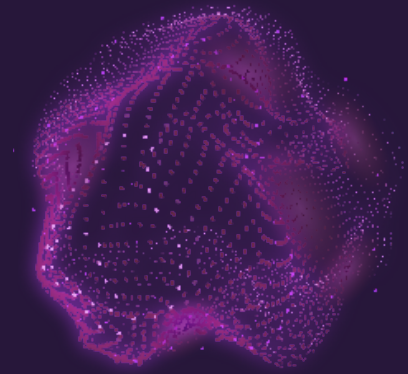


# Natural Language Processing (NLP)

: Dr. Aakash Singh

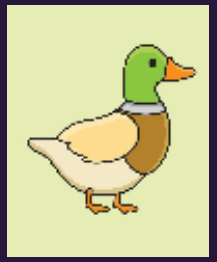
# What is Natural Language

- Refers to the languages that humans use naturally to communicate with each other.
- Such as English, Spanish, Chinese, Hindi, and thousands of others.
- Evolve naturally over time and are governed by complex rules of grammar, syntax, and semantics.
- Can be in spoken, written or sign form.

# Key Features of Natural Language

- **Human-Centric**
  - Used for everyday communication among humans.
  - Developed organically without formal engineering.
- **Ambiguity**
  - Words and sentences can have multiple meanings depending on context.
  - Example: "I saw his duck" (Could mean the bird or an action).
- **Complexity**
  - Governed by rules of grammar and syntax that vary across languages.
  - Includes nuances like tone, intonation, and stress in spoken forms.

or

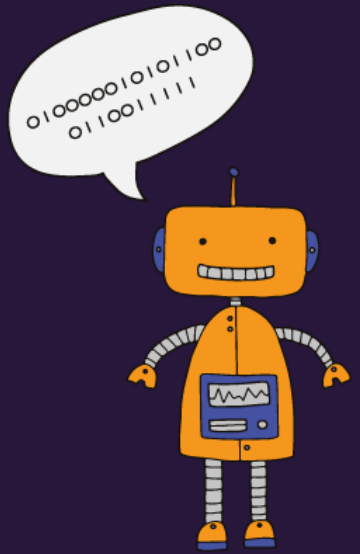


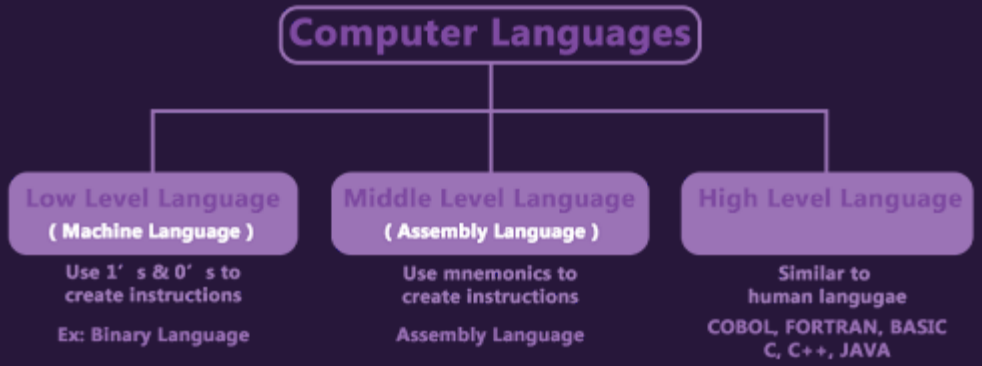
# Key Features of Natural Language

- **Context-Dependence**
  - Meaning often depends on context, shared knowledge, and cultural norms.
  - Example: "Can you pass the salt?" is often a polite request, not a literal inquiry about ability.
- **Dynamic Evolution**
  - Vocabulary, usage, and grammar rules change over time.
  - Example: The addition of words like "selfie" or "emoji" to modern dictionaries.

# Computer Language

- Computers fundamentally understand machine language, which is a low-level language consisting of binary code (0s and 1s).
- This binary system represents the on/off states of electrical signals within a computer's hardware.
- Programming languages bridge human and computer languages by abstracting complexities, enabling humans to communicate instructions to computers more easily.



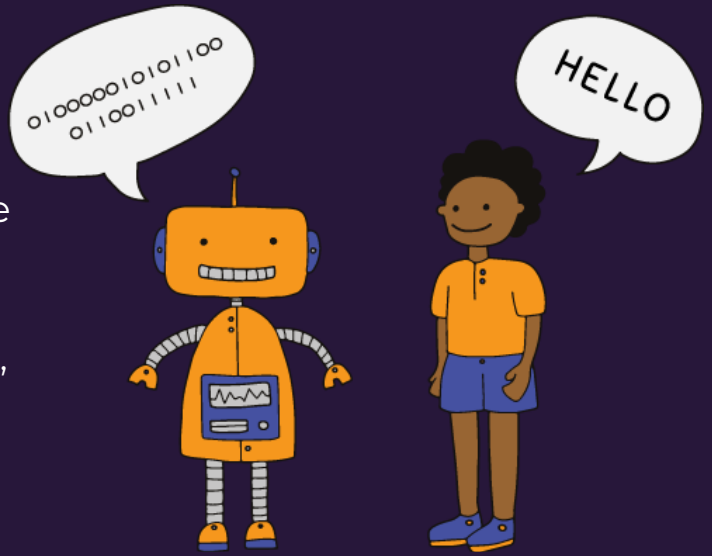



# Difference

| Aspect | Natural Language | Computer (Machine) Language | Programming Language |
|---|---|---|---|
| Purpose | Communication between humans. | Direct communication with computer hardware. | Communication between humans and computers. |
| Structure | Flexible, less rigid; allows for ambiguity. | Rigid and low-level (binary code). | Structured with strict rules and syntax. |
| Readability | Easy for humans to understand and use. | Not possible for humans to read and understand. | Human-readable but requires technical knowledge. |
| Syntax | Varies with context, culture, and grammar. | Binary code, highly specific to machine architecture. | Defined rules and grammar for structured code. |
| Complexity | Complex and evolving over time. | Simple but complex to work with manually. | Varies from simple (Python) to complex (C++) |
| Execution | Not directly executable by machines. | Directly executed by machine (hardware). | Needs to be compiled/interpreted into machine code. |
| Flexibility | Very flexible and ambiguous. | Not flexible; strictly defined by the machine architecture. | Flexible but constrained by syntax rules. |
| Learning Curve | Easy for native speakers to learn. | Very difficult for humans to directly work with. | Moderate to high, depending on the language. |
| Error Tolerance | Tolerates errors like typos and slang. | No tolerance for errors; needs to be exact. | Limited error tolerance, strict syntax rules. |

# Natural Language Processing

- Branch of artificial intelligence that focuses on enabling computers to understand, interpret, process, and generate human language in a meaningful way.
- Allows machines to interact with human language (natural language) rather than relying solely on structured programming languages.
- NLP employs techniques like Rule-Based approch, Machine Learning, and Deep Learning to perform several tasks.



# Major Types of NLP Problems

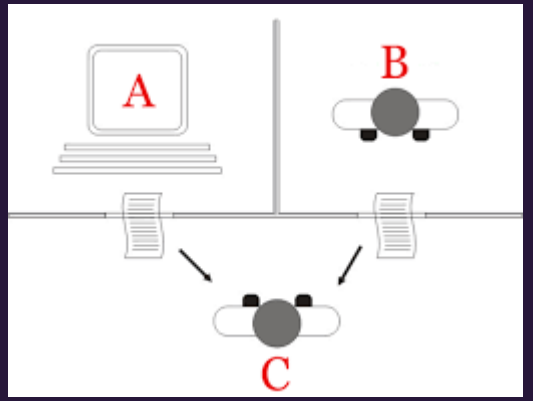
- Text Classification
- Text Generation
- Named Entity Recognition (NER)
- Machine Translation
- Question Answering (QA)
- Text Summarization
- Speech-to-Text & Text-to-Speech
- Text Similarity

# NLP History

- The history of Natural Language Processing (NLP) begins with the desire to create systems that allow humans to communicate with machines using natural language.
- NLP was "born" in the 1950s when early attempts were made to enable computers to process human language.
- Major developments in this era include
  - Turing Test
  - Noam Chomsky’s Grammar

# Turing Test

- Proposed by Alan Turing, the test aimed to determine if a machine could exhibit human-like intelligence in conversation.
- This idea directly inspired the development of early conversational systems like chatbots (e.g., ELIZA) and set the vision for creating machines capable of understanding and generating natural language.
- The Turing Test emphasised the importance of interaction, driving research in dialogue systems, natural language understanding, and semantics.
- Advanced NLP systems, such as GPT-based conversational models, aim to pass the Turing Test by delivering natural, context-aware responses.




Loebner Prize

# Noam Chomsky's Grammar

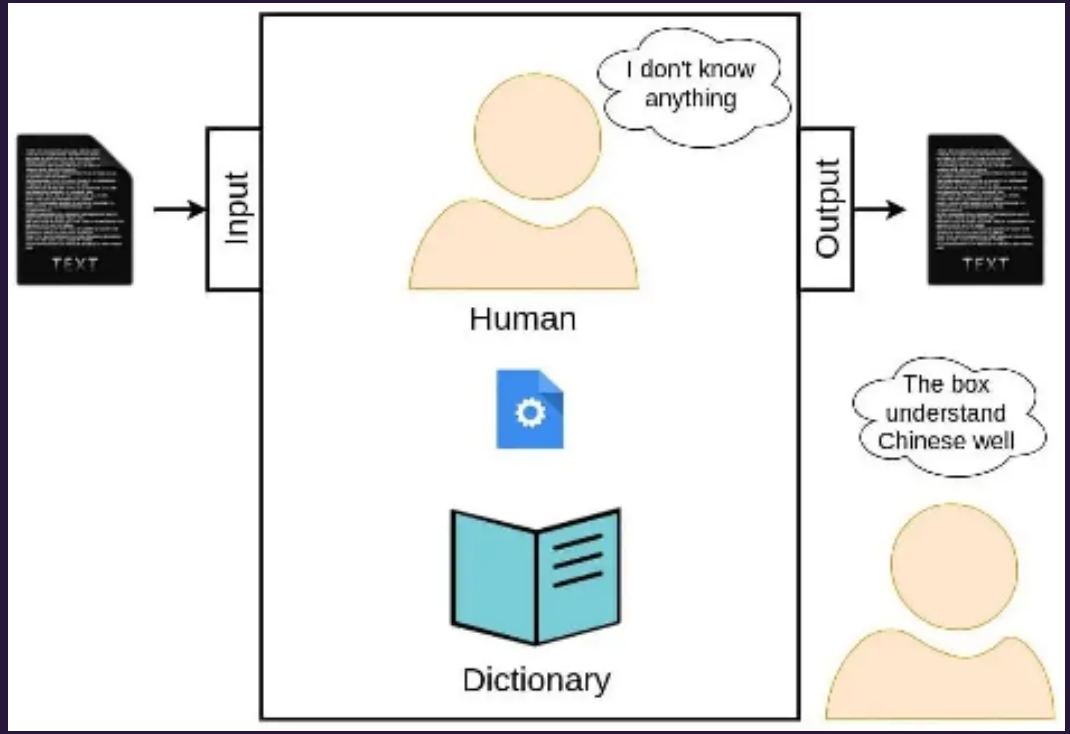
- Defined a formal framework to describe the structure of natural languages.
- Classified formal languages (e.g., regular, context-free) foundational for computational linguistics.
- Enabled the development of algorithms to analyse sentence structures, essential for NLP tasks.
- Proposed a shared underlying structure for all human languages, aiding multilingual NLP.
- Early NLP systems like machine translation and syntax analysers relied on his theories.
- Rule-based approach to describe the structure of a language.

# Georgetown-IBM Experiment

- First Public Demonstration of Machine Translation (1954).
- Involved translation of Russian sentences into English using IBM's IBM 701 computer.
- The translation system used a rule-based word-for-word approach, which focused on matching Russian words with English equivalents.
- Input: 60 sentences of Russian text from the UN (United Nations) and scientific papers.
- Output: English translations displayed on a screen.
- The demonstration created significant excitement in the field of computational linguistics and artificial intelligence.

# Chinese Room

- Introduced by philosopher John Searle in 1980.
- Challenges the claims of the Turing Test and the notion that machines can "understand" language in the same way humans do.
- Focuses on the distinction between syntax and semantics.
- Argues that machines should understand the meaning behind language, not just follow rules.



# Statistical & Machine Learning

- In the 1980s and 1990s, researchers shifted from rule-based systems to Data-Driven approaches.
- The increased availability of computational power and large text corpora facilitated the development of statistical methods such as Hidden Markov Models (HMMs) and Decision Trees.
- IBM's Candide project (1988) introduced statistical methods into machine translation, leading to the development of Statistical Machine Translation (SMT).
- The 1990s saw the development of part-of-speech (POS) tagging.
- This period also saw progress in identifying and classifying named entities (NER).

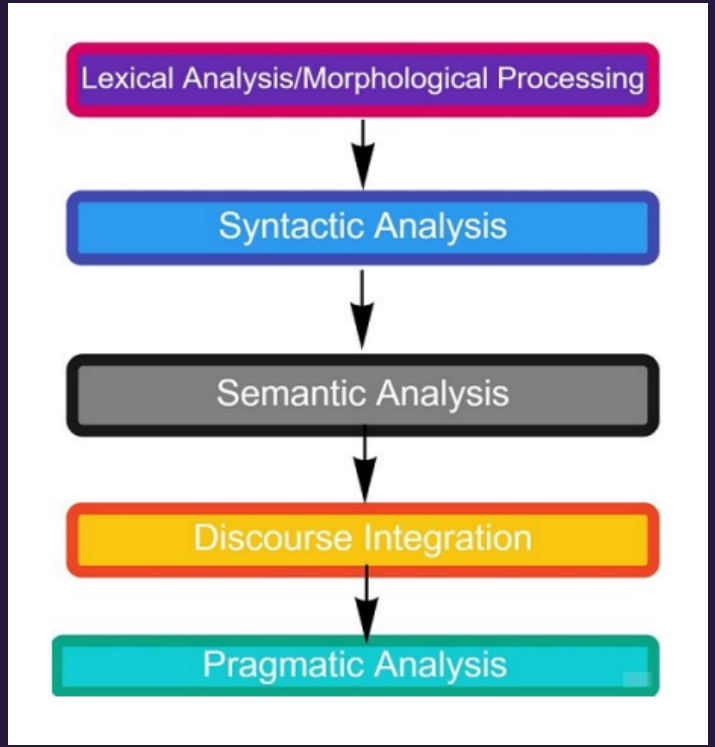
# Neural Networks & Deep Learning

- Neural networks, explored in the 1980s, gained NLP prominence in the 2000s with deep learning advancements.
- The breakthrough came in 2013 with Word2Vec by Tomas Mikolov at Google, using neural networks to map words to high-dimensional vectors, improving word semantics by capturing contextual relationships.
- RNNs, introduced in the 1980s, gained popularity in the 2000s for handling sequence data, especially in language modeling and machine translation, marking a significant shift in NLP tasks.
- LSTM networks, developed in the 1990s, addressed the vanishing gradient problem in RNNs and became widely used in NLP tasks with longer sequences, like speech recognition and text generation.

# Transformer Models

- In 2017, the Transformer architecture by Vaswani et al. revolutionized NLP by using the attention mechanism, enabling parallel word processing and better capturing long-range dependencies, outperforming RNNs and LSTMs.
- In 2018, Google’s BERT model set new performance benchmarks on various NLP tasks. BERT's bidirectional approach to reading text (considering both left and right context) significantly improved tasks like question answering and sentiment analysis.
- Models like GPT-2 (2019) and GPT-3 (2020) by OpenAI advanced text generation, using the Transformer architecture to create coherent, context-aware paragraphs, and became widely used in chatbots and content generation.
- The pretraining-finetuning paradigm, popularized by models like BERT, GPT, and RoBERTa, allowed for more efficient training on large text corpora and specialized finetuning for specific NLP tasks.

