dimentional space.

The obtained vectors are called **word embeddings**. The approach is called **vector semantics** and is the

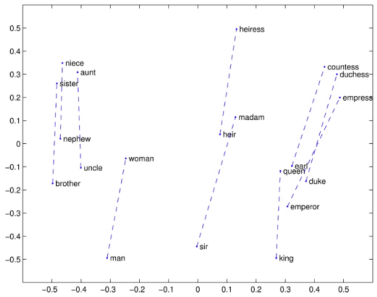



Figure 2.2: Example: Vectors projected into 2-dimentional space.

standard way to represent word meaning in NLP. All the learning algorithms we report are **unsupervised**. We discuss two families of word embeddings:
**Sparse vectors:** Vector components are computed through some function of the counts of nearby words;
**Dense vectors:** Vector components are computed through some optimization or approximation process.

There is a review of vectors and vector's opertions which i'm gonna omit. You are at the fifth year of engineering. If you need a review, i'm sorry, but you dumb :)

## 2.2 Term-context matrix

Let $\{w_1, ..., w_W\}$ be a set of **term** words and let $\{c_1, ..., c_C\}$ be a set of **context** words. For each word $w_i$ in the corpus, we consider context words appearing inside a window of size L, at the left or at the right of $w_i$. A **term-context matrix** F is a matrix $W \times C$, where each element $f_{i,j}$ is the number of times context word $c_j$ appears in the context of term word $w_i$ in the corpus.

## 2.3 Probabilities & information theory

**Pointwise mutual information** is a measure of how often two events x and y occur, compared with what we would expect if they were independent:

$$I(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

It is simmetric, it can take positive or negative values. If the randome varaibles are independent then it is zero. The PMI between $w_i$ and $c_j$ is

$$PMI(w_i, c_j) = \log_2 \frac{P(w_i, c_j)}{P(w_i)P(c_j)}$$

The numerator tells us how often we observed the two words together. The denominator tells us how often we would expect the two words to co-occur assuming they each occurred independently. The ratio gives us an estimate of how much more the tw words co-occur than we expected by chance. We can rewrite the PMI as:

$$PMI(w_i, c_j) = \log_2 \frac{P(w_i|c_j)}{P(w_i)}$$

PMI may then be understoodd as quantifying how much our confidence in the outcome $w_i$ increases after we observe $c_j$.

## 2.4 Probability estimation

How do we estimate the Probabilities $P(w_i, c_j), P(w_i), P(c_j)$? We can estimate:

$$P(w_i, c_j) = \frac{f_{i,j}}{\sum_{i=1}^{W} \sum_{j=1}^{C} f_{i,j}}$$

Note that the denominator is the total number of possible target/context pairs.

$$P(w_i) = \sum_{j=1}^{C} P(w_i, c_j) = \frac{\sum_{j=1}^{C} f_{i,j}}{\sum_{i=1}^{W} \sum_{j=1}^{C} f_{i,j}}$$

Similarly we can estimate:

$$P(c_j i) = \sum_{i=1}^{W} P(w_i, c_j) = \frac{\sum_{i=1}^{W} f_{i,j}}{\sum_{i=1}^{W} \sum_{j=1}^{C} f_{i,j}}$$

It is difficult to estimate negative values of PMI. We can use **Positive PMI**:

$$PPMI(w_t, w_c) = \max\{\log_2 \frac{P(w_t, w_c)}{P(w_t)P(w_c)}, 0\}$$

## 2.5 Practical issues

PMI has the problem of being **biased** towards infrequent events: very rare words ttend to have very high PMI values. One way to reuce this bias is to slightly change the computation for $P(w_c)$ in the PPMI:

$$PPMI_\alpha(w_t, w_c) = \max\{\log_2 \frac{P(w_t, w_c)}{P(w_t)P_\alpha(w_c)}, 0\}$$

$$P_\alpha(w_c) = \frac{C(w_c)^\alpha}{\sum_{v \in V} (C(v))^\alpha}$$

We can use the rows of the PPMI term-context matrix as word embeddings. Notice that these vectors have the size of the vocabulary, which can be quite large, and when viewed as arrays, they are very sparse. Dot product between these word embeddings needs to use specialised software libraries, implementing vectors as dictionaries rather than arrays.
We can obtain dense word embeddigns from th ePPMI term-context matrix. This is done through a matrix decomposition technique known as **truncated singular value decomposition**.

### 2.5.1 Truncated singular value decomposition

For a matrix $A \in \mathbb{R}^{n \times m}$, the **Frobenius norm** is

$$||A||_F = \sum_{i=1}^{n} \sum_{j=1}^{m} a_{i,j}^2$$

where $a_{i,j}$ are the elements of the matrix. Let $|V|$ be the vocabulary size, and let $K << |V|$ be the size of the desired embedding. Let $P \in \mathbb{R}^{|V| \times |V|}$ ne the PPMI term-context matrix. Let $U, V \in \mathbb{R}^{|V| \times K}$ be learnable parameters. The basic idea is to approximate P by means of a matrix $\tilde{P}(U, V) \in \mathbb{R}^{|V| \times |V|}$ such that

$$\min_{U,V} ||P - \tilde{P}(U, V)||_F$$

**Truncated singular value decomposition** solves the following constrained optimization problem:

$$\min_{U \in \mathbb{R}^{|V| \times k}, S \in \mathbb{R}^{K \times K}, V \in \mathbb{R}^{|V| \times K}} ||P - USV^T||_F$$