# Types of Analysis involved in NLP

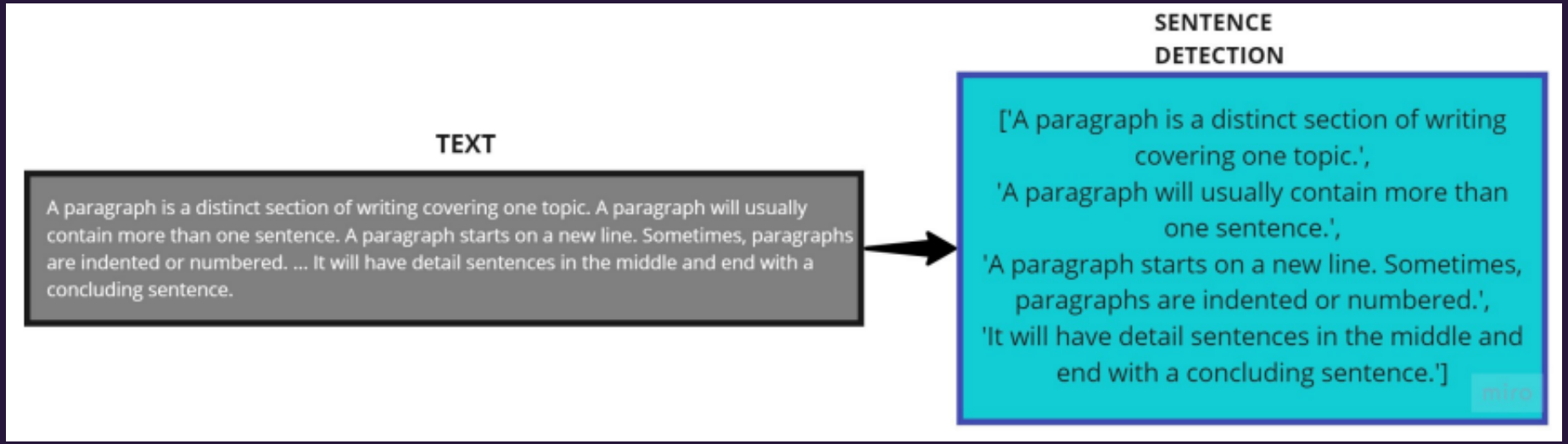


# Lexical Analysis

- The first phase is lexical analysis/morphological processing. In this phase, the sentences, paragraphs are broken into tokens.
- These tokens are the smallest unit of text. It scans the entire source text and divides it into meaningful lexemes.
- For example, The sentence “He goes to college.” is divided into

  [ ‘He’ , ‘goes’ , ‘to’ , ‘college’, ‘.’] .
- There are five tokens in the sentence. A paragraph may also be divided into sentences.

# Lexical Analysis



# Lexical Analysis

- **Tokenization**: The process of splitting text into individual tokens (words, punctuation marks, numbers, etc.).
- **Lexical Categories**: Tokens are classified into different categories, such as nouns, verbs, adjectives, and adverbs (known as parts of speech).
- **Normalization**: Standardizing tokens by converting them to a consistent form (e.g., lowercasing, stemming, or lemmatization).

"Running" → "run" or "runn", "Better" → "bett".

"Running" → "run", "Better" → "good".

- **Stop-words**: Common words that are removed from text before it's analyzed. They are removed because they don't add much meaning to the text.

# Syntactic Analysis/Parsing

- The second phase is Syntactic analysis. In this phase, the sentence is checked whether it is wellformed or not.
- The word arrangement is studied and a syntactic relationship is found between them. It is checked for word arrangements and grammar.
- For example, the sentence “Mohan is goes not him” is rejected by the syntactic parser.

# Syntactic Analysis/Parsing

- **Phrase Structure**: This refers to the way words are grouped into phrases, such as noun phrases (NP), verb phrases (VP), etc.
- **Syntactic Tree**: The structure that visually represents the syntactic relations of words in a sentence, where each node represents a part of the sentence (word or phrase).
- **Grammar Rules**: These are the formal rules (often context-free grammar) used to define the syntactic structure. For example:
  - S → NP VP (A sentence is a noun phrase followed by a verb phrase)
  - NP → Det N (A noun phrase consists of a determiner and a noun)

# Semantic Analysis

- The third phase is Semantic Analysis. In this phase, the sentence is checked for the literal meaning of each word and their arrangement together.
- For example, The sentence “Delhi goes to him” will get rejected by the semantic analyzer because it doesn’t make sense.
- E.g.. “colorless green idea.” This would be rejected by the Symantec analysis as colorless Here; green doesn’t make any sense.

# Discourse Analysis

- Discourse analysis focuses on understanding the structure and meaning of larger units of text or speech (beyond individual sentences). It deals with how the components of a text fit together to convey meaning..
- For example, the word “that” in the sentence “He wanted that” depends upon the prior discourse context.

- **Coherence**: Understanding how different sentences logically flow and contribute to the overall meaning of a discourse.
- **Anaphora and Reference**: Anaphora refers to the use of pronouns or other linguistic elements that refer back to something mentioned earlier in the discourse.
- **Discourse Structure**: Identifying the larger structure of a discourse, such as the introduction, body, and conclusion, or argumentation structure.

# Pragmatic Analysis

- It examines how language is used in practice, including the speaker's intentions, social context, and implied meaning.
- The Goal is to interpret the meaning behind a sentence or phrase in a given context, beyond the literal meaning.

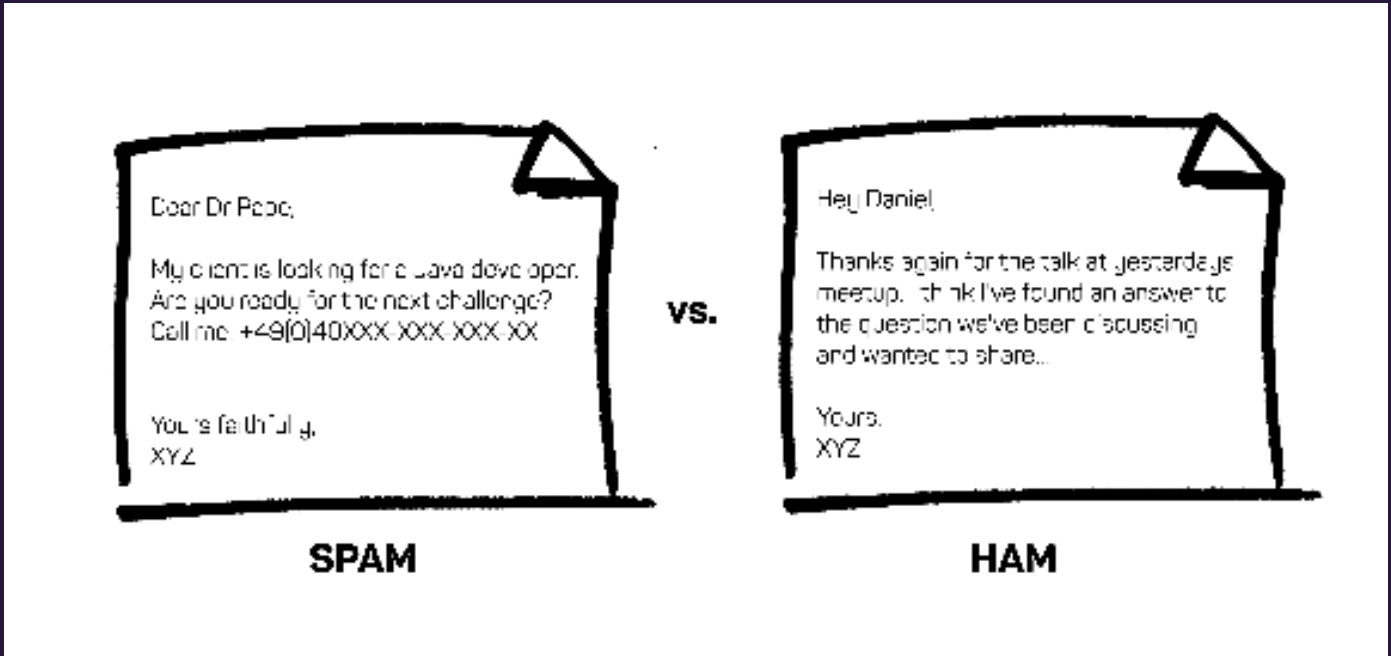
- **Speech Acts**: Analyzing the intent behind an utterance, such as requesting, promising, or questioning (e.g., "Can you open the window?" is a request, not just a question).
- **Implicature**: Understanding what is implied but not explicitly stated (e.g., "I'm running late" could imply that the speaker may need the listener's help or understanding).
- **Contextual Understanding**: Understanding the broader social and situational context in which language is used to determine meaning.

# Popular use cases of NLP

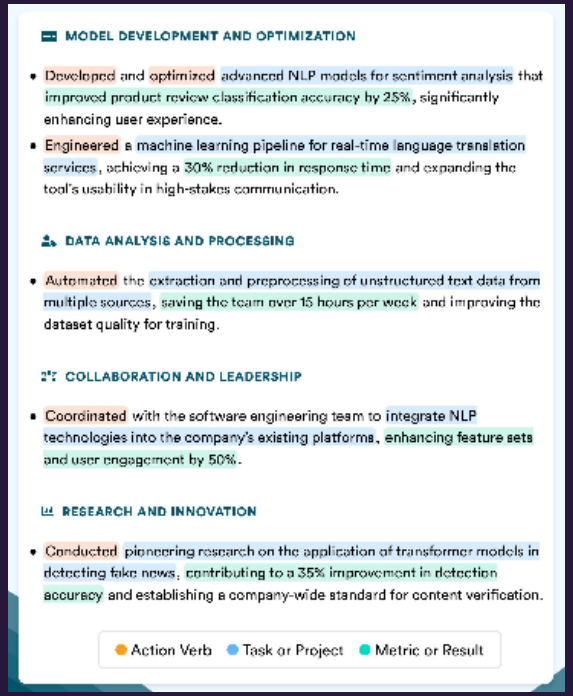
- Sentiment Analysis
- Spam Detection
- Resume Mining
- Chatbots
- Social Media Moderation:
  - Hate Speech Detection
  - Offensive Language Detection
  - Cyberbulling Detection
  - Profane Content Detection
  - Hostility Detection

# Sentiment Analysis

# Spam Detection



# Resume Mining

# ML/ DL compatible data

- Anything that can be converted into numbers, we can apply ML/ DL to it.
- ML/DL models operate on numerical data as inputs. Converting data into numbers makes it compatible with mathematical computations used in algorithms.
- Examples Across Domains:
  - **Text**: Words can be converted into numerical vectors (e.g., word embeddings like Word2Vec, BERT).
  - **Images**: Images are represented as pixel intensity values in matrices.
  - **Audio**: Sound is transformed into numerical waveforms or spectrograms.
  - **Video**: Frames (images) and audio are combined into numerical sequences.
  - **Tabular Data**: Structured data, like spreadsheets, is already in numerical form or can be encoded (e.g., one-hot encoding for categorical data).

# Text to Numbers

- One-Hot Encoding
- Bag of Words
- TF-IDF
- GloVe
- Word2Vec
- Transformer based

# One-Hot Encoding

- It is a technique used to convert categorical data (such as words or characters) into numerical vectors that can be used by machine learning models. In NLP, one-hot encoding is often applied to represent words in a vocabulary.

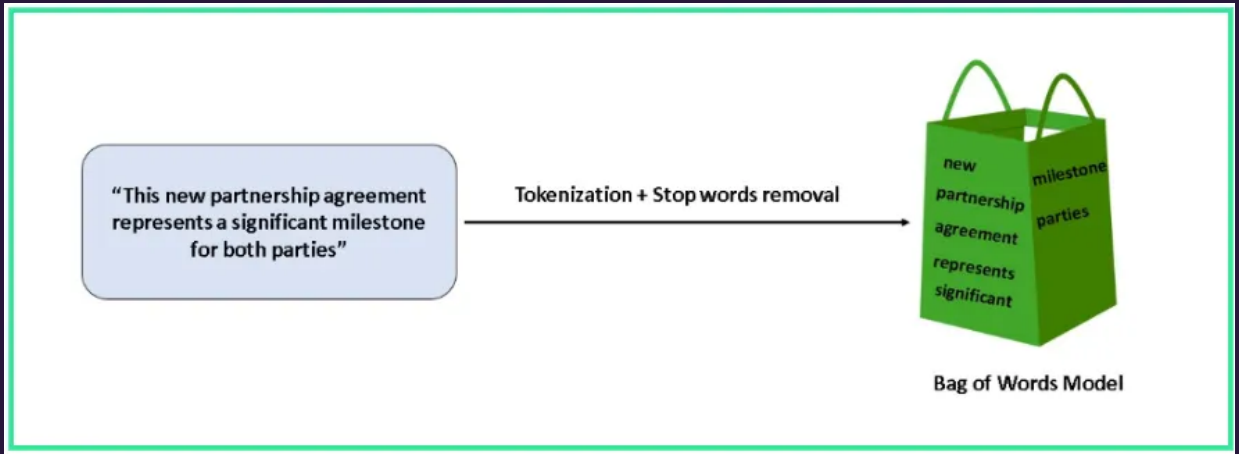
- **Create a Vocabulary**: Identify all unique words in the dataset.
  - For the text "I love NLP", the vocabulary is
  - ["I", "love", "NLP"]
- **Assign an Index**: Assign each word a unique index.
  - Example:
    - "I" → 0
    - “love" → 1
    - "NLP" → 2

# One-Hot Encoding

- **Create Vectors**: Represent each word as a binary vector with the same length as the vocabulary.
  - The index corresponding to the word is marked as 1.
  - All other indices are marked as 0.
  - Example:
    - "I" → [1, 0, 0]
    - "love" → [0, 1, 0]
    - "NLP" → [0, 0, 1]

# Bag of Words (BoW)

- It is a fundamental text representation technique in NLP.
- It converts text into numerical vectors based on word frequency, disregarding grammar, word order, and semantics.
- It focuses solely on the occurrence of words in a given document or corpus.



# Bag of Words (BoW)

- Text Preprocessing
  - Tokenize the text into words.
  - Convert text to lowercase.
  - Remove punctuation, stopwords, and special characters.
- Create a Vocabulary
  - List all unique words in the entire corpus.
- Generate Vectors
  - For each document, create a vector indicating the frequency of each word in the vocabulary.
  - If a word is absent, its frequency is 0.

# Bag of Words (BoW)

- **Corpus**:
  - "He is a good boy."
  - "She is a good girl."
  - "Boys and girls are good.“
- **Step 1: Text Preprocessing**
  - **Remove Stopwords**: Remove common words like "he," "is," "a," "and," etc.
  - **Stemming**: Reduce words to their root form (e.g., "boys" → "boy," "girls" → "girl").
- **Processed Sentences**:
  - "good boy"
  - "good girl"
  - "boy girl good"

# Bag of Words (BoW)

- Step 2: Create Vocabulary
  - Extract unique words: ["good", "boy", "girl"].
- Step 3: Generate Frequency Matrix

| Word | good | boy | girl |
|---|---|---|---|
| Sentence 1 | 1 | 1 | 0 |
| Sentence 2 | 1 | 0 | 1 |
| Sentence 3 | 1 | 1 | 1 |

- This table represents the Bag of Words matrix for the given corpus.
- There exist several variation on Bag of Word approach.

# TF-IDF

- It stands for Term Frequency-Inverse Document Frequency.
- It is a statistical measure used in Natural Language Processing (NLP) to evaluate the importance of a word in a document relative to a collection (or corpus) of documents.
- It combines two components: Term Frequency (TF) and Inverse Document Frequency (IDF).

# TF-IDF

- Term Frequency (TF)
  - Measures how frequently a term appears in a document.
- Formula:

$$\text{TF} = \frac{\text{Number of times a word appears in the document}}{\text{Total number of words in the document}}$$

- Inverse Document Frequency (IDF)
  - Measures how unique or rare a word is across all documents.
  - Words that occur in many documents (e.g., "the," "is") get lower scores, while rare words get higher scores.

$$\text{IDF} = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing the word}}\right)$$

# TF-IDF

- **Corpus**:
  - "He is a good boy."
  - "She is a good girl."
  - "Boys and girls are good.“
- Vocabulary: ["good", "boy", "girl"]
- Compute Term Frequencies (TF)

| Word | Sentence 1: "good boy" | Sentence 2: "good girl" | Sentence 3: "boy girl good" |
|---|---|---|---|
| good | $\frac{1}{2} = 0.5$ | $\frac{1}{2} = 0.5$ | $\frac{1}{3} \approx 0.33$ |
| boy | $\frac{1}{2} = 0.5$ | $0$ | $\frac{1}{3} \approx 0.33$ |
| girl | $0$ | $\frac{1}{2} = 0.5$ | $\frac{1}{3} \approx 0.33$ |

# TF-IDF

- Compute Inverse Document Frequencies (IDF)

$$\text{IDF} = \log\left(\frac{\text{Total Documents (3)}}{\text{Number of Documents Containing the Word}}\right)$$

| Word | Document Count | IDF |
|---|---|---|
| good | 3 | $\log\left(\frac{3}{3}\right) = 0$ |
| boy | 2 | $\log\left(\frac{3}{2}\right) \approx 0.18$ |
| girl | 2 | $\log\left(\frac{3}{2}\right) \approx 0.18$ |

# TF-IDF

- Compute TF-IDF
  - Multiply the TF values by the corresponding IDF values

| Word | Sentence 1 | Sentence 2 | Sentence 3 |
|---|---|---|---|
| good | $0.5 \times 0 = 0$ | $0.5 \times 0 = 0$ | $0.33 \times 0 = 0$ |
| boy | $0.5 \times 0.18 = 0.09$ | $0 \times 0.18 = 0$ | $0.33 \times 0.18 \approx 0.06$ |
| girl | $0 \times 0.18 = 0$ | $0.5 \times 0.18 = 0.09$ | $0.33 \times 0.18 \approx 0.06$ |

# TF-IDF

- Practice question:
  - Document 1 (D1): "The cat sat on the mat.“
  - Document 2 (D2): "The dog sat on the mat."
  - Document 3 (D3): "The cat and the dog are friends.“
  - Document 4 (D4): "Cats and dogs love to play on the mat."