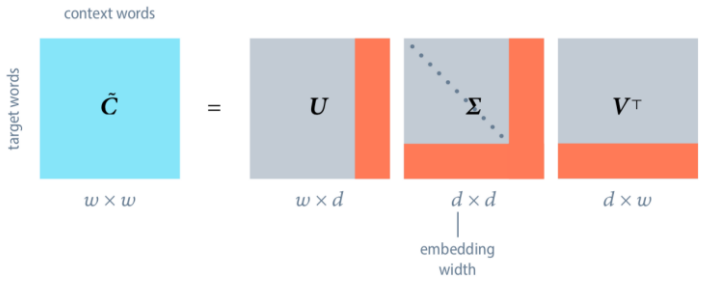
subject to conditions $U^T U = \mathbb{I}, V^T V = \mathbb{I}$ and S is diagonal with its elements in decreasing order. U are the target embeddings, $SV^T$ are the context embeddings.
Let $U_k, V_k \in |\mathbb{V}| \times \nVdash$ be the k-th columns of matrices U and V, respectively. In truncated SVD, P is approximated by the sum of K rank-1 matrices:

$$P \approx \sum_{k=1}^{K} s_k U_k V_k^T$$

Equivalently, each element $p_{i,j}$ of P is approximated through the dot product between the embedding of the i-th target word and the embedding of the j-th context word, weighted by the singular values

$$p_{i,j} \approx \sum_{k=1}^{K} s_k u_{i,k} v_{j,k}$$

We can alternatively view the previous approximation approacch as a factorization problem for matrix P, $P = USV^T$, where $U, S, V \in |\mathbb{V}| \times |\mathbb{V}|, U^T U = \mathbb{I}, V^T V = \mathbb{I}$ and S is a diagonal with **singular values** $s_1 > s_2 > ... > s_V$ in its diagonal. We then choose $K << |V|$ on the basis of mass distribution at the singular values, and truncate matrices U,S,V accordingly.




## 2.6 Neural word embeddings

Word embeddings can also be computed using neural networks. Generally speaking, neural word embeddings have the following properties: small number of dimension, vectors are dense, the dimensions don't have a clear interpretation, components can be negative. Neural word embeddings work better than previous embeddings in every NLP task.
**Word2vec** is a software package including two different algorithms for learning word embeddings: skip-gram with negative sampling (SGNS) and continuous bag-of-words (CBOW). The idea is that instead of counting how oftne a context word appears near a target word, we train a classifier on the following binary prediction task: *Is a giiven context word likely to appear near a given target word?* We don't really care about this prediction task: intead, we use the learned parameters as the word embeddings.

### 2.6.1 Skip-Gram

More specifically, the basic steps of the **skip-gram** algorithm are as follows:
For each target word $w_t \in V$ :

I treat $w_t$ andd any neighboring context wor $w_c$ as positive examples;

II randomly sample other words $w_n \in V$, called **noise words**, to produce negative examples for $w_t$

Using **logistic regression** tto train a classfier to distinguish positive and negative examples. Use the learned weights as the embeddings.

### 2.6.2 Logistic Regression

For any pair of words $w_t, u \in V$, we need to define the probability that u is/is not a context word for target word $w_t$

$$P(+|w_t, u) \qquad P(-|w_t, u) = 1 - P(+|w_t, u)$$

For each word $w \in V$, we construct two complementary embeddings: a target embedding $e_t(w) \in \mathbb{R}^d$ and a context embedding $e_c(w) \in \mathbb{R}^d$. Integer d is the size of the embedding. Operator $e_t(w)$ is always assigned to target words, operator $e_c(w)$ is always assigned to context or noise words. We use the **dot product**:

$$e_t(w) \cdot e_c(w) = |e_t(w)||e_c(w)| \cos\theta$$

where $\theta$ is the angle between $e_t(w)$ and $e_c(w)$. We also use the **logistic sigmoid function** $\sigma$, which maps real values to probabilities. We can now define:

$$P(+|w, u) = \sigma(e_t(w) \cdot e_c(w)) = \frac{1}{1 + \exp(-e_t(w) \cdot e_c(w))}$$

We then have vectors $e_t(w), e_c(w)$ with small angle: positive cosine similarity, $P(+|w, u)$ close to one, and vectors $e_t(w), e_c(w)$ with large angle: negative cosine similarity, $P(+|w, u)$ close to zero.

## 2.7 Training

For simplicity, let us consider a dataset with only one target/context pair (w,u), along with k noise words $v_1, ..., v_k$(negative examples). Skip-gram makes the simplifying assumption that all context words are independent. Then the log-likelihood of the data is:

$$LL_w = \log\left(P(+|w, u)\prod_{i=1}^{k} P(-|w, v_k)\right) = \log P(+|w, u) + \sum_{i=1}^{k} \log P(-|w, v_k)$$

Which we can rewrite as:

$$LL_w = \log\sigma(e_t(w) \cdot e_c(u)) + \sum_{i=1}^{k} \log(1 - \sigma(e_t(w) \cdot e_c(v_k)))$$