# TF-IDF

- Preprocessing:
  - Tokenize,
  - Lowercase,
  - Remove Punctuation
  - Stop Words

- D1: ["cat", "sat", "mat"]
- D2: ["dog", "sat", "mat"]
- D3: ["cat", "dog", "friend"]
- D4: ["cat", "dog", "love", "play", "mat"]

# TF-IDF

- Term-frequency $$TF = \frac{\text{Number of times a word appears in the document}}{\text{Total number of words in the document}}$$

| Word | D1 TF | D2 TF | D3 TF | D4 TF |
|---|---|---|---|---|
| cat | 1/3 ≈ 0.33 | 0 | 1/3 ≈ 0.33 | 1/5 = 0.2 |
| dog | 0 | 1/3 ≈ 0.33 | 1/3 ≈ 0.33 | 1/5 = 0.2 |
| sat | 1/3 ≈ 0.33 | 1/3 ≈ 0.33 | 0 | 0 |
| mat | 1/3 ≈ 0.33 | 1/3 ≈ 0.33 | 0 | 1/5 = 0.2 |
| friend | 0 | 0 | 1/3 ≈ 0.33 | 0 |
| love | 0 | 0 | 0 | 1/5 = 0.2 |
| play | 0 | 0 | 0 | 1/5 = 0.2 |

# TF-IDF

- Inverse document frequency

| Word | Docs Containing Word | IDF |
|---|---|---|
| cat | 3 | log(4/3) ≈ 0.13 |
| dog | 3 | log(4/3) ≈ 0.13 |
| sat | 2 | log(4/2) ≈ 0.17 |
| mat | 3 | log(4/3) ≈ 0.13 |
| friend | 1 | log(4/1) ≈ 0.60 |
| love | 1 | log(4/1) ≈ 0.60 |
| play | 1 | log(4/1) ≈ 0.60 |

# TF-IDF

- TF x IDF

| Word | D1 TF-IDF | D2 TF-IDF | D3 TF-IDF | D4 TF-IDF |
|---|---|---|---|---|
| cat | 0.33x0.13= 0.04 | 0x0.13= 0 | 0.33x0.13= 0.04 | 0.2x0.13 = 0.03 |
| dog | 0x0.13= 0 | 0.33x0.13= 0.04 | 0.33x0.13= 0.04 | 0.2x0.13 = 0.03 |
| sat | 0.33x0.17= 0.06 | 0.33x0.17= 0.06 | 0x0.17= 0 | 0x0.17 = 0 |
| mat | 0.33x0.13= 0.04 | 0.33x0.13= 0.04 | 0x0.13 = 0 | 0.2x0.13 = 0.03 |
| friend | 0x0.60= 0 | 0x0.60 = 0 | 0.33x0.60= 0.20 | 0x0.60 = 0 |
| love | 0x0.60= 0 | 0x0.60 = 0 | 0x0.60 = 0 | 0.2x0.60 = 0.12 |
| play | 0x0.60= 0 | 0x0.60 = 0 | 0x0.60 = 0 | 0.2x0.60 = 0.12 |

# TF-IDF

- Vector representation
- D1: [0.04, 0, 0.06, 0.04, 0, 0, 0]
- D2: [0, 0.04, 0.06, 0.04, 0, 0, 0]
- D3: [0.04, 0.04, 0, 0, 0.20, 0, 0]
- D4: [0.03, 0.03, 0, 0.03, 0, 0.12, 0.12]

# Document Similarity

- Measures how similar two text documents are in terms of content.
- Used in NLP for tasks like **information retrieval, plagiarism detection, and recommendation systems**.
- Can be computed using various methods such as **Cosine Similarity, Jaccard similarity,  and Euclidean distance**.
- Uses documents as **vectors** for comparison.

# Cosine Similarity

- Measures the **cosine of the angle** between two document vectors in a multi-dimensional space.
- Ranges between **0 and 1**:
  - **1** → Documents are identical.
  - **0** → Documents are completely different.
- Computed as:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \times ||\mathbf{B}||}$$


- **A** and **B** are document vectors.
- ||A|| and ||B|| are vector magnitudes.

# Cosine Similarity

- D1 and D2 similarity:
- Compute the Dot Product
  - D1·D2=
    (0.04×0)+(0×0.04)+(0.06×0.06)+(0.04×0.04)+(0×0)+(0×0)+(0×0)
    =0+0+0.0036+0.0016+0+0+0
    =0.0052
- Compute the Magnitude of Each Vector:

$$||D1|| = \sqrt{(0.04)^2 + (0)^2 + (0.06)^2 + (0.04)^2 + (0)^2 + (0)^2 + (0)^2}$$

$$= \sqrt{0.0016 + 0 + 0.0036 + 0.0016 + 0 + 0 + 0}$$

$$= \sqrt{0.0068} \approx 0.0825$$

$$||D2|| = \sqrt{(0)^2 + (0.04)^2 + (0.06)^2 + (0.04)^2 + (0)^2 + (0)^2 + (0)^2}$$

$$= \sqrt{0 + 0.0016 + 0.0036 + 0.0016 + 0 + 0 + 0}$$

$$= \sqrt{0.0068} \approx 0.0825$$

# Cosine Similarity

- Compute Cosine Similarity

$$\cos(\theta) = \frac{D1 \cdot D2}{||D1|| \times ||D2||}$$

$$= \frac{0.0052}{(0.0825 \times 0.0825)}$$

$$= \frac{0.0052}{0.0068}$$

$$\approx 0.7647$$

# Cosine Similarity

- For D1 and D3

**Step 1: Calculate the dot product $\mathbf{D1} \cdot \mathbf{D3}$**

$$\mathbf{D1} \cdot \mathbf{D3} = (0.04 \times 0.04) + (0 \times 0.04) + (0.06 \times 0) + (0.04 \times 0) + (0 \times 0.20) + (0 \times 0) + (0 \times 0)$$

$$= 0.0016$$

**Step 2: Calculate the magnitudes $\|\mathbf{D1}\|$ and $\|\mathbf{D3}\|$**

$$\|\mathbf{D1}\| = \sqrt{(0.04^2) + (0^2) + (0.06^2) + (0.04^2) + (0^2) + (0^2) + (0^2)} = \sqrt{0.0016 + 0 + 0.0036 + 0.0016} = \sqrt{0.0068} \approx 0.0825$$

$$\|\mathbf{D3}\| = \sqrt{(0.04^2) + (0.04^2) + (0^2) + (0^2) + (0.20^2) + (0^2) + (0^2)} = \sqrt{0.0016 + 0.0016 + 0 + 0 + 0.04} = \sqrt{0.0432} \approx 0.2078$$

**Step 3: Calculate the cosine similarity**

$$\text{Cosine Similarity} = \frac{0.0016}{(0.0825 \times 0.2078)} \approx \frac{0.0016}{0.0171} \approx 0.0937$$

# Document Classification

- Document classification is the process of assigning predefined categories (e.g., sports, politics, spam, non-spam) to text documents using NLP techniques.
- **Applications**:
  - Spam detection (spam vs. non-spam emails)
  - Sentiment analysis (positive, negative, neutral)
  - News categorization (sports, politics, business, etc.)
  - Legal document classification (contracts, policies, etc.)
- **Common Algorithms for Classification**:
  - Naïve Bayes
  - Support Vector Machines (SVM)
  - Logistic Regression
  - Deep Learning (LSTMs, Transformers)

# Naïve Bayes Classification

- The Naïve Bayes classifier is a probabilistic algorithm used for classification tasks.
- It is based on Bayes' Theorem.
- It is called "naïve" because it assumes that the features are independent of each other.
- This is often not true in real-world scenarios (specially for words in text classification)

# Bayes' Theorem

- Named after the Reverend Thomas Bayes.
- Bayes' Theorem is derived using the definition of conditional probability
- **Conditional probability** is known as the possibility of an event or outcome happening, based on the existence of a previous event or outcome.

The **conditional probability** of event $A$ given event $B$ is:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Similarly, the **conditional probability** of event $B$ given event $A$ is:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

# Bayes' Theorem

- From the last equations, we can express the joint probability $P(A \cap B)$P(A∩B) in two ways:

$$P(A \cap B) = P(A|B)P(B)$$

$$P(A \cap B) = P(B|A)P(A)$$

- Since both are equal to $P(A \cap B)$P(A∩B), we equate them:

$$P(A|B)P(B) = P(B|A)P(A)$$

- Rearranging to solve for $P(A|B)$P(A|B)

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

# Bayes' Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$P(A|B)$ **(Posterior Probability)**: Probability of event $A$ occurring given that $B$ has already occurred.

$P(B|A)$ **(Likelihood)**: Probability of event $B$ occurring given that $A$ is true.

$P(A)$ **(Prior Probability)**: Initial probability of event $A$ before considering $B$.

$P(B)$ **(Evidence)**: Total probability of event $B$, across all possible cases.

# Naïve Bayes Classification

- We want to classify a document D (with features/words $w1$,$w2$,...,$wn$) into a class C
- According to Bayes' Theorem, the probability of class C given the document D can be expressed as:

$$P(C|w_1, w_2, \ldots, w_n) = \frac{P(w_1, w_2, \ldots, w_n|C) \cdot P(C)}{P(w_1, w_2, \ldots, w_n)}$$

Where:

- $P(C|w_1, w_2, \ldots, w_n)$ is the **posterior probability**: the probability that the document $D$ belongs to class $C$, given the words in the document.
- $P(w_1, w_2, \ldots, w_n|C)$ is the **likelihood**: the probability of the words appearing in the document given that it belongs to class $C$.
- $P(C)$ is the **prior probability**: the probability that a document belongs to class $C$ without considering the words.
- $P(w_1, w_2, \ldots, w_n)$ is the **evidence**: the total probability of observing the words in the document. This is often ignored in Naïve Bayes because it's constant for all classes.

# Naïve Bayes Classification

- The key assumption in Naïve Bayes is conditional independence.
- It assumes that the words (features) in the document are independent of each other, given the class C.
- This simplifies the likelihood term:

$$P(w_1, w_2, \ldots, w_n|C) = P(w_1|C) \cdot P(w_2|C) \cdot \cdots \cdot P(w_n|C)$$

- The formula becomes:

$$P(C|w_1, w_2, \ldots, w_n) = \frac{P(w_1|C) \cdot P(w_2|C) \cdots P(w_n|C) \cdot P(C)}{P(w_1, w_2, \ldots, w_n)}$$

# Naïve Bayes Classification

- Spam Classification example:

| Email | Label |
|---|---|
| "win lottery now" | Spam |
| "lottery ticket free" | Spam |
| "meeting at office now" | Not Spam |
| "office work is pending" | Not Spam |

- We want to classify the new message:
  - "free lottery now"

# Naïve Bayes Classification

- Using bag of word approach:

| Email No. | win | lottery | now | ticket | free | meeting | at | office | is | work | pending | Label |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Spam |
| 2 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | Spam |
| 3 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | Not Spam |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | Not Spam |

# Naïve Bayes Classification

- For example message we have:
  “free”, “lottery”, “now“
- We need to find 2 probabilities:
  - P(Spam | Message) = P(Spam | " free”, “lottery”, “now“)
  - P(Not Spam | Message) = P(Spam | " free”, “lottery”, “now“)

P(Spam | “free”, “lottery”, “now“) $\propto P(\text{Spam}) \times P(\text{free}|\text{Spam}) \times P(\text{lottery}|\text{Spam}) \times P(\text{now}|\text{Spam})$

P(Not Spam | “free”, “lottery”, “now“)

$\propto P(\text{Not Spam}) \times P(\text{free}|\text{Not Spam}) \times P(\text{lottery}|\text{Not Spam}) \times P(\text{now}|\text{Not Spam})$

# Naïve Bayes Classification

- Problem will arise when a word does not appear in a class.
- This will make is conditional probability 0.
- Since the likelihood probability is the multiple of all the conditional probabilities of words present in example message, it will make net probability 0.
- To avoid this we use Laplace Smoothing:

$$P(W|C) = \frac{\text{Word Count} + 1}{\text{Total Words in Class} + \text{Vocabulary Size}}$$

# Naïve Bayes Classification

**Spam Class Likelihoods**

$$P(\text{free}|\text{Spam}) = \frac{1+1}{5+11} = \frac{2}{16} = 0.125$$

$$P(\text{lottery}|\text{Spam}) = \frac{2+1}{5+11} = \frac{3}{16} = 0.188$$

$$P(\text{now}|\text{Spam}) = \frac{1+1}{5+11} = 0.125$$

Total Spam Probability:

$$P(\text{Spam}|X) \propto P(\text{Spam}) \times P(\text{free}|\text{Spam}) \times P(\text{lottery}|\text{Spam}) \times P(\text{now}|\text{Spam})$$

$$= \text{0.5 x 0.125 x 0.188 x 0.125} = 0.00147$$

# Naïve Bayes Classification

## Not Spam Class Likelihoods

$$P(\text{free}|\text{Not Spam}) = \frac{0+1}{7+11} = \frac{1}{18} = 0.056$$

$$P(\text{lottery}|\text{Not Spam}) = \frac{0+1}{7+11} = 0.056$$

$$P(\text{now}|\text{Not Spam}) = \frac{1+1}{7+11} = 0.111$$

Total Not Spam Probability:

$$P(\text{Not Spam}|X) \propto P(\text{Not Spam}) \times P(\text{free}|\text{Not Spam}) \times P(\text{lottery}|\text{Not Spam}) \times P(\text{now}|\text{Not Spam})$$

$$= \quad 0.5 \text{ x } 0.056 \text{ x } 0.056 \text{ x } 0.111 \quad = 0.00017$$

# Naïve Bayes Classification

- Practice example:
- “Call office now”

| Email No. | win | lottery | now | ticket | free | meeting | at | office | is | work | pending | Label |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Spam |
| 2 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | Spam |
| 3 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | Not Spam |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | Not Spam |

# Naïve Bayes Classification

P(call/ spam) =
= (0+1)/(5+11) = 1/16 = 0.0625

P(office/ spam) =
= (0+1)/(5+11) = 1/16 = 0.0625

P(now/ spam) =
= (1+1)/(5+11) = 2/16 = 0.125

P(spam/ “call”, “office”, “now”) ∝

= 0.5 x 0.0625 x 0.0625 x 0.125
= 0.00024

P(call/ Not spam) =
= (0+1)/(7+11) = 1/18 = 0.056

P(office/ Not spam) =
= (1+1)/(7+11) = 2/18 = 0.1111

P(now/ Not spam) =
= (1+1)/(7+11) = 2/18 = 0.1111

P(Not spam/ “call”, “office”, “now”) ∝

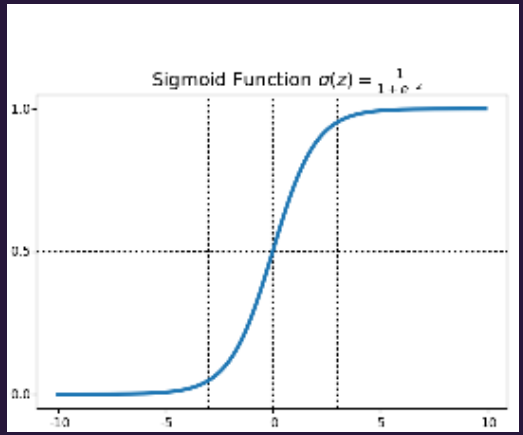
= 0.5 x 0.056 x 0.1111 x 0.1111
= 0.00035

# Logistic Regression Classification

- A supervised machine learning algorithm used for binary classification.
- Computes the probability that an input belongs to one of two classes (e.g., Spam vs. Not Spam).
- Uses the sigmoid function to convert numerical values into probabilities.

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

- Where x is a linear combination of the input features.



# Logistic Regression Classification

- Training Logistic Regression:
  - The model learns a function that predicts whether an email is "Spam" or "Not Spam" based on word presence.
  - It assigns weights to each word, showing its contribution to predicting spam.
  - X = $(w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n)$

$$P(Spam) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n)}}$$

# Logistic Regression Classification

- Sigmoid Function for Classification
  - Logistic regression applies a sigmoid function to compute a probability.

$$P = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n)}}$$

where:

- $w_0$ is the bias term,
- $w_1, w_2, \ldots, w_n$ are learned weights,
- $x_1, x_2, \ldots, x_n$ are word presence values (0 or 1).

  - Decision Boundary:
    - If $P > 0.5$, the email is classified as Spam.
    - If $P \leq 0.5$, it is classified as Not Spam.