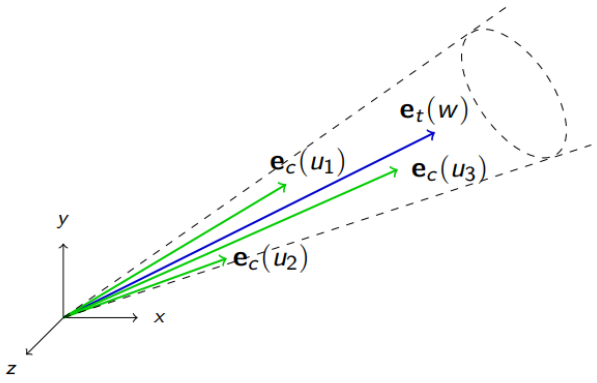
We can alternatively minimize the inverse of $LL_w$, which in ccaase of the logistic regression is the **cross-entropy** loss function:

$$L_{CE} = -\log\sigma(e_t(w) \cdot e_c(u)) - \sum_{i=1}^{k} \log(1 - \sigma(e_t(w) \cdot e_c(v_k)))$$

The model parameters are the two $\mathbb{R}^{|V|\times d}$ matrices with the target and context embeddings $e_t(w), e_c(w), w \in V$, which are randomly initialized. By making an update to minimize $L_{CE}$ we force an increase in similarity between $e_t(w)$ and $e_c(w)$, and we force a decrease in similarity between $e_t(w)$ and $e_c(v_k)$, for all of the noise words $v_k$. We train the model with **stochastic gradient descent**, as usual for logistic regression. in the end, we retain the target embeddings $e_t(w)$ and ignore the context embeddings $e_c(w)$.

### 2.7.1 Geometric interpretation

Target vector $e_t(w)$ is surrounded by its context vectors $e_c(u_j)$ which form a thin hypercone:
Let w be a target word with context words $u_j's$. Let also w' be a target word similar to w. Then we have that $e_c(u_j)$ form a thin cone surrounding vector $e_t(w)$; vector $e_t(w')$ must be placed within that cone, because w' and w share their context word; $e_t(w)$ and $e_t(w')$ are force to be at a small angle. As a result, similar words cluster together. Within a similar reasoning, we can argue that different words repel each other.




### 2.7.2 Practical isseus

Clean up text: convert any punctuation into token $< PERIOD >$. Remove all words w such that $f(w) \leq D$: greatly redduces issues due to noise into data and improves the quality of the vector representation:
Indexing: convert words in V to non-negative integers and back again, this makes your code simpler.
Words that shouw up very often don't provide much context and are often regardes as noise. The process of discarding occurences of these qords by mean of some probability distribution is called subsampling. We discard occurence $w_i$ with probability given by

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

where t is a threshold parameter depending on dataset size.
The rato k between the positive and the negative examples is a hyperparameter and must be adjusted on the development set. The noise words are chosen according to the relative frequency estimator and a $\alpha$ corrector:

$$P_\alpha(v) = \frac{C(v)^\alpha}{\sum_{w \in V}(C(w))^\alpha}$$

Once hyperparameter L is fixed, context window size can be chosen randomly in the range $[1, ..., L]$, with uniform distribution. this results in linear decay for context words, depending on distances from target word: words at distance 1 from target will be chosen with P=1, words at distance L from target will be chosen with $P0\frac{1}{L}$.
Should we estimate one embedding per word form or else estimate distinct embeddings for each word sense? Intuitively, if word representations are to capture the meaning of individual words, then words with multiple meanings should have multiple embeddings.

## 2.8 Miscellanea

### 2.8.1 FastText

FastText is an embedding that deals with unknown words and sparsity in languages with rich morphology, by using subword models. Each word in FastText is represented by itself plus a bag of character N-grams, for bounded values of N.

### 2.8.2 GloVe

GloVe (Global Vectors), is an embedding that accounts for global corpus statistics, which was somehow disregarded by word2vec. GloVe combines the intuitions of count-based models like PPMI, while also capturing the linear structures used by methods like word2vec.

### 2.8.3 Semantic properties

Both sparse and dense vectors are sensible to the context window size used to collect counts, controlled by hyperparameter L. The choice of L depends on the goals of the representation:

- small values of L provides semantically similar words with the same parts of speech.
- large values of L provides words that are tpically related but not similar.

another ssemantic property of embeddings is their ability to capture relational meanings. The parallelogram model can be used to solve analogy problems.

## 2.9 Evaluation

**Extrinsic evaluation**: use the model to be evaluated in some end-to-end application and measure performance.
**Intrinsic evaluation**: look at performance of model in isolation, w.r.t a given evaluation measure.
The most common evaluation metric for embedding models is extrinsic evaluation on end-to-end tasks: machine translation, sentiment analysis, etc. Three types of intrinsic evaluation:

- word **similarity**: compute a numerical score for the semantic similarity between two words;
- word **relatedness**: compute the degree of how much one word has to do with another words;
- word **analogy**: comparison of two things to show their similarities.

## 2.10 Cross-lingual word embedding

**Cross.lingual word embeddings** encode words from two or more languages in a shared high dimensional space. Vectors representing words with similar meaning are closely located, regardless of language.
Cross-lingual word embeddings are very useful in comparing the meaning of word across languages; are key to applications such as machine translation, multilingual lexicon induction, andd cross-lingual information retrieval; enable model transfer between resourse-rrich and low-resource languages, by rpoviding a common representation space.
The class of **objective functions** minimized by most cross-lingual word embedding methods can be formulated as:

$$J = \mathcal{L}^1 + ... + \mathcal{L}^\ell + \Omega$$

where $\mathcal{L}^i$ is the monolingual loss of the i-th language and $\Omega$ is a regularization term. Joint optimization of multiple non-convex losses is difficult: most approaches optimize one loss at a time, while keeping certain variables ffixed; this step-wise approach is approximate and does not guaranteee to reach even a local optimum.
The choice of multilingual parallel data sources is more important for the model performance than the actual underlying architecture. Method differ along the following two dimensions: type of text **alignment**: at the level of words, sentences, or documents, which introduce stronger or weaker supervision; Type of text **comparability**: extract translations, weak translations, or similar meaning.
**Variation in ambiguity**: ambiguity for word w in one language does not transfer to the translation of w in anoter language.

# Chapter 3

# Language Models

## 3.1 Language Modeling

**Language modeling** is the task of predicting which word comes next in a sequence of words. More formally, given a sequence of words $w_1 w_2 ... w_t$ we wwant to know the probability of the next word $w_{t+1}$:

$$P(w_{t+1}|w_1 w_2 ... w_t)$$

When the string is understood from the context, we write $w_{1:t}$.
Rather than as **predictive** models, language models can also be viewed as **generative** models that asign probability to a piece of text:

$$P(w_1 ... w_t)$$

These two view are **equivalent**, as the probability of a sequence can be expressed as a product of conditional probabilities:

$$P(w_{1:n}) = \prod_{t=1}^{n} P(w_t|w_{1:t-1})$$

Conversely, a conditional probability can be expressed as a ratio of two sequence probabilities:

$$P(w_{t+1}|w_{1:t}) = \frac{P(w_{1:t+1})}{P(w_{1:t})}$$

### Probability Estimation

Assume the text *its water is so transparent that*. We want to know the probability that the next word is *the*. Let C(its water is so transparent that) be the number of times the string is seen in a large corpus. One way to estimate the above probability is to set

$$P(the|\text{its water is so transparent that}) = \frac{C(\text{its water is so transparent that the})}{C(\text{its water is so transparent that})}$$

In general, given a large corpus, we can set:

$$P(w_t|w_{1:t-1}) = \frac{C(w_{1:t})}{C(w_{1:t-1})}$$

The quantities $C(w_{1:t})$ are called **frequencies**, the ratio $\frac{C(w_{1:t})}{C(w_{1:t-1})}$ is called **relative frequency**. The estimator above is therefore called **relative frequency estimator**. In practive, consider an aggressive upper bound of N=20 words in our sequence, and an English vocabulary V of size $|V| = 10^5$. Then the number of possible sequencces is $|V|^{20} = 10^{100}$. This estimator is extremely data-hungry, and suffers from **high varaince**: depending on what data happens to be in the corpus, we could get very different probability estimations.

## 3.2 N-gram Model

A string $w_{t-N+1:t}$ of N words is called **N-gram**. The **N-gram model** approximates the probability of a word given the entire sentence history by conditioning only on the past N-1 words. The 2-gram model, for example, makes the approximation

$$P(w_t|w_{1:t-1}) \approx P(w_t|w_{t-1})$$

The general equation for the N-gram model is

$$P(w_t|w_{1:t-1}) \approx P(w_t|w_{t-N+1:t-1})$$

The relative frequency estimator for the N-gram model is Then

$$P(w_t|w_{t-N+1:t-1}) = \frac{C(w_{t-N+1:t})}{C(w_{t-N+1:t-1})}$$

For N=2 we have:

$$P(w_t|w_{t-1}) = \frac{C(w_{t-1}w_t)}{\sum_u C(w_{t-1}u)} = \frac{C(w_{t-1}w_t}{C(w_{t-1})}$$

For N=1 we have $P(w_t) = \frac{C(w_t)}{n}$ where n is the length of the training set. The model requires estimating and storing the probability of only $|V|^n$ events, which is exponential in N, not in the lenght of the sentence.

### 3.2.1 Learning

N is a hyperparameter. When setting its value, we face the **bias-variance tradeoff**. When No is to small, it fails to recover long-distance word relations. When N is too large, we get data sparsity.
The relative frequency estimator can be mathematiccally derived by maximizing the **likelihood** of the dataset. This can be done by solving a **constrained optimization problem**, using Lagrange multipliers. Therefore, the relative frequency estimator is also called the **maximum likelihood estimator** (MLE).

### 3.2.2 Practical issues

To compute 3-gram probabilities for words at the start of a sentence, we use two markers. Similarly for higher order N-grams. Multiplying many small probabilities results in **underflow**. it is mush safer and more efficient to use **negative log probabilities**, -log(p), and add them.

### 3.2.3 Evaluation

**Extrinsic evaluation:** use the model in some application and measure performance on that application.
**Intrinsic evaluation:** look at performance of model in isolation, with respect to a given evaluation measure.

#### Perplexity

Intrinsic evaluation of language models is based on the inverse probability of the test set, normalized by the number of words. For a test set $W = w_1w_2...w_n$ we define **perplexity** as:

$$PP(W) = P(w_{1:n})^{-\frac{1}{n}} = \sqrt[n]{\prod_{j=1}^{n} \frac{1}{P(w_j|w_{1:j-1})}}$$

The multiplicative inverse probability $1/P(w_j|w_{1:j-1})$ can be seen as a measure of how surprising the next word is. The degree of the root averages overall words of the test set, providing **average surprise per word**. For large enough test data obtained in a uniform way, perplexity is mor or less constant, i.e. independent of n. The lower the perplexity, the better the model.

#### Sparse data

If there isn't enough data in the training set, counts will be zero for some grammatical sequences. Then some of the N-gram probabilities will be **zero** or **undefined**.