# Logistic Regression Classification

- Limitations:
  - Assumes a linear relationship among the features.
  - Performance degrades with huge vocabulary sizes.
  - Ideally designed for binary classifications only.

- How to use it for multi-class classifications?
  - One vs. Rest approach
  - One vs. one approach
  - Use SoftMax function

# One vs. Rest

- Also known as **One-vs-All (OvA)**.
- Trains one binary classifier per class while treating all other classes as a single combined class.
- The class with the highest probability is chosen as the prediction.

- How It Works:
- For N classes, train N binary classifiers:
- Each classifier distinguishes one class vs. all others.
- For a test sample, all classifiers assign probabilities.
- The class with the highest probability is the final prediction.

- Example (for 3 classes: A, B, C)
  - **Model 1:** "A vs. (B & C)"
  - **Model 2:** "B vs. (A & C)"
  - **Model 3:** "C vs. (A & B)"
  - Final prediction = Class with **highest probability.**

# One vs. One approach

- Trains binary classifiers for every possible pair of classes.
- The final prediction is based on a voting system.

- How It Works:
  - For N classes, train N(N-1)/2 binary classifiers.
  - Each classifier distinguishes between two classes only.
  - For a test sample, all classifiers vote for a class.
  - The class that gets the most votes is the final prediction.

- **Model 1:** "A vs. B"
- **Model 2:** "A vs. C"
- **Model 3:** "B vs. C"
- Each classifier gives a result, and the class with the most votes is selected.

# Parametric Vs Non-Parametric

- Parametric models make strong assumptions about the functional form of the relationship between input and output variables. They have a fixed number of parameters, regardless of the size of the dataset.
- Explicit training: The model tries to learn the value of parameters.
- Limited flexibility: Less flexible, constrained by the fixed number of parameters.
- Less prone to overfitting but can underfit if assumptions are incorrect.
- Example: Linear regression, Logistic regression, ANN

- Non-parametric models do not make strong assumptions about the form of the relationship between input and output variables. The number of parameters grows with the size of the dataset, allowing them to adapt to more complex patterns.
- No Explicit Training/ Implicit training: The model store & use entire dataset for prediction
- High flexibility: Can capture complex, non-linear relationships.
- More prone to overfitting, especially with limited data.
- Example: k-Nearest Neighbors, Decision tree, Random forest, SOM

# Terminologies

| | Definition | Examples |
|---|---|---|
| **Field** | Broad domain where AI/ ML is applied. | Natural Language Processing (NLP), Computer Vision (CV), Speech Processing, Time Series Analysis. |
| **Task/Problem** | The specific type of problem to solve within a field. | Classification, Regression, Clustering, Translation, Summarization, Question/ Answering. |
| **Approach** | The method used to solve a task/problem. | Rule-Based Systems, Machine Learning (ML), Deep Learning (DL) |

# Document Clustering

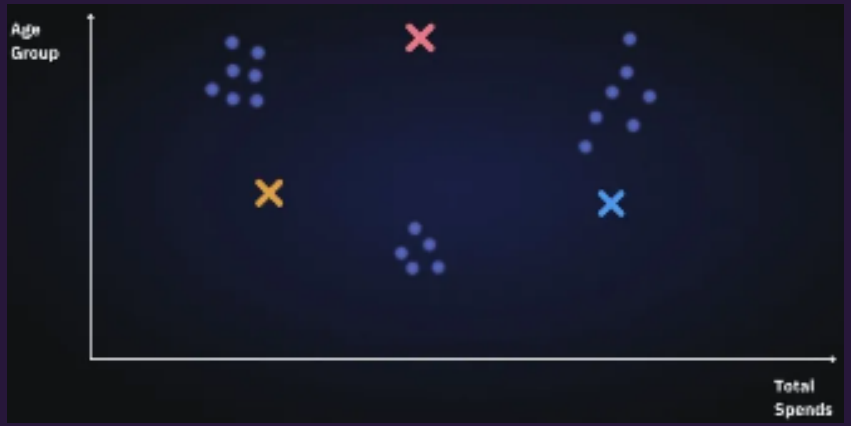
- Text clustering is an unsupervised machine learning technique that groups similar texts together based on their content, without predefined labels.
- **Text Representation**:
  - Bag-of-Words (BoW)
  - Term Frequency-Inverse Document Frequency (TF-IDF)
  - Word Embeddings (Word2Vec, GloVe, FastText)
  - Contextual embeddings (BERT, GPT)
- **Techniques**:
  - Partition-based clustering
  - Hierarchical clustering
  - Density-based clustering
  - Topic Modeling

# K-Means Clustering

- K-Means is an unsupervised machine learning algorithm used for clustering similar data points into K groups (clusters).
- It minimizes the distance between data points and the cluster centre (centroid).
- Use case: an online store uses K-Means to group customers based on purchase frequency and spending.
- The idea is to create segments like Budget Shoppers, Frequent Buyers and Big Spenders for personalized marketing.

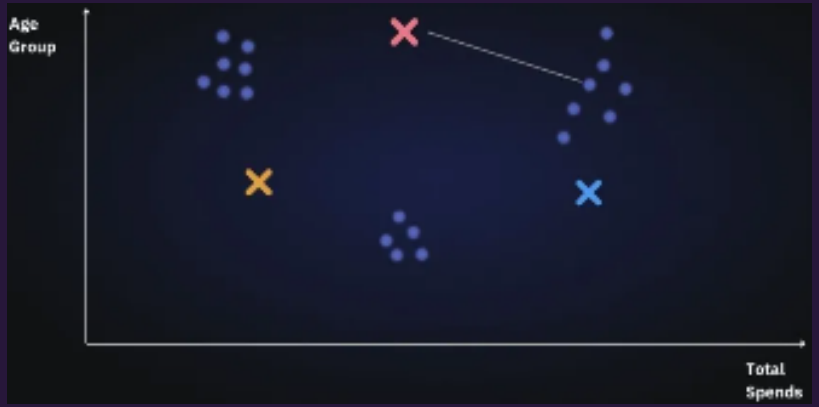
# K-Means Clustering

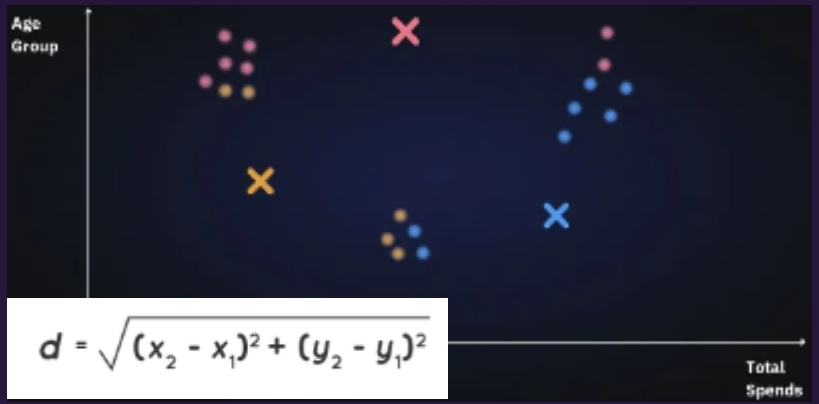
- Steps involved:
- **Choosing the number of clusters**
  The first step is to define the K number of clusters in which we will group the data. Let's select K=3.
- **Initializing centroids**
  Centroid is the center of a cluster but initially, the exact center of data points will be unknown so, we select random data points and define them as centroids for each cluster.



# K-Means Clustering

- **Assign data points to the nearest cluster**
  Now that centroids are initialized, the next step is to assign data points Xn to their closest cluster centroid Ck
- In this step, we will first calculate the distance between data point X and centroid C using Euclidean Distance metric.
- And then choose the cluster for data points where the distance between the data point and the centroid is minimum.

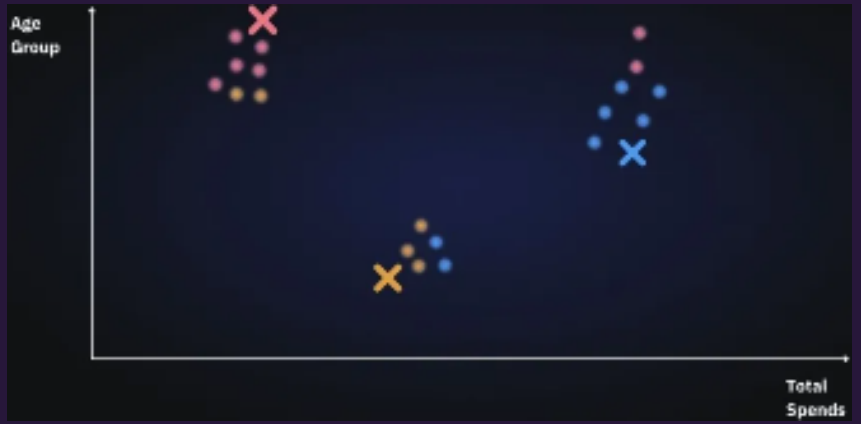



$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

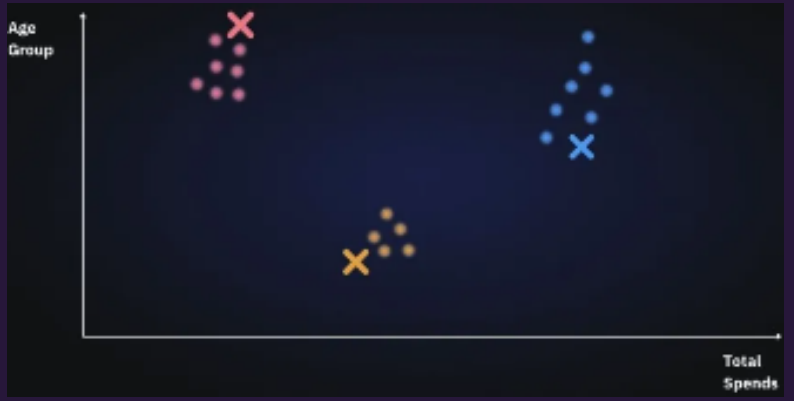
# K-Means Clustering

- **Re-initialize centroids**
  Next, we will re-initialize the centroids by calculating the average of all data points of that cluster.



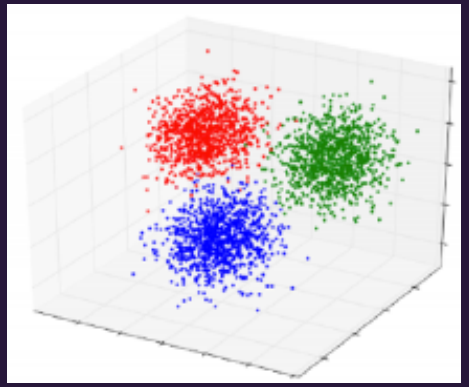
# K-Means Clustering

- **Repeat last two steps**
  We will keep repeating last two steps until we have optimal centroids and the assignments of data points to correct clusters are not changing anymore.



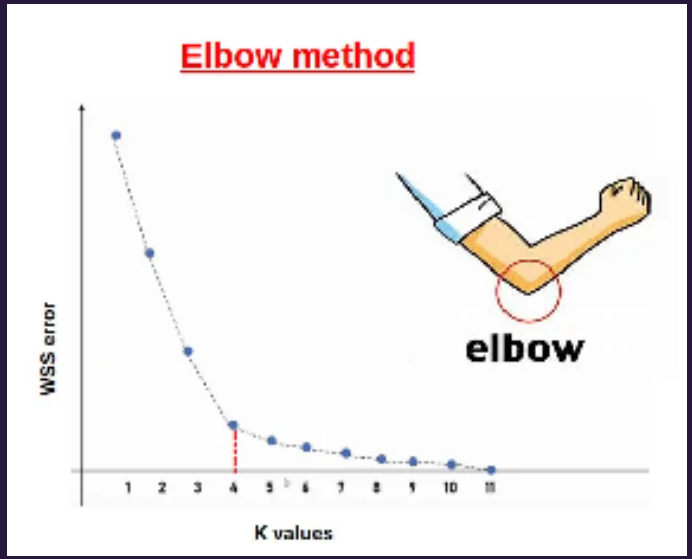
# K-Means Clustering

- As the number of features increases the number of dimensions also increases.
- In the case of text we have words as features (BoW).
- So vocabulary size = dimensions involved.



# K-Means Clustering

- How do you decide the value of k?
- In the Elbow Method, we systematically experiment with different numbers of clusters (K) ranging from 1 to 10.
- With each K value, we compute the Within-Cluster Sum of Squares (WCSS).
- When we plot WCSS against K, the resulting graph resembles an elbow.
- As we increase the number of clusters, the WCSS value begins to decrease.



# NLP in Information Retrieval

- Information Retrieval (IR) is the process of finding relevant information from a large collection of unstructured data (usually text), in response to a user query.
- Natural Language Processing (NLP) enhances Information Retrieval (IR) by enabling machines to understand, process, and retrieve relevant information from text-based data.

- **Applications:**
  - Web Search Engines (Google, Bing)
  - Chatbots & Conversational Search
  - Legal & Medical Document Retrieval
  - Question Answering Systems

# NLP in Information Retrieval

- **Corpus**
  1. **Document 1:** The apple is red.
  2. **Document 2:** Bananas are yellow and sweet.
  3. **Document 3:** Grapes can be red or green.

- **Query:**
  *"red sweet fruit"*

# NLP in Information Retrieval

- Use already known TF-IDF
- **Step 1: Tokenization (Unique Terms in Corpus)**
  - Terms: ["apple", "red", "bananas", "yellow", "sweet", "grapes", "green"]

# NLP in Information Retrieval

- **Step 2: Term Frequency (TF) Table**

| Term | Document 1 | Document 2 | Document 3 | Query |
|---|---|---|---|---|
| apple | 1 | 0 | 0 | 0 |
| red | 1 | 0 | 1 | 1 |
| bananas | 0 | 1 | 0 | 0 |
| yellow | 0 | 1 | 0 | 0 |
| sweet | 0 | 1 | 0 | 1 |
| grapes | 0 | 0 | 1 | 0 |
| green | 0 | 0 | 1 | 0 |

# NLP in Information Retrieval

$$IDF = \log\left(\frac{N}{DF}\right)$$

| Term | DF | IDF (log(N/DF) |
|---|---|---|
| apple | 1 | 0.477 |
| red | 2 | 0.176 |
| bananas | 1 | 0.477 |
| yellow | 1 | 0.477 |
| sweet | 1 | 0.477 |
| grapes | 1 | 0.477 |
| green | 1 | 0.477 |

# NLP in Information Retrieval

**Step 2: Compute TF-IDF Scores**

| Term | Document 1 | Document 2 | Document 3 | Query |
|---|---|---|---|---|
| apple | 0.477 | 0 | 0 | 0 |
| red | 0.176 | 0 | 0.176 | 0.176 |
| bananas | 0 | 0.477 | 0 | 0 |
| yellow | 0 | 0.477 | 0 | 0 |
| sweet | 0 | 0.477 | 0 | 0.477 |
| grapes | 0 | 0 | 0.477 | 0 |
| green | 0 | 0 | 0.477 | 0 |

# NLP in Information Retrieval

- Step 5: Compute Cosine Similarities

The cosine similarity between two vectors $A$ and $B$ is given by:

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

where:

- $A \cdot B$ is the **dot product** of the two vectors.
- $\|A\|$ and $\|B\|$ are the **magnitudes** (Euclidean norm) of the vectors.

# NLP in Information Retrieval

**Document 1: "The apple is red."**

- Query Vector: $[0, 0.176, 0, 0, 0.477, 0, 0]$
- Document 1 Vector: $[0.477, 0.176, 0, 0, 0, 0, 0]$

Dot Product:

$$(0.477 \times 0) + (0.176 \times 0.176) + (0 \times 0) + (0 \times 0) + (0 \times 0.477) + (0 \times 0) + (0 \times 0) = 0.031$$

Magnitude:

$$\|D_1\| = \sqrt{(0.477)^2 + (0.176)^2} = \sqrt{0.227 + 0.031} = \sqrt{0.258} = 0.508$$

$$\|Q\| = \sqrt{(0.176)^2 + (0.477)^2} = \sqrt{0.031 + 0.227} = \sqrt{0.258} = 0.508$$

$$\cos(\theta) = \frac{0.031}{0.508 \times 0.508} = \frac{0.031}{0.258} = 0.12$$

# NLP in Information Retrieval

**Document 2: "Bananas are yellow and sweet."**

- Query Vector: $[0, 0.176, 0, 0, 0.477, 0, 0]$
- Document 2 Vector: $[0, 0, 0.477, 0.477, 0.477, 0, 0]$

Dot Product:

$$(0 \times 0) + (0 \times 0.176) + (0.477 \times 0) + (0.477 \times 0) + (0.477 \times 0.477) + (0 \times 0) + (0 \times 0) = 0.228$$

Magnitude:

$$\|D_2\| = \sqrt{(0.477)^2 + (0.477)^2 + (0.477)^2} = \sqrt{0.228 + 0.228 + 0.228} = \sqrt{0.683} = 0.826$$

$$\|Q\| = 0.508$$

$$\cos(\theta) = \frac{0.228}{0.826 \times 0.508} = \frac{0.228}{0.42} = 0.54$$

# NLP in Information Retrieval

**Document 3: "Grapes can be red or green."**

- Query Vector: $[0, 0.176, 0, 0, 0.477, 0, 0]$
- Document 3 Vector: $[0, 0.176, 0, 0, 0, 0.477, 0.477]$

Dot Product:

$$(0 \times 0) + (0.176 \times 0.176) + (0 \times 0) + (0 \times 0) + (0 \times 0.477) + (0.477 \times 0) + (0.477 \times 0) = 0.031$$

Magnitude:

$$\|D_3\| = \sqrt{(0.176)^2 + (0.477)^2 + (0.477)^2} = \sqrt{0.031 + 0.228 + 0.228} = \sqrt{0.487} = 0.698$$

$$\|Q\| = 0.508$$

$$\cos(\theta) = \frac{0.031}{0.698 \times 0.508} = \frac{0.031}{0.354} = 0.087$$

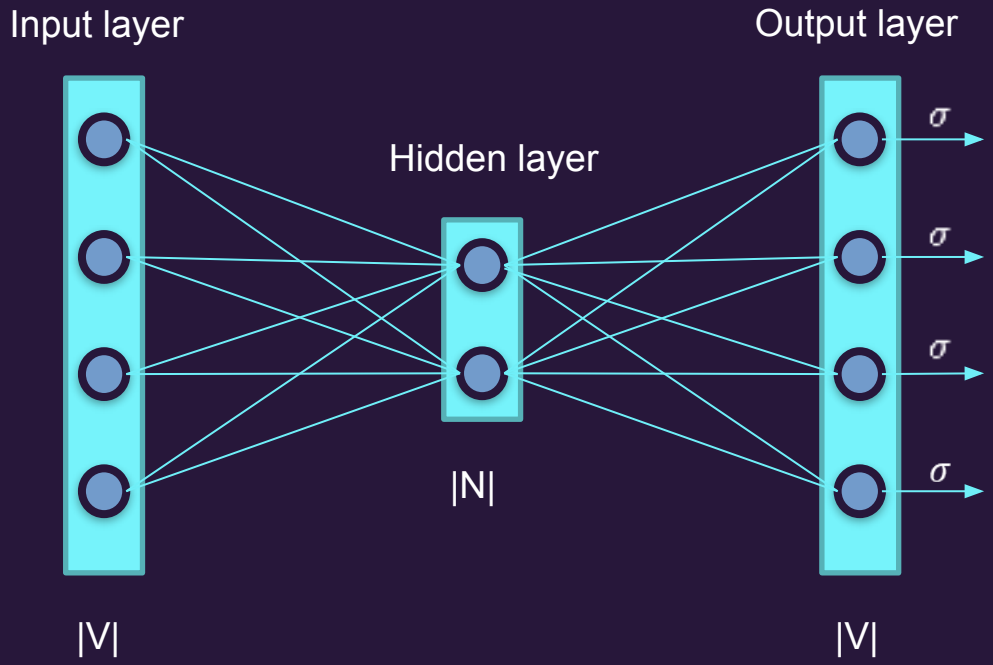
# NLP in Information Retrieval

- **Final Ranked Order**
  1. **Document 2** (Most relevant, highest similarity: **0.54**)
  2. **Document 1** (Second highest similarity: **0.12**)
  3. **Document 3** (Least relevant, lowest similarity: **0.087**)

# Neural Networks for Embedding

- Simple, shallow architecture of Word2Vec
  - Input layer: One-hot vector for the target word (size = vocab size, say |V|)
  - Hidden layer: Size = embedding dimension (say |N|)
  - Output layer: Predicts context words (also size = vocab size)

# Neural Networks for Embedding

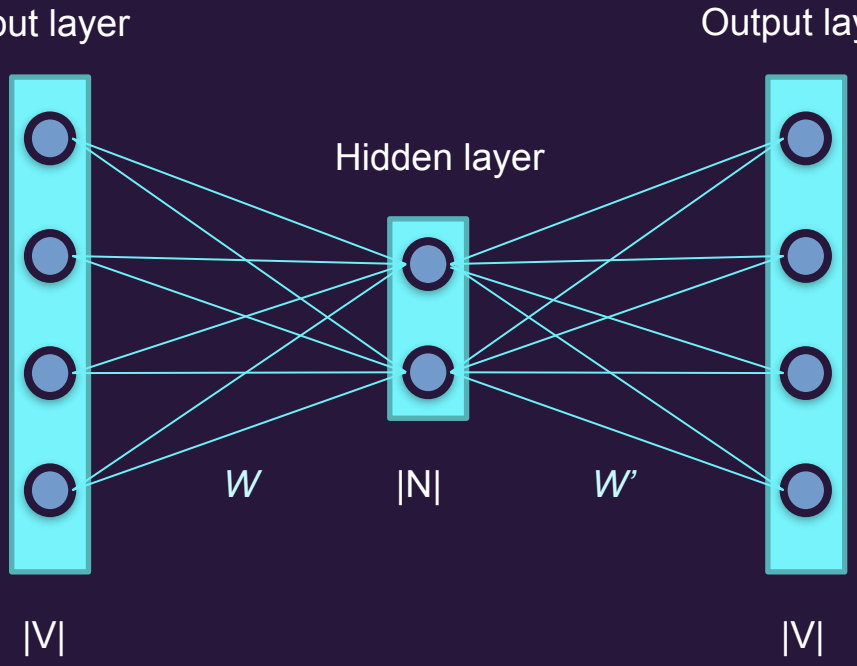


# Neural Networks for Embedding

- Let’s say:
  - Corpus: “The cat sat on”
  - Vocabulary = ["the", "cat", "sat", "on"] → V=4
  - Embedding dimension N=2 (tiny, just to visualize)

- So:
- Input weight matrix W: shape = 4×2
- Output weight matrix W': shape = 2×4

# Neural Networks for Embedding




$$W = \begin{bmatrix} 0.01 & 0.02 \\ 0.03 & -0.01 \\ -0.02 & 0.04 \\ 0.00 & -0.03 \end{bmatrix}$$

Random Initialization

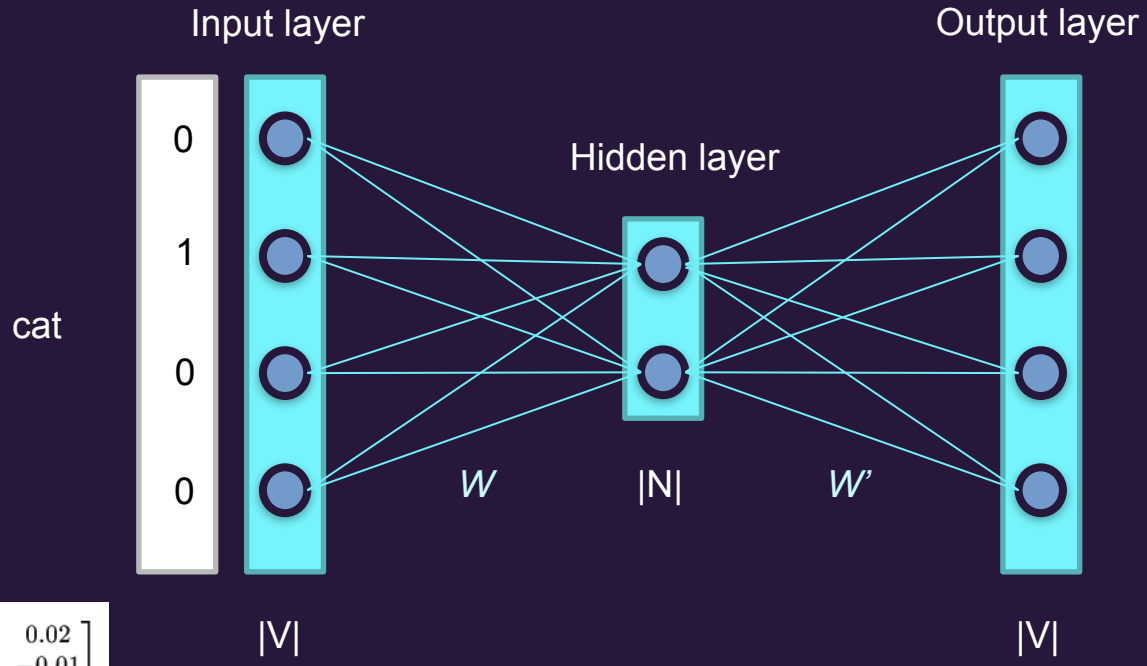
$$W' = \begin{bmatrix} 0.02 & -0.01 & 0.03 & 0.00 \\ -0.02 & 0.01 & 0.00 & 0.04 \end{bmatrix}$$

Random Initialization

# Neural Networks for Embedding

- We represent the words using one-hot encoding.
- Vocabulary = ["the", "cat", "sat", "on"]
- One-hot for "cat":
  - x=[0,1,0,0](only index 1 is 1)
- Training Dataset (target, context):
  - (the, cat)
  - (cat, the), (cat, sat)
  - (sat, cat), (sat, on)
  - (on, sat)

# Neural Networks for Embedding




$$W = \begin{bmatrix} 0.01 & 0.02 \\ 0.03 & -0.01 \\ -0.02 & 0.04 \\ 0.00 & -0.03 \end{bmatrix}$$

Random Initialization

$$W' = \begin{bmatrix} 0.02 & -0.01 & 0.03 & 0.00 \\ -0.02 & 0.01 & 0.00 & 0.04 \end{bmatrix}$$

Random Initialization

# Neural Networks for Embedding

$$h = x^T W$$

$$\begin{array}{|cccc|} \hline 0 & 1 & 0 & 0 \\ \hline \end{array} \cdot W = \begin{bmatrix} 0.01 & 0.02 \\ 0.03 & -0.01 \\ -0.02 & 0.04 \\ 0.00 & -0.03 \end{bmatrix} = [0.03, -0.01]$$

$$u = h \cdot W'$$

$$[0.03, -0.01] \cdot \begin{bmatrix} 0.02 & -0.01 & 0.03 & 0.00 \\ -0.02 & 0.01 & 0.00 & 0.04 \end{bmatrix} = \begin{bmatrix} 0.0008 \\ -0.0004 \\ 0.0009 \\ -0.0004 \end{bmatrix}$$

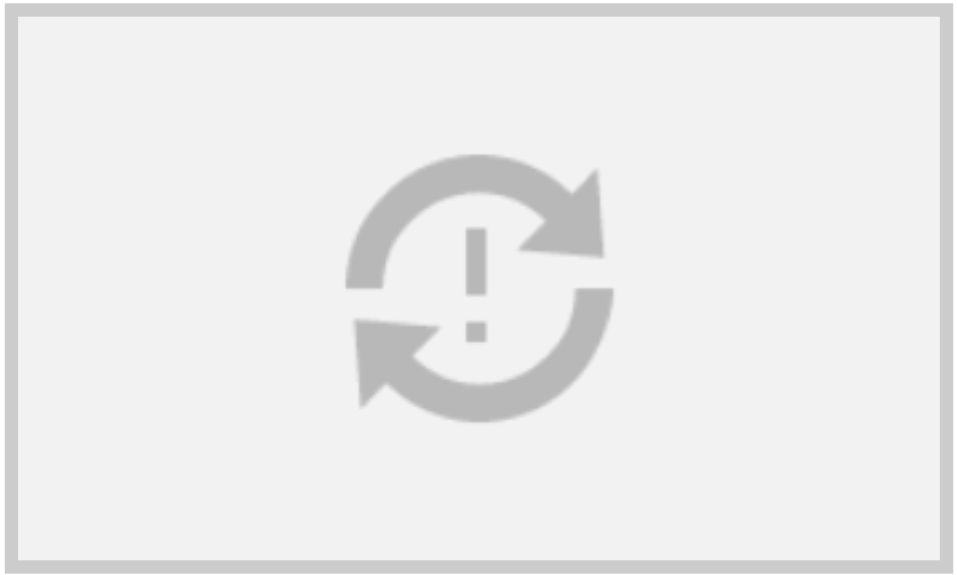
$$\sigma(u_{\text{“sat”}}) = \frac{1}{1 + e^{-0.0009}} \approx 0.5002$$

# Neural Networks for Embedding

- Backpropagation
  - Loss = difference between predicted and actual (true label = 1 for "sat")
  - Gradients are calculated
  - Both W and W' are updated via gradient descent:
- This process repeats for millions of target-context pairs extracted from a large corpus.

# Feedforward Neural Networks

Can also be called **multi-layer perceptrons** (or **MLPs**) for historical reasons



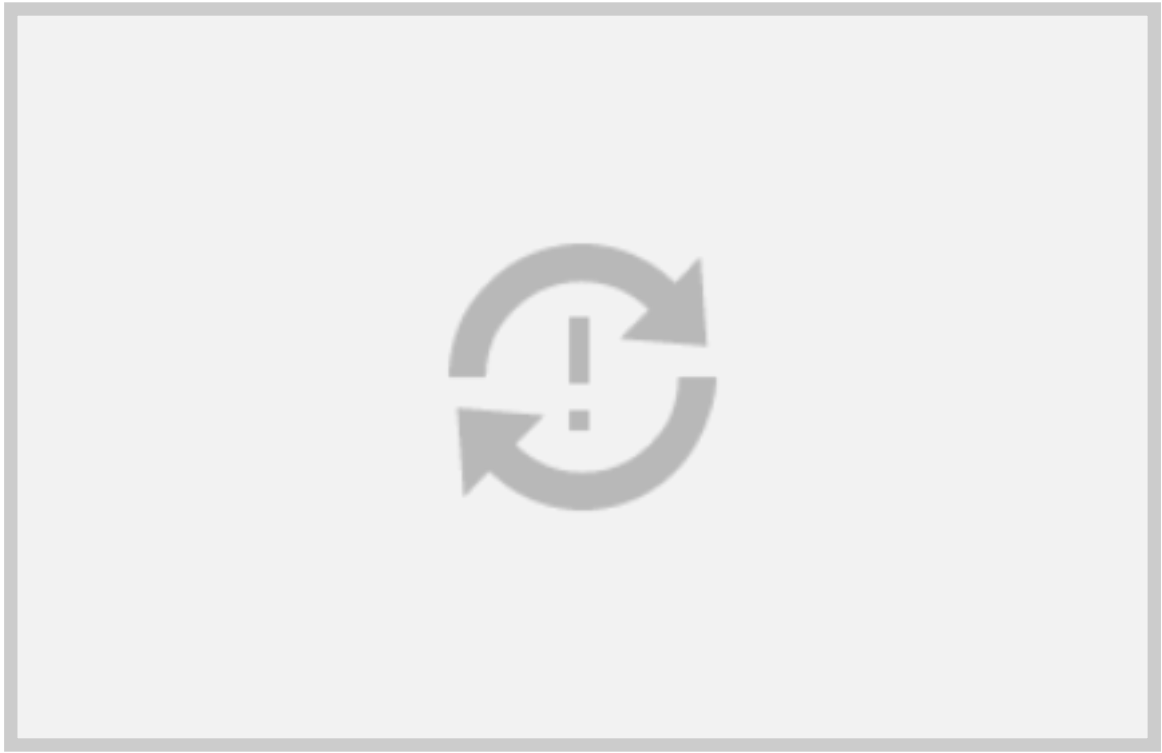
# Use cases for feedforward networks

Let's consider 2 (simplified) sample tasks:

1. Text classification
2. Language modeling

State of the art systems use more powerful neural architectures, but simple models are useful to consider!