## 3.3 Smoothing

**Smoothing** techniques (also called **discounting**) deal with words that are in our vocabularry V but were never seen before in the given context. Smoothing prevents LM from assigning **zero probability** to these events. The idea is to shave off a bit of probability mass from more frequent events and givve it to the events we have never seen in the training set.

### 3.3.1 Laplace Smoothing

**Laplace Smoothing** does not perform well enough, but provides a useful baseline. The idea is to pretend that everything occurs once more than the actual count. In order to apply smoothing to our N-gram model, let us rewrite the relative frequency estimator in a more convenient form:

$$P(w_t|w_{1:t-1}) = \frac{C(w_{1:t})}{\sum_u C(w_{1:t-1}u)}$$

In the summation u is a single word ranging over the entire vocabulary V andd the sentence end marker.

#### Laplace for 1-grams

Let n be the number of tokens, that iss, the lenght of the training set, and call that $|V|$ is the number of word types. The **adjusted estimate** of the probability of word $w_t \in V$ is then:

$$P_L(w_t) = \frac{C(w_t) + 1}{n + |V|}$$

The extra $|V|$ comes from pretending there are $|V|$ more observations, one for each word type.
Alternatively, we can think of $P_L$ as applying an **adjusted count** $C^*$ to the n actual observations

$$C^*(w_t) = (C(w_t) + 1)\frac{n}{n + |V|}$$

$$P_L(w_t) = \frac{C^*(w_t)}{n} = \frac{C(w_t) + 1}{n + |V|}$$

Under this view, the smoothing algorithm amouts to **discounting** (lowering) count for high frequency words and redistributing

$$\sum_{u \in V} C(u) = \sum_{u \in V} C^*(u) = n$$

We can consider the **relative discount** $d(w_t)$, defined as the ratio of the discounted counts to the original counts:

$$d(w_t) = \frac{C^*(w_t)}{C(w_t)} = \left(1 + \frac{1}{C(w_t)}\frac{n}{n + |V|}\right)$$

By solving $d(w_t) < 1$, we find that discounting happens for high frequency types u succh that $C(u) > \frac{n}{|V|}$.

#### Laplacce for 2-grams

The 2-gram model relative frequency estimator is:

$$P(w_t|w_{t-1}) = \frac{C(w_{t-1}w_t)}{\sum_u C(w_{t-1}u)} = \frac{C(w_{t-1}w_t)}{C(w_{t-1})}$$

The adjusted estimate of the probability of 2-grm $w_{t-1}w_t$ is then:

$$P_L(w_t|w_{t-1}) = \frac{C(w_{t-1}w_t) + 1}{\sum_u [C(w_{t-1}u) + 1]} = \frac{C(w_{t-1}w_t) + 1}{C(w_{t-1}) + |V|}$$

The adjusted count is:

$$C^*(w_t|w_{t-1}) = \frac{[C(w_{t-1}w_t) + 1]C(w_{t-1})}{C(w_{t-1}) + |V|}$$

$P_L(w_t|w_{t-1})$ is larger than $P(w_t|w_{t-1})$ for 2-gramm sequences that occur zero or few times in the training set. However, $P_L(w_t|w_{t-1})$ will be much lower ffor 2-graam sequences that occur often. So Laplace smoothing is **too crude** in practice.

#### Add-k smoothing

**Add-k smoothing** is a generalization of add-one smoothing. For some $0 \leq k < 1$:

$$P_{Add-k}(w_t|w_{t-1}) = \frac{C(w_{t-1}w_t) + k}{C(w_{t-1}) + k|V|}$$

**Jeffreys-Perks law** corresponds to the case k=0.5, which works well in practice and benefits from some theoretical justification.

**Smoothing and Perplexity**

When smooting a language model, we are redistributing proobabiility mass to outcomes we have never observed. This leaves a smaller fraction of the probaibility mass to the outcomes that we actually did observe during training. Thus, the more probability we are taking away from observed outcomes, the **higher** the perplexity on the training data.

## 3.4 Backoff and Interpolation

Backoff and interpolation techniques deal with words that are in our vocabulary, but in the test set combine to form previously unseen contexts. These techniques prevent LM from creating **undefined probabilities** for these events.

### 3.4.1 Backoff

**Backoff** combines fine grained models with coarse graained models. The idea is simple: if you have a trigram use those; if not use bigrams; if you don't have neither, use unigrams.
**Katz backoff** is a popular but rather complex algorithm for backoff.

### 3.4.2 Stupid backoff

With very large text collections a rough approximation of Katz backoff is often sufficient, called **stupid backoff**. For some small $\lambda$:

$$P_S(w_t|w_{t-N+1:t-1}) = \begin{cases} P(w_t|w_{t-N+1:t-1}) = \frac{C(w_{t-N+1:t-1})}{C(w_{t-N+1:t-1})} \text{ if } C(w_{t-N+1:t}) > 0 \\ \lambda P_S(w_t|w_{r-N+2:t-1}) \text{ otherwise} \end{cases}$$

### 3.4.3 Linear interpolation

In **simple linear interpolation**, we can combine different order N-grams by linearly interpolating all the models. Simple linear interpolation for N=3:

$$P_L(w_t|w_{t-2}w_{t-1}) = \lambda_1 P(w_t|w_{t-2}w_{t-1}) + \lambda_2 P(w_t|w_{t-1}) + \lambda_3 P(w_t)$$

for some choices of possitive $\lambda_1, \lambda_2, \lambda_3$ such that $\sum_j \lambda_j = 1$.
What are good choices for the $\lambda_j's$? Algorithms exist that attempt to optimise likelihood of training data. We might have different values for the $\lambda_j's$, depending on sequences $w_{t-2}w_{t-1}$, subject to

$$\sum_j \lambda_j(w_{t-2}w_{t-1}) = 1$$

### 3.4.4 Unknown words

Unknown words, also called **out of vocabulary** (OOV) words, are words we haven't seen before. Replace by new word token $< UNK >$ all words that occur fewe than d times in the training set, d some small number. Proceed to train LM as before, treating $< UNK >$ as a regular word. At the test time, replace all unknown words by $< UNK >$ and run the model.