# Sentiment Features



# Text classification

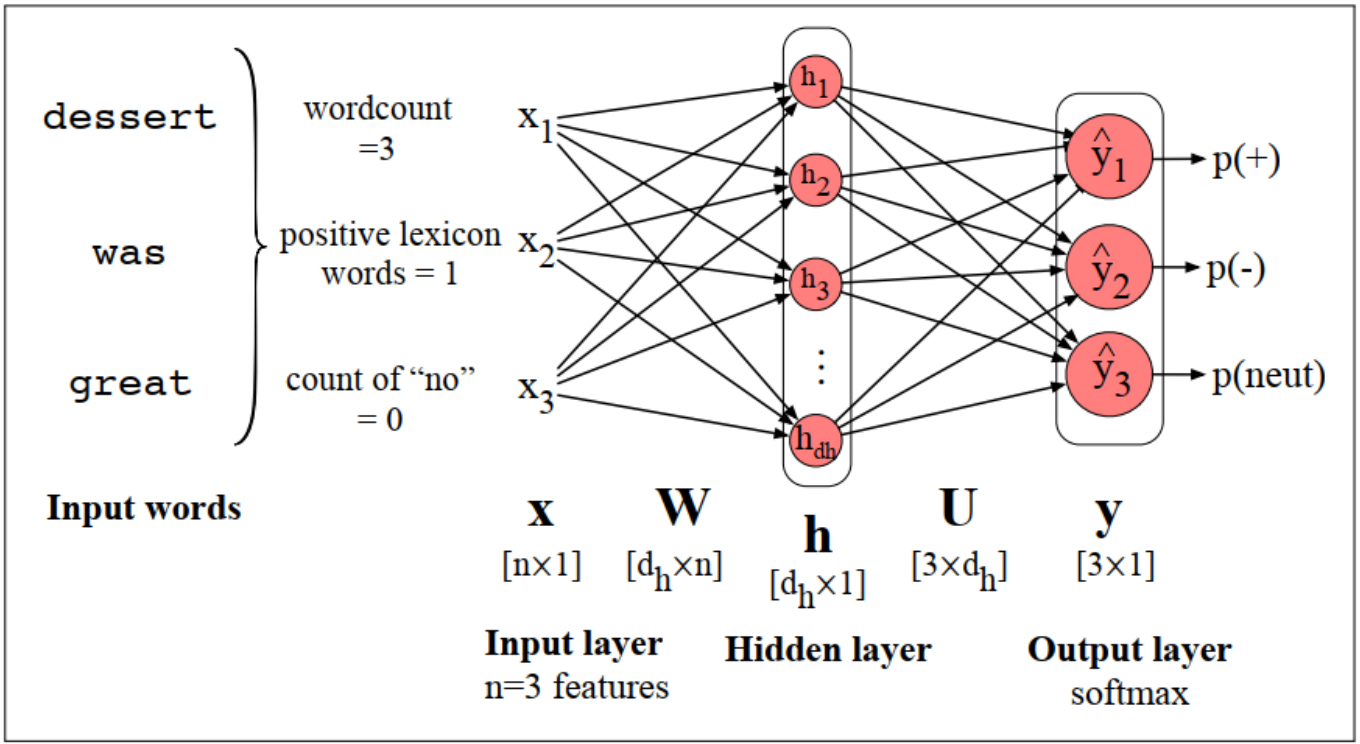



**Figure 7.10** Feedforward network sentiment analysis using traditional hand-built features of the input text.

# Text classification

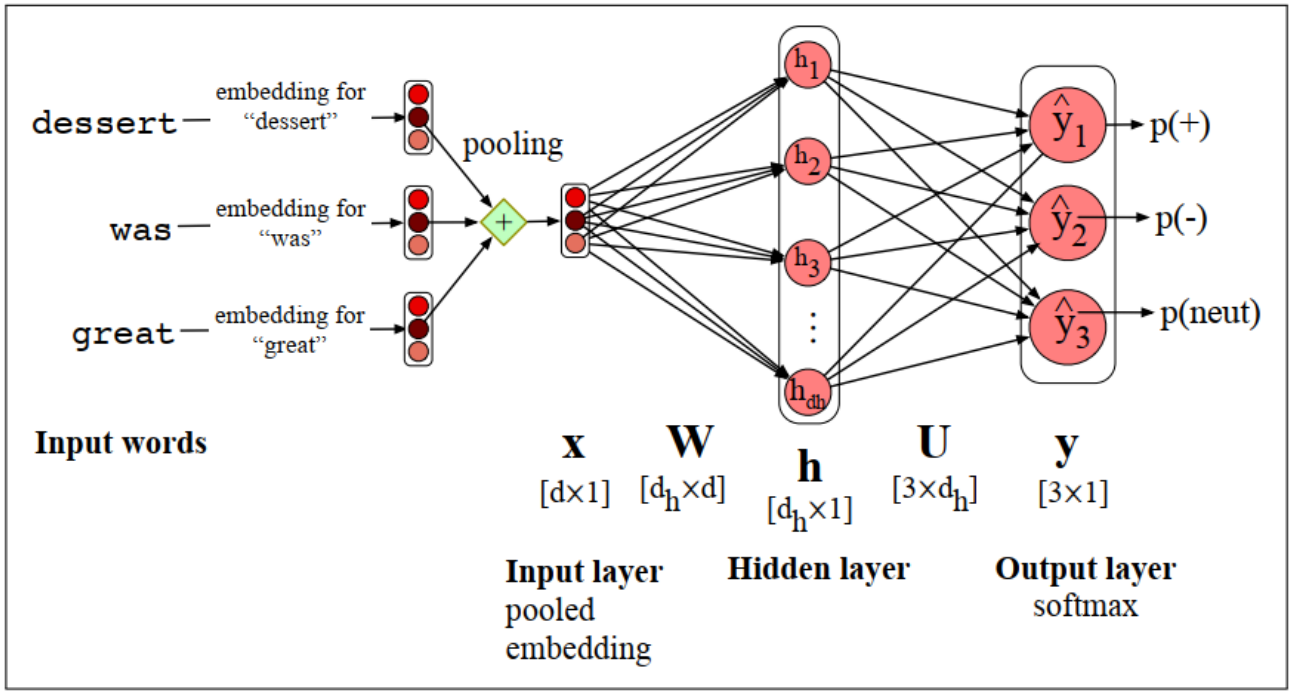



**Figure 7.11** Feedforward network sentiment analysis using a pooled embedding of the input words.

# Neural Language Models (LMs)

**Language Modeling**: Calculating the probability of the next word in a sequence given some history.

- We've seen N-gram based LMs
- But neural network LMs far outperform n-gram language models

State-of-the-art neural LMs are based on more powerful neural network technology like Transformers

But **simple feedforward LMs** can do almost as well!

A feedforward neural language model (LM) is a feedforward network that takes as input at time t a representation of some number of previous words (wt-1;wt-2,etc.) and outputs a probability distribution over possible next words.

Neural language models represent words in this prior context by their embeddings, rather than just by their word identity as used in n-gram language models.

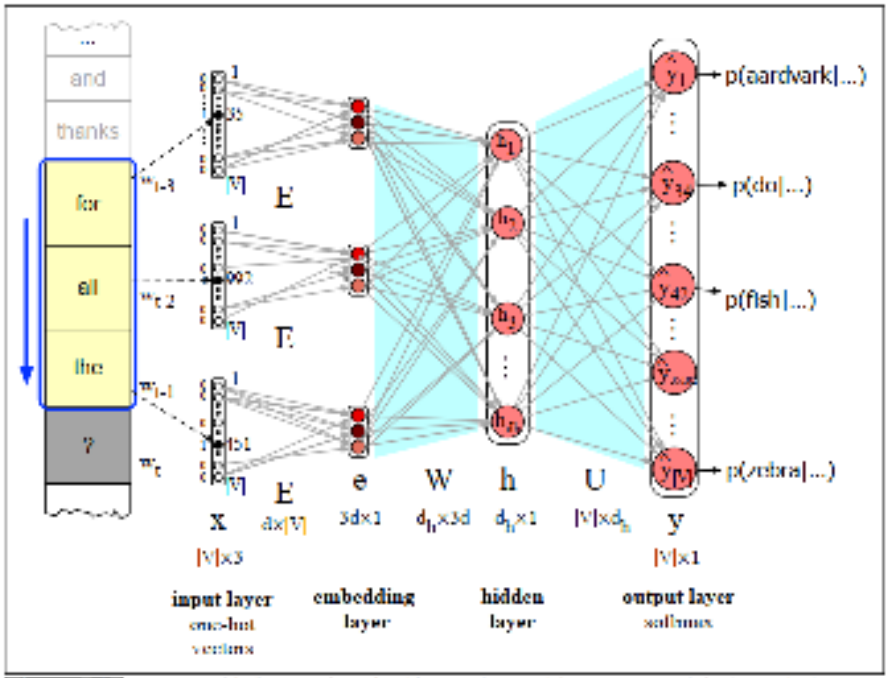



**Figure 7.17** Forward inference in a feedforward neural language model. At each timestep $t$ the network computes a $d$-dimensional embedding for each context word (by multiplying a one-hot vector by the embedding matrix $\mathbf{E}$), and concatenates the 3 resulting embeddings to get the embedding layer $\mathbf{e}$. The embedding vector $\mathbf{e}$ is multiplied by a weight matrix $\mathbf{W}$ and then an activation function is applied element-wise to produce the hidden layer $\mathbf{h}$, which is then multiplied by another weight matrix $\mathbf{U}$. Finally, a softmax output layer predicts at each node $i$ the probability that the next word $w_t$ will be vocabulary word $V_i$.

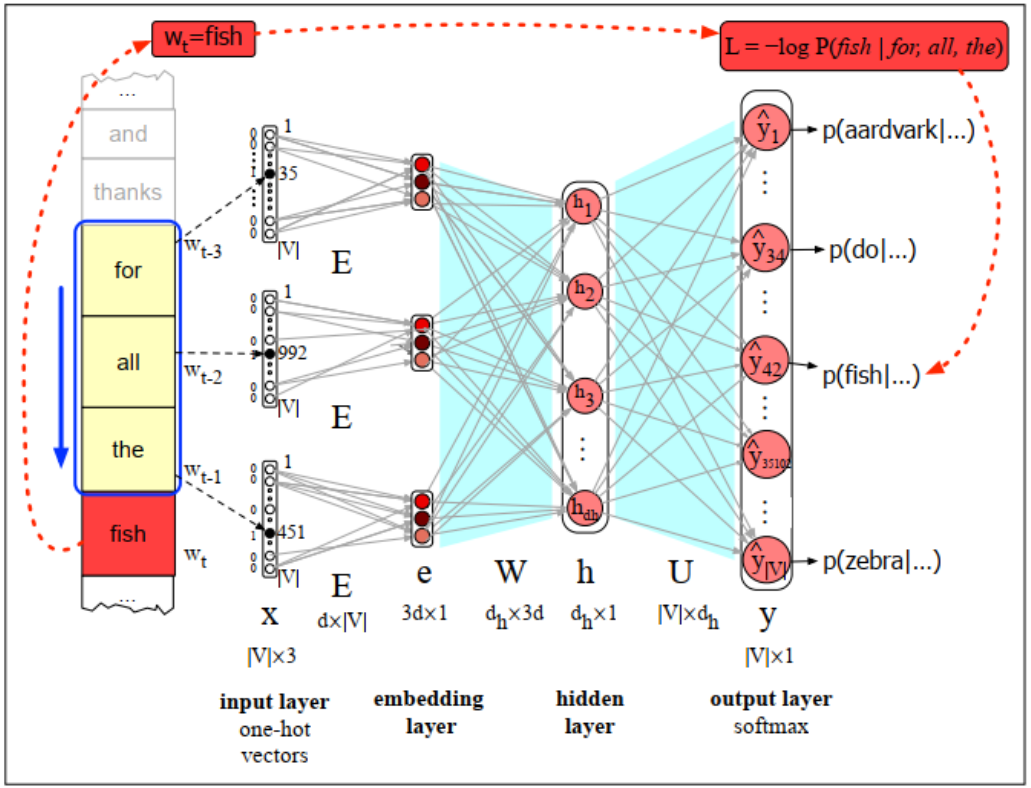



**Figure 7.18** Learning all the way back to embeddings. Again, the embedding matrix **E** is shared among the 3 context words.

Using embeddings allows neural language models to generalize better to unseen data.

For example, suppose we've seen this sentence in training:
"I have to make sure that the cat gets fed."

Our test set has
"I forgot to make sure that the dog gets …."

N-gram model will fail

But a neural LM, knowing that "cat" and "dog" have similar embeddings, will be able to generalize from the "cat" context to assign a high enough probability to "fed" even after seeing "dog".

# Modeling Time in Neural Networks

- Language is inherently temporal

Yet the simple NLP classifiers we've seen (for example for sentiment analysis) mostly ignore time

- (Feedforward neural LMs use a "moving window" approach to time.)

Here we introduce a deep learning architecture with a different way of representing time

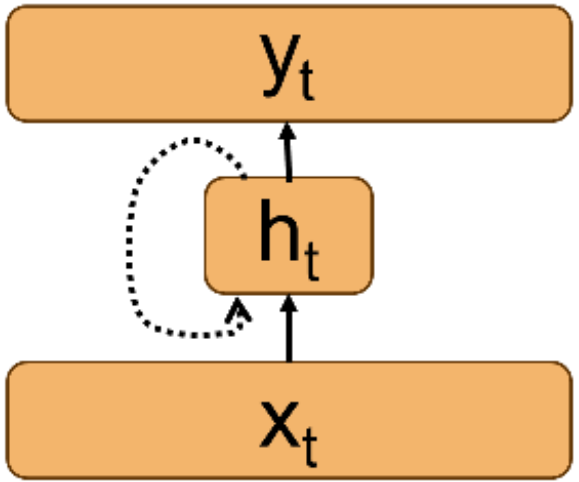
- RNNs and their variants like LSTMs

# Recurrent Neural Networks (RNNs)

Any network that contains a cycle within its network connections.

The value of some unit is directly, or indirectly, dependent on its own earlier outputs as an input.

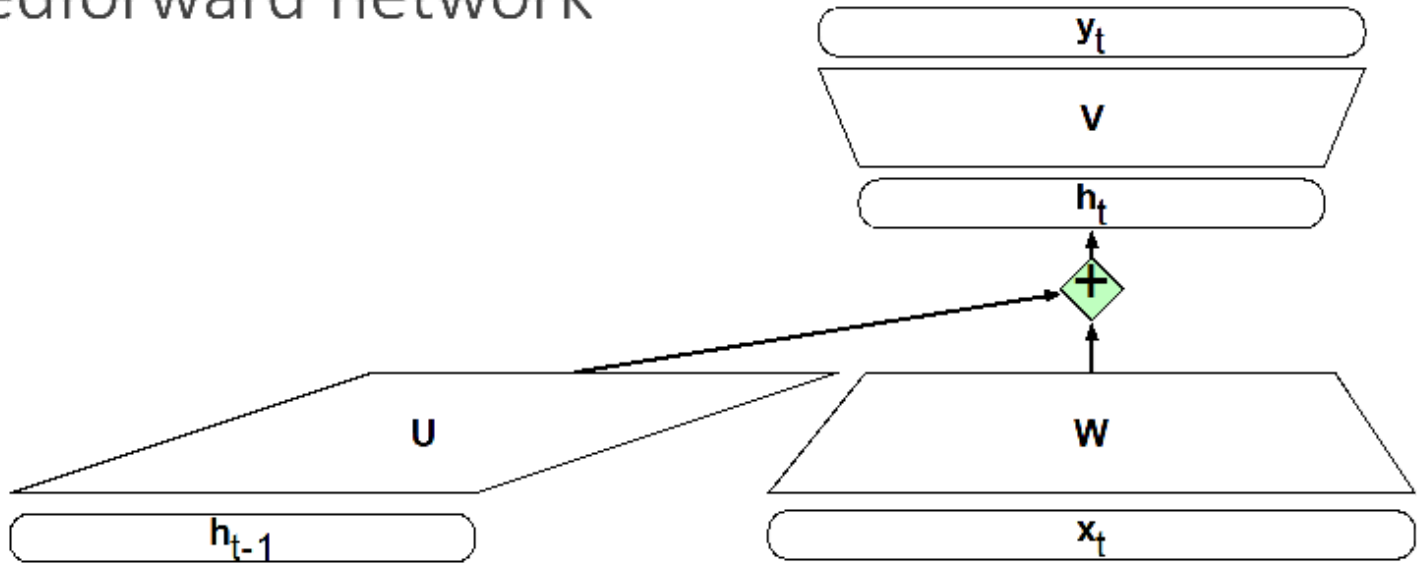
# Simple Recurrent Nets (Elman nets)




The hidden layer has a recurrence as part of its input

The activation value $h_t$ depends on $x_t$ but also $h_{t-1}$!