### 3.4.5 Limitations

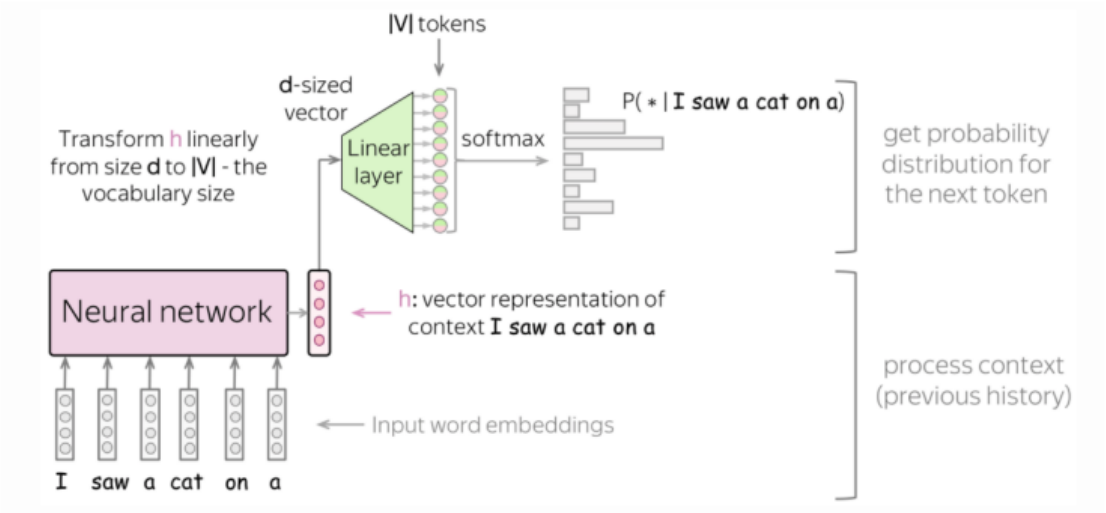
N-gram language models have several **limitations**. Scaling to larger N-gram sizes is problematic, both for computational reasons and because of increased sparsity. Smoothing techniques are intricate and require careful engineering to retain a well-defined probabilistic interpretation. Without additional effort, N-gram models are unable to share statistical strength across similar words.

## 3.5 Neural language models

N-gram language models have been largely supplanted by **neural language models** (NLM). Main advantages of NLM:

- can incorporate arbitrarily distant contextual information, while remaining computationally and statistically tractable
- can generalize better over contexts of similar words, and are more accurate at word-prediction

On the other hand, as compared with N-gram language models, NLM are much more complex, are slower, need more time to train, and are less interpretable.
The idea is to have a vector representation for the previous contet and to generate a probability distribution for the next token. Most natural choice for NN architecture is recurrent neural network but feedforward neural network and convolutional neural network have also been exploited.




### 3.5.1 Feedforward NLM: inference

Like the N-gram language model, the feedforward NLM uses the following approximation:

$$P(w_t|w_{1:t-1}) \approx P(w_t|w_{t-N+1:t-1})$$

and a moving window that can see N-1 words into the past. For $w \in V$, let $ind(w) \in [1..|V|]$ be the **index** associated with w.
We represent each input word $w_t$ as a **one-hot vector** $x_t$ of size $|V|$, defined as follows: element $x_t[ind(w_t)]$ is set to one, and all the other elements of $x_t$ are set to zero.
At the first layer we convert one-hot vectors for the words in the N-1 window into word embeddings of size d and we concatenate the N-1 embeddings. The first hidden layer equation is(assuming N=4):

$$e_t = [Ex_{t-3}; Ex_{t-2}; Ex_{t-1}]$$

where E is a $d \times |V|$ is a learnable matrix with the word embeddings, $x_{t-i}$ is $|V| \times 1$ are 1-hot representation of word $w_{t-i}$ and $e_t$ is a $3d \times 1$ is the concatenation of the embeddings of the N-1 previous words.
The model equations for the remaining layers:

$$h_t = g(We_t + b), \qquad z_t = Uh_t, \qquad \hat{y}_t = \text{softmax}(z_t)$$

where W is a $d_h \times 3d$ learnable matrix, with $d_h$ the size of the second hidden representation, b is a $d_h \times 1$ learnable vector, $h_t$ has dimension $d_h \times 1$, obtained through some activation function g, U is a $|V| \times d_h$ learnable matriz, and $z_t, \hat{y}_t$ are $|V| \times 1$ scores and distribution.
The vector $z_t$ can be thought of as a set of scores over V, also called **logits**: rwa predictions that a classification model generates. Passing these scores through the **softmax** function normalizes into a probability distribution. The element of $\hat{y}_t$ with index $ind(w_t)$ is the probability that the next word is $w_t$:

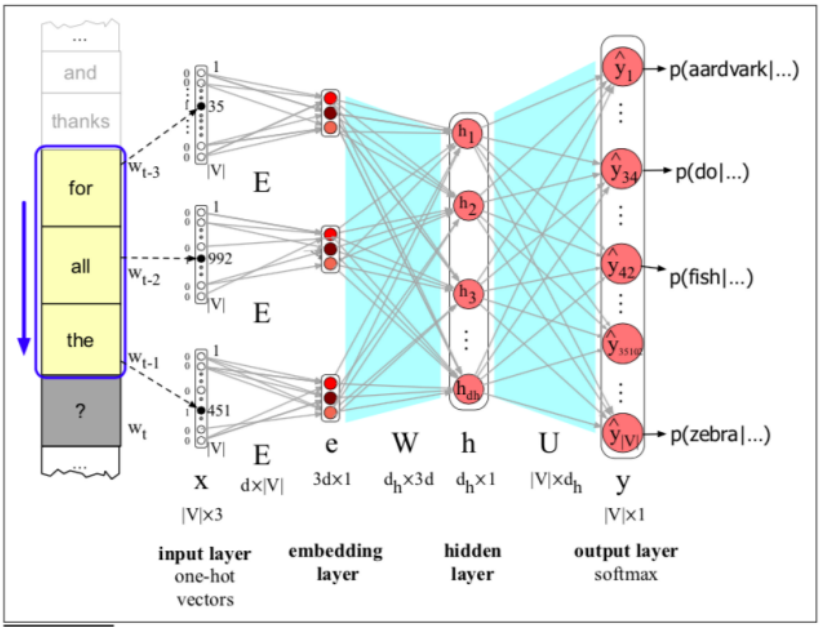
$$\hat{y}_t[ind(w_t)] = P(w_t|w_{t-3:t-1})$$

The **parameters** of the model are $\theta = E, W, U, b$. Let $w_t$ be the word at position t in the trainign data. Then the **true distribution** $y_t$ for the words at position t is a 1-hot vector of size $|V|$ with $y_t[ind(w_t)] = 1$ and $y_t[k] = 0$ everywhere else. We use the cross-entropy loss for training the model:

$$L_{CE}(\hat{y}_t, y_t) = -\sum_{k=1}^{|V|} y_t[k] \log \hat{y}_t[k] = -\log \hat{y}_t[ind(w_t)]$$

Recall the equation for the estimated distribution:

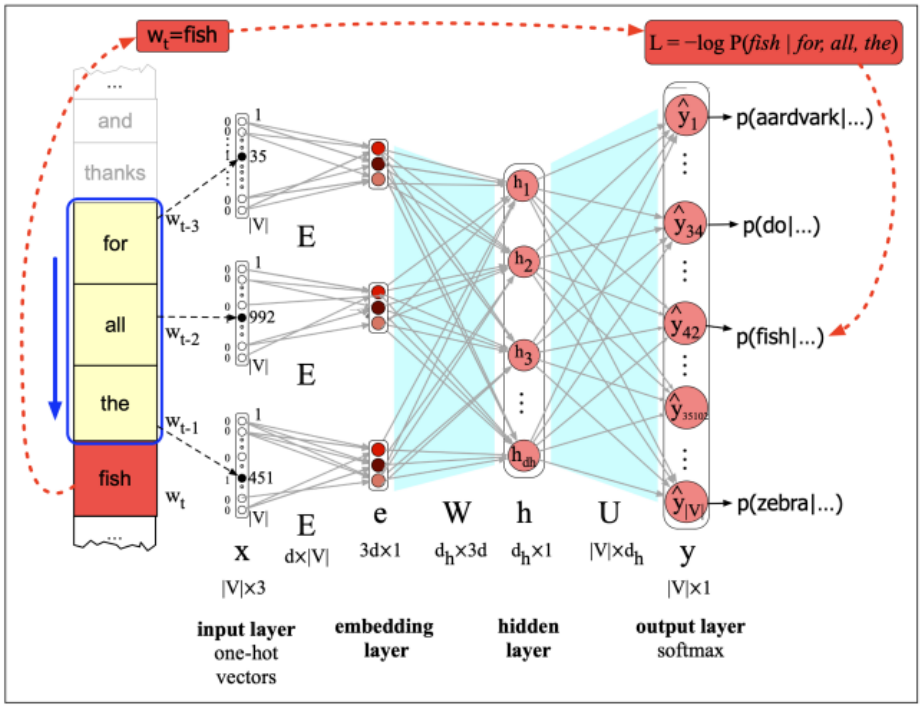
$$\hat{y}_t[ind(w_t)] = P(w_t|w_{t-N+1:t-1})$$




Replacing in the cross-entropy loss equation, we obtain:

$$L_{CE}(\hat{y}_t, y_t) = -\log P(w_t|w_{t-N+1:t-1})$$

Observe that the cros-entropy loss equals the **negative log likelihood** of the training data.




### 3.5.2 Recurrent NLM

RNN language models process the input one word at a time, predicting the next word from the ccurrent word an the previous **hidden state**. RNNs can model probability distribution $P(w_t|w_{1:t-1})$ without the N-1 window approximation of feedforward NLM. The model quations are:

$$e_t = Ex_t, \qquad h_t = g(Uh_{t-1} + We_t), \qquad \hat{y}_t = \text{softmax}(Vh_t)$$

where $x_t$ is a $|V| \times 1$ 1-hot representation of word $w_t$, E is a $d_h \times |V|$ learnable matrix with the word embeddings, U and W are $d_h \times d_h$ learnable matrices, $h_t$ is a $d_h \times 1$ hidden vector at step t, V is a $|V| \times d_h$ learnable matirx

and $\hat{y}_t$ is a $|V| \times 1$ probability distribution.
The vector resulting from $Vh_t$ records the **logits** over the vocabulary V, given the evidence provided by $h_t$. The softmax normalizes the logits, resulting in the **estimated distribution** $\hat{y}_t$ for the word at step t. More precisely, for each word $w \in V$, the element of $\hat{y}_t$ with index ind(w) estimates the probability that the next word is w:

$$\hat{y}_t[ind(w)] = P(w_{t+1} = w|w_{1:t})$$

The number of parameters of the model is $O(|V|)$, since d is a constant. Let $y_t$ be the **true distribution** at

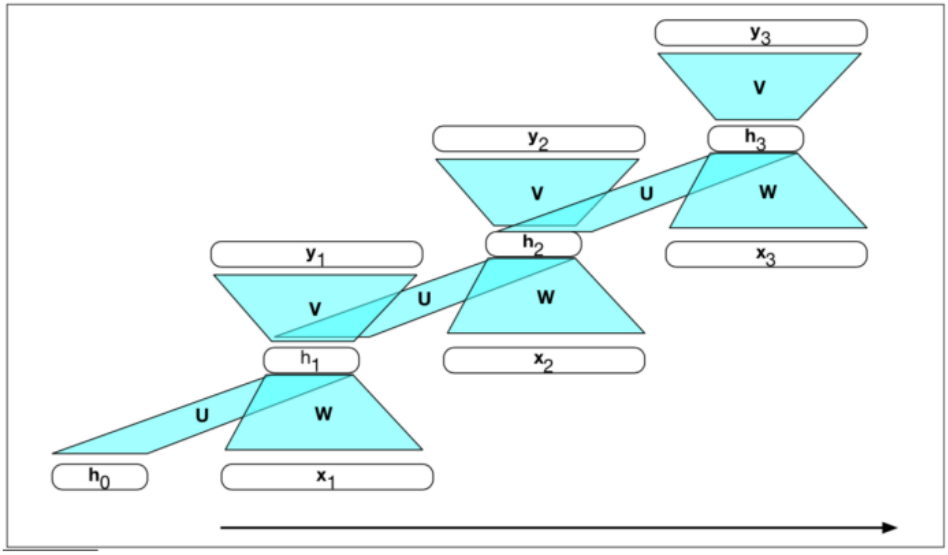



Figure 3.1: The recurrent NLM unrolled in time

step t. This is a 1-hot vector over V, obtained from the training set. We train the model to **minimize** the error in prediccting the true next word $w_{t+1}$ in the training sequence, using cross-entropy as the loss ffunction:

$$L_{CE}(\hat{y}_t, y_t) = -\sum_{w \in V} y_t[ind(w)] \log \hat{y}_t[ind(w)] = -\log \hat{y}_t[ind(w_{t+1})]$$

At each step t durring training the prediction is based on the correct sequence of tokens $w_{1:t}$ and wwe ignore what the model predicted at previous steps. The idea that we always give the model the correct history sequence to predict the next word is called **teacher forcing**. Teacher forcing hase some disantantages: the model is never exposed to prediction mistakes; therefore at inference time the model is not able to recover from errors.

#### Character-level NLM

**Characcter-level NLM** improves modeling of uncommon and unknown words and reduce training parameters due to the small softmax. Performance usually worse than the word-level NLMs, since longer history is needed to predict the next word correcctly. A variety of solutions that combine character and word level information have been proposed, called **character-aware LM**.

### 3.5.3 Practical issues

Both the feedforward NLM and the recurrent NLM learn word embeddings E simultaneously with training the network. Alternatively, one can resort at **freexzing**: use pretrained word embeddings, for istance word2vec and hold E constant while training, and modify the remaining parameters in $\theta$.
In the recurrent NLM model the columns of E provide the learned word embeddings, while the row of V provide a second set of learned word embeddings, that capture relevant aspects of word meaning and function.
**Weigth tying**, also known as **parameter sharing**, means that we impose $E^T = V$. Weigght tying can significantly reduce model size, and has an effect similar to **regularization**, preventing overfitting of the NLM.
RNNs suffer from the **vanishing gradient** problem: past events have weights that decrease exponetially with the distance from actual word $w_t$. Gated recurrent units (GRU) and long-short term memory (LSTM) neural networks are much better in capturing long distance relations.
The last step in NLMs, involving softmax over the entire vocabulary, dominates the ccomputation both at training and at test time. An effective alternative is **hierarchical softmax**, based on word clustering. **Adaptive softmax** is a simple variant of the hierarchical softmax, based on Zipf's law, especially tailored for GPUs.

The text generated by sampling our NLM should be **coherent**: text has to make sense; and **diverse**: the model has to be able to produce very different samples. A very popular method for modifying language model behavior is to change the **softmax temperature** $\tau$:

$$\frac{\exp(\frac{V_i h_t}{\tau})}{\sum_j \exp \frac{V_j h_t}{\tau}}$$

where $V_i$ denotes the i-th row of V.
**Contrastitive evaluation** is used to test specific linguistic constructions in NLM.

# Chapter 4

# Large Language Models