# Simple recurrent neural network illustrated as a feedforward network




$$h_t = g(Uh_{t-1} + Wx_t)$$

$$y_t = \text{softmax}(Vh_t)$$

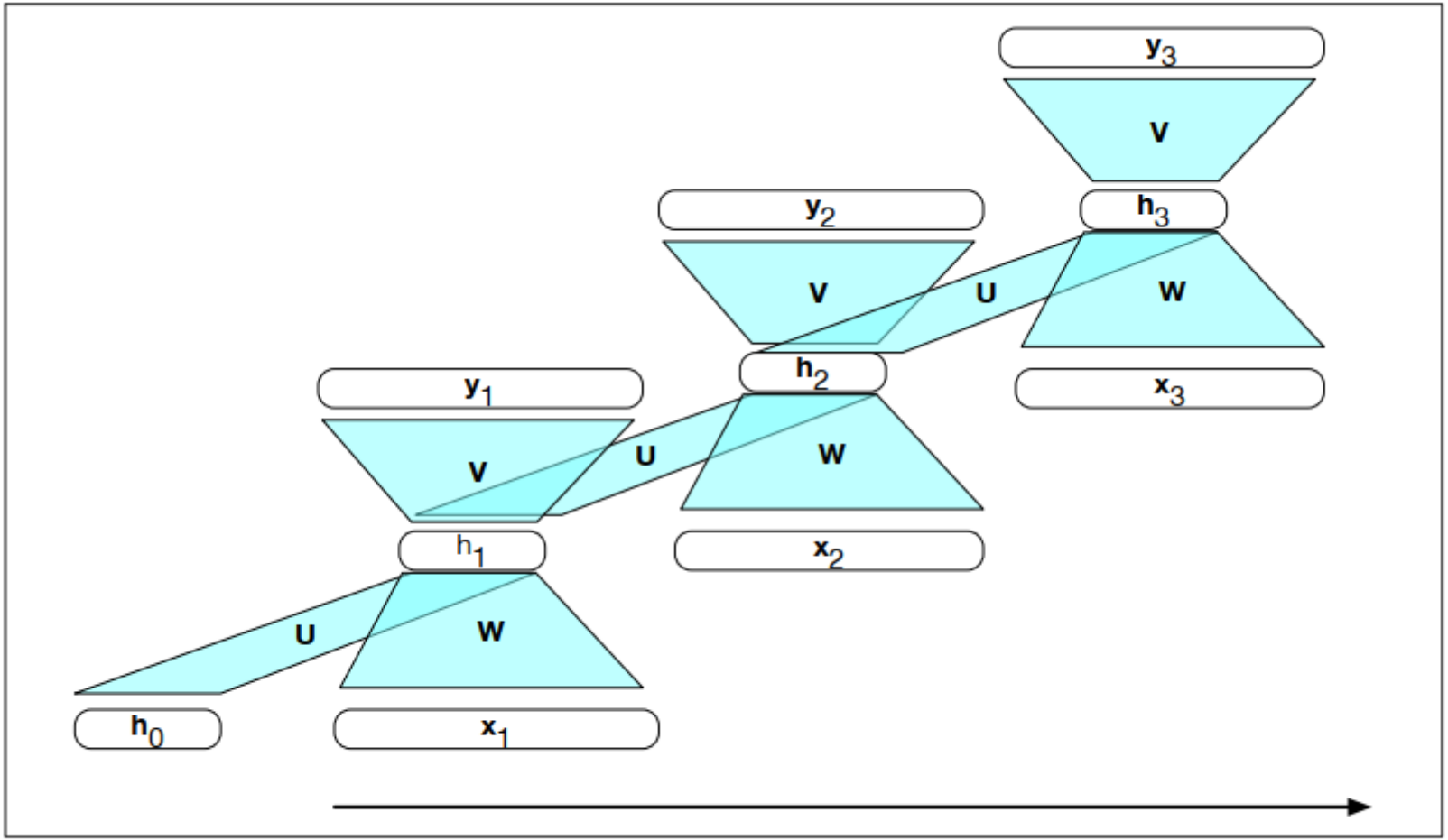
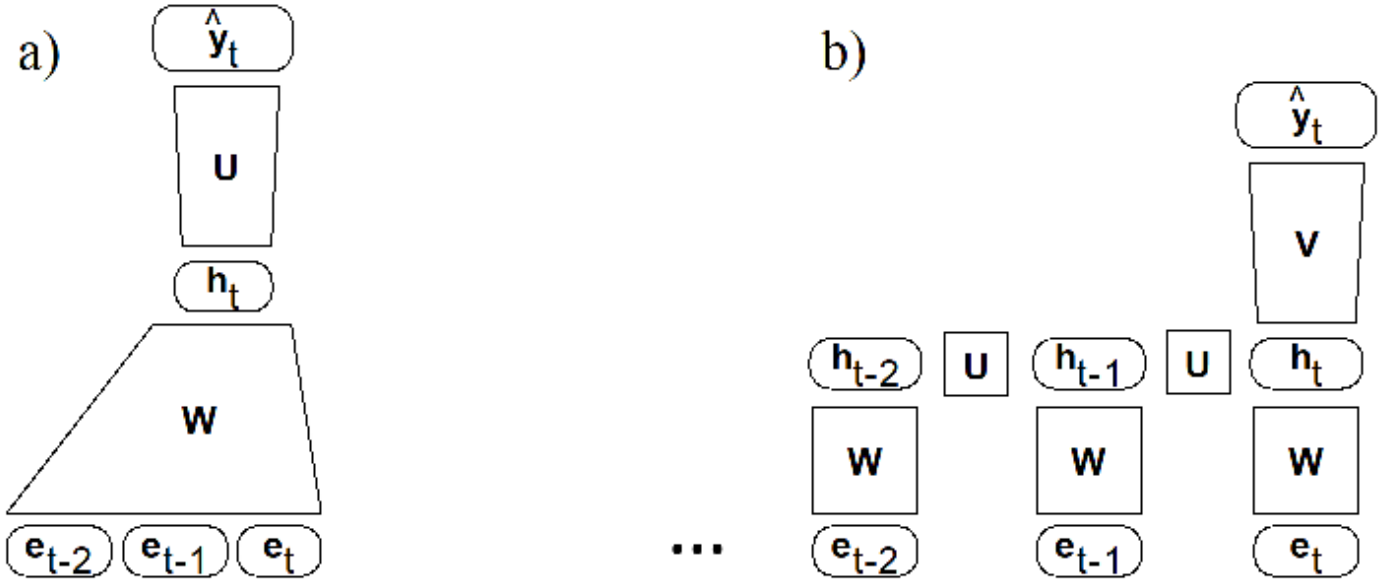



**Figure 8.4** A simple recurrent neural network shown unrolled in time. Network layers are recalculated for each time step, while the weights **U**, **V** and **W** are shared across all time steps.

# FFN LMs vs RNN LMs




FFN RNN

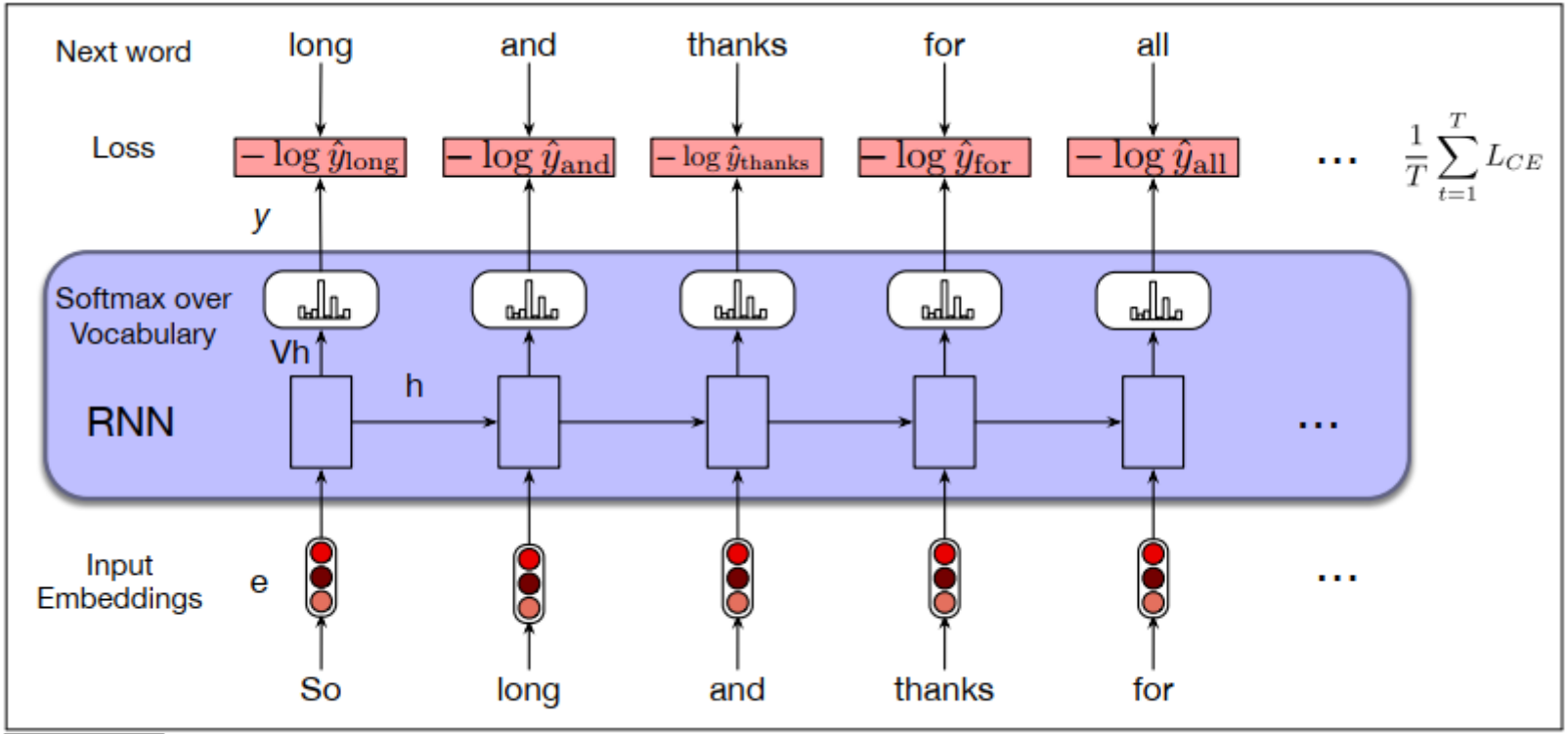



**Figure 8.6** Training RNNs as language models.

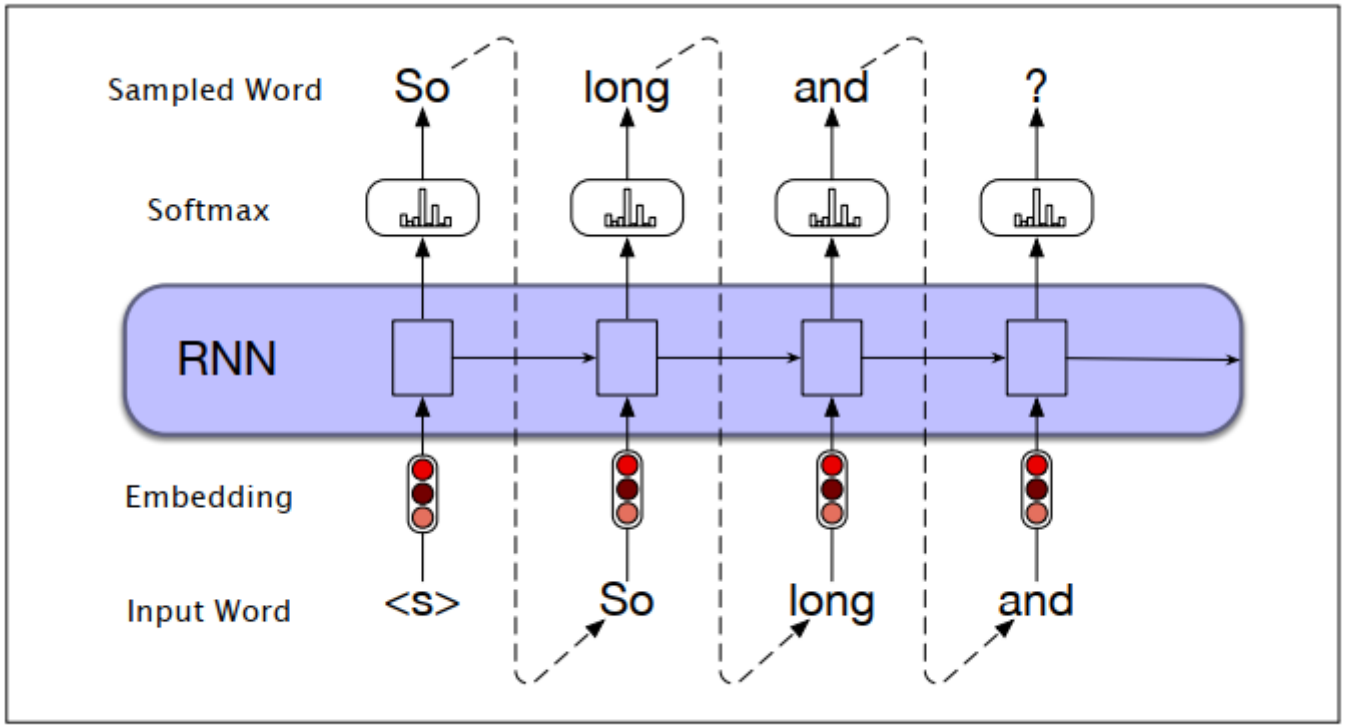



**Figure 8.9** Autoregressive generation with an RNN-based neural language model.

# Teacher forcing

We always give the model the correct history to predict the next word (rather than feeding the model the possible buggy guess from the prior time step).

This is called **teacher forcing** (in training we **force** the context to be correct based on the gold words)

What teacher forcing looks like:

- At word position *t*
- the model takes as input the correct word *wt* together with *ht*−1, computes a probability distribution over possible next words
- That gives loss for the next token *wt*+1
- Then we move on to next word, ignore what the model predicted for the next word and instead use the correct word *wt*+1 along with the prior history encoded to estimate the probability of token *wt*+2.

# RNNs for sequence labeling

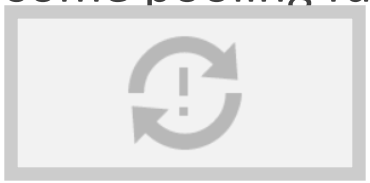
Assign a label to each element of a sequence

Part-of-speech tagging

# RNNs for sequence classification

Text classification

Instead of taking the last state, could use some pooling function of all the output states, like **mean pooling**



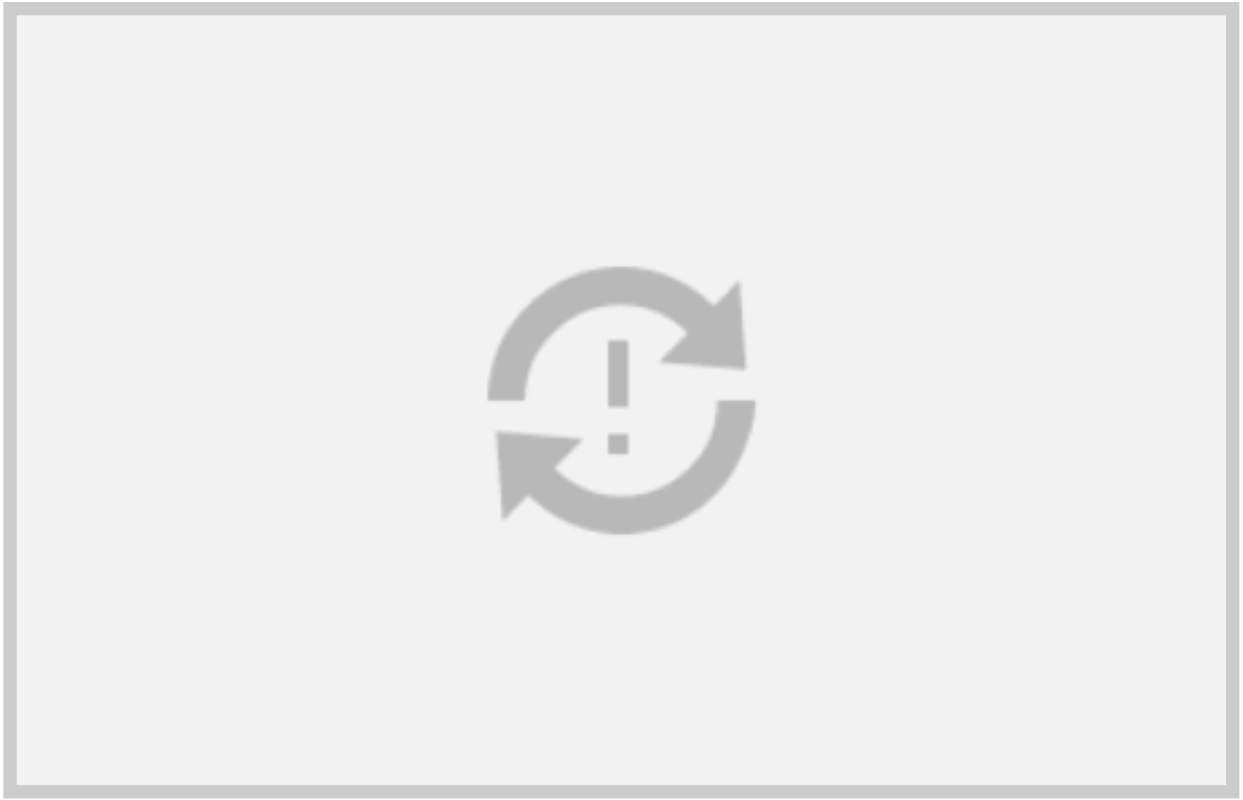
# Bidirectional RNNs for classification

# The LSTM: Long short-term memory network

LSTMs divide the context management problem into two subproblems:

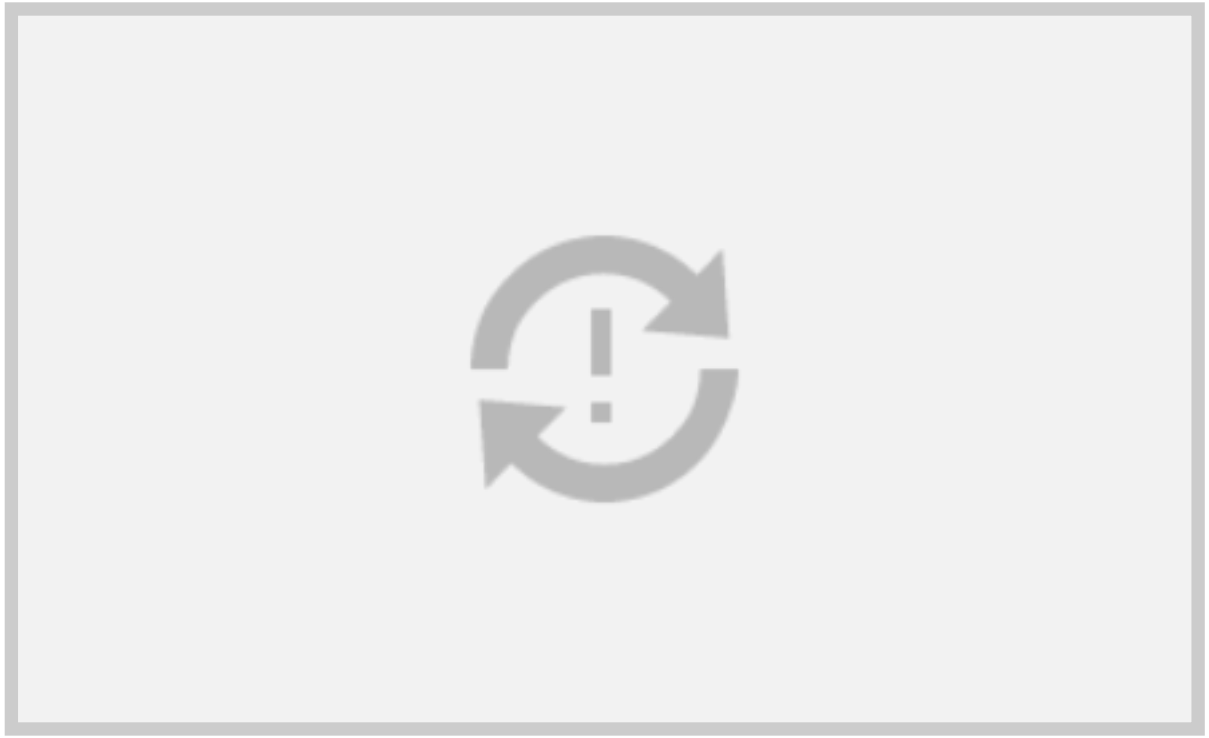
- removing information no longer needed from the context,
- adding information likely to be needed for later decision making
- LSTMs add:
  - explicit context layer
  - Neural circuits with **gates** to control information flow

# Units



FFN SRN LST M

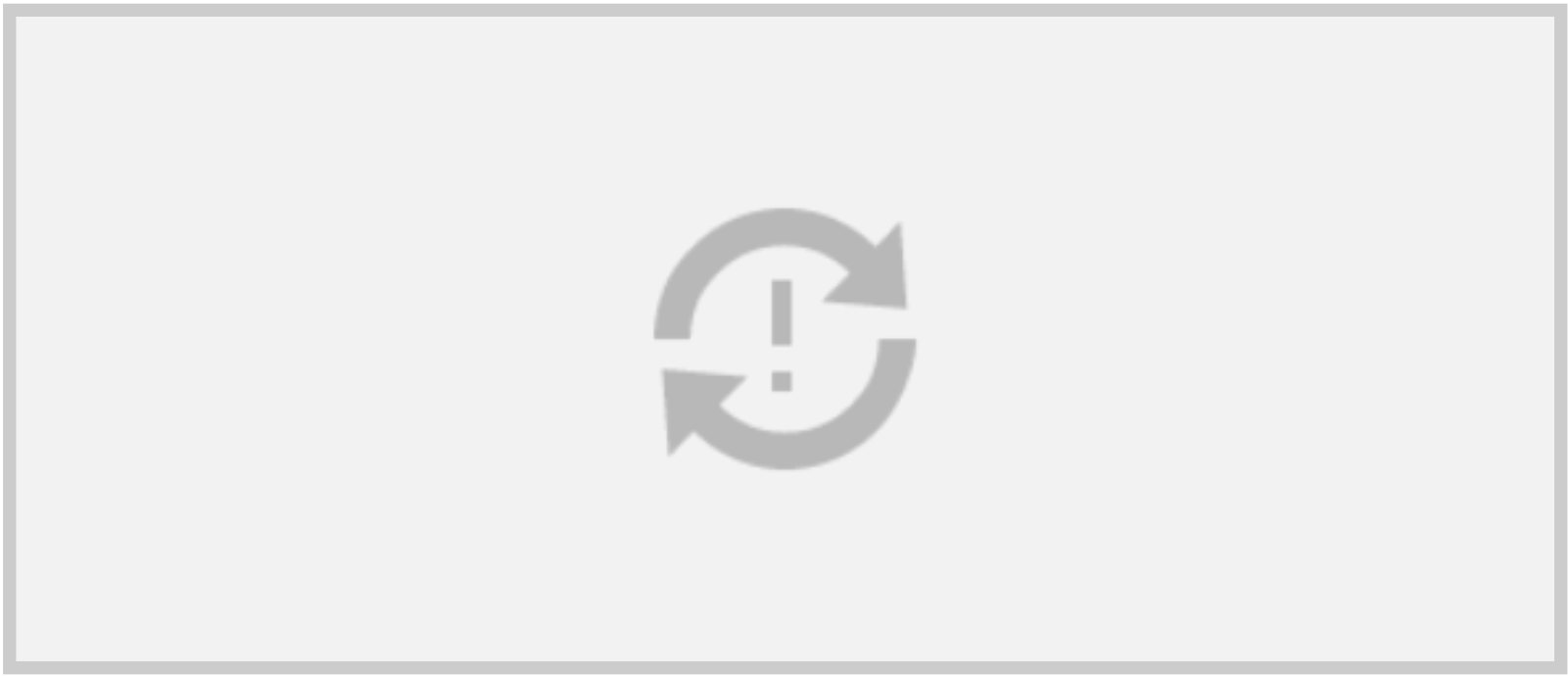
# Four architectures for NLP tasks with RNNs



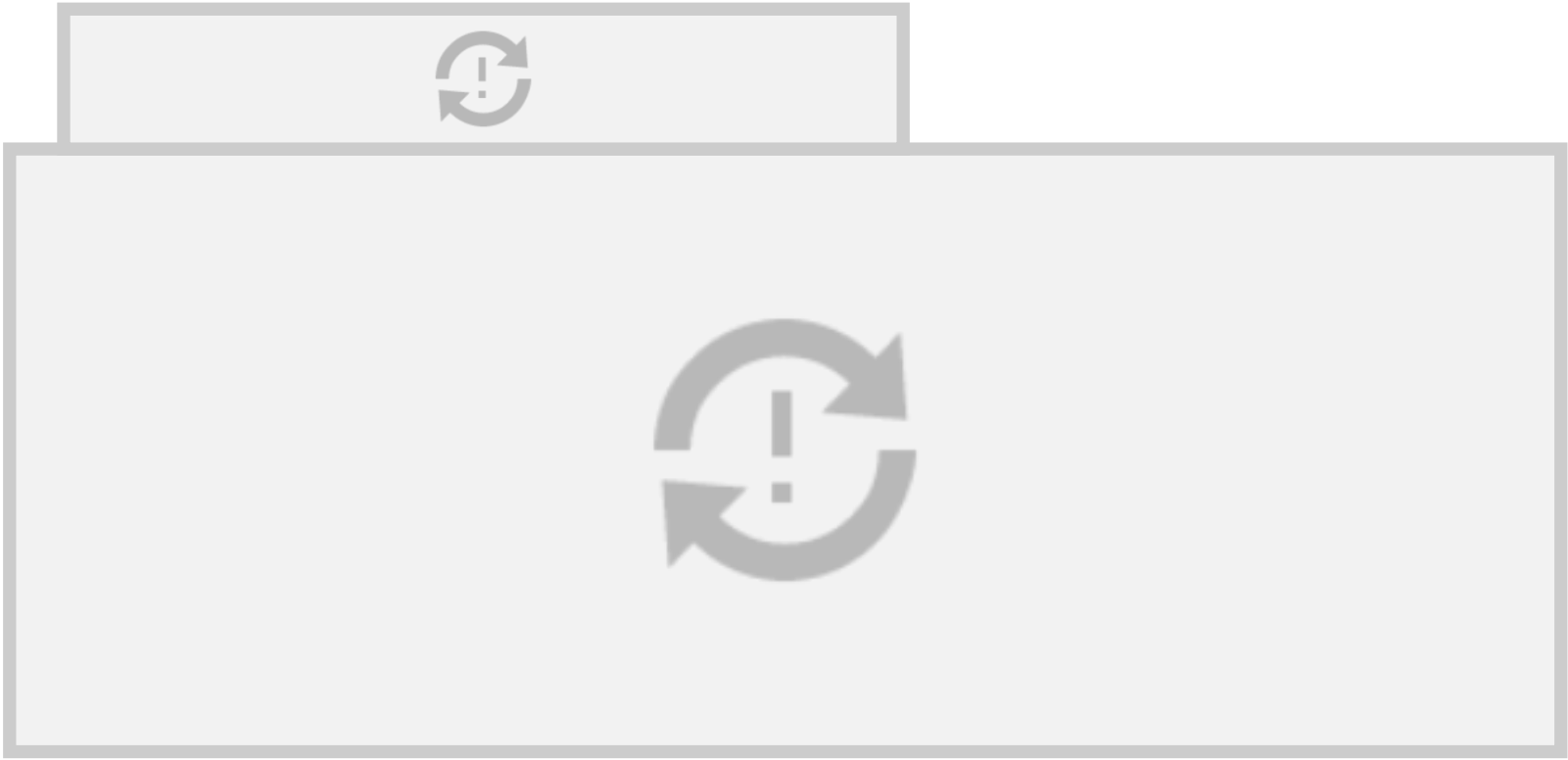
# 3 components of an encoder-decoder

1. An encoder that accepts an input sequence, $x_{1:n}$, and generates a corresponding sequence of contextualized representations, $h_{1:n}$.
2. A context vector, $c$, which is a function of $h_{1:n}$, and conveys the essence of the input to the decoder.
3. A decoder, which accepts $c$ as input and generates an arbitrary length sequence of hidden states $h_{1:m}$, from which a corresponding sequence of output states $y_{1:m}$, can be obtained

# Encoder-decoder simplified



# Encoder-decoder showing context



# Problem with passing context c only from end

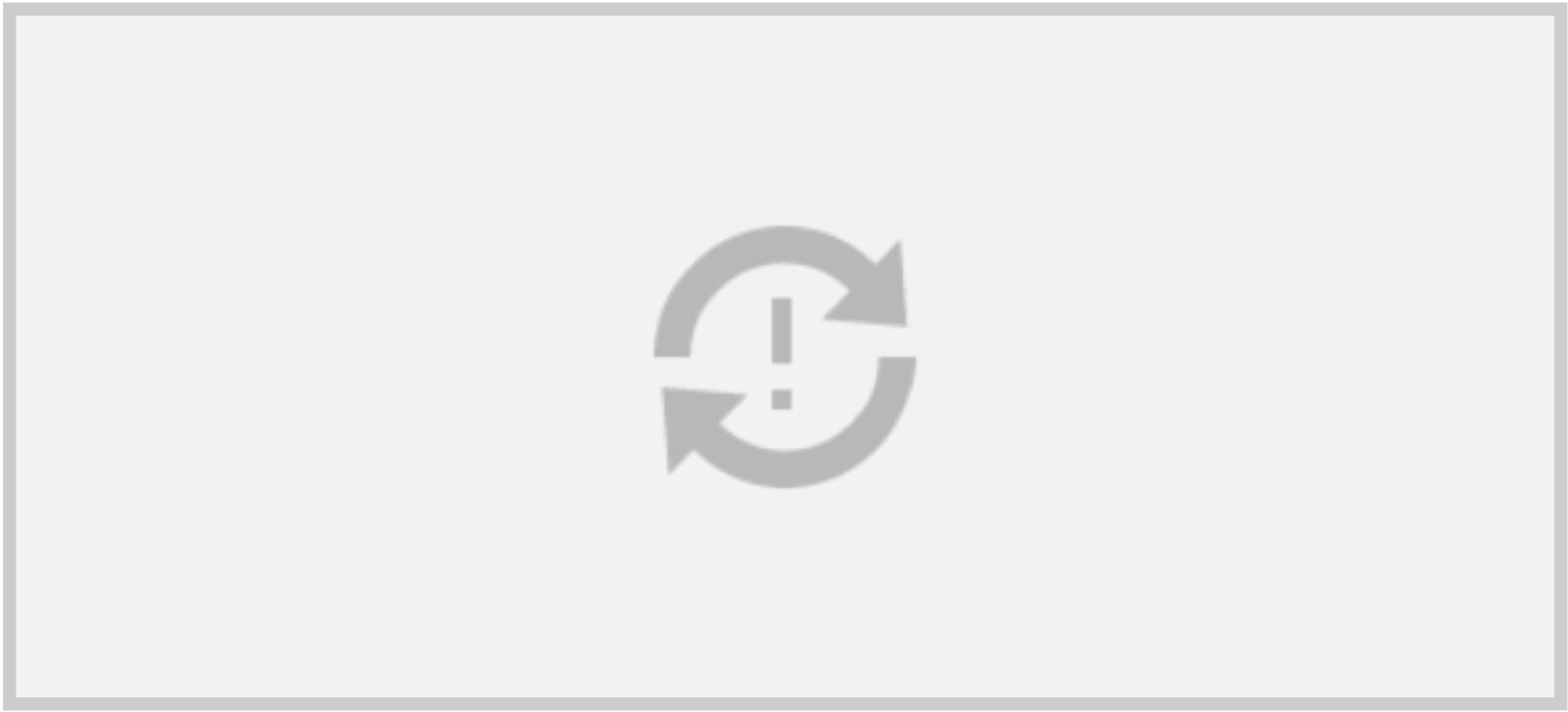
Requiring the context *c* to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

# Solution: attention

instead of being taken from the last hidden state, the context it’s a weighted average of all the hidden states of the decoder.

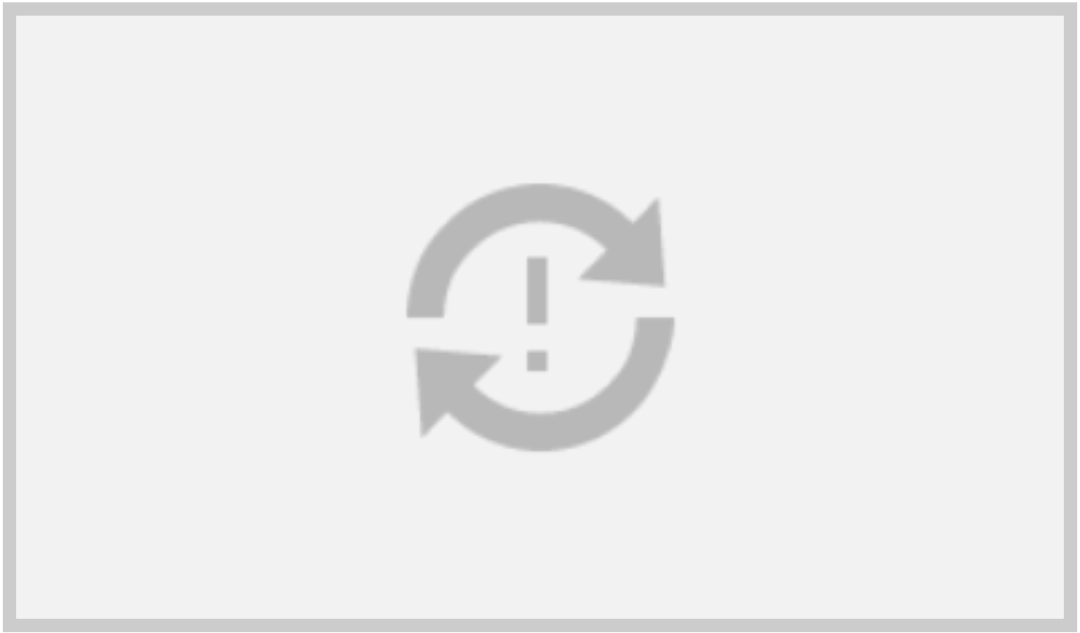
this weighted average is also informed by part of the decoder state as well, the state of the decoder right before the current token *i*.

# Encoder-decoder with attention, focusing on the computation of c



# LLMs are built out of transformers

Transformer: a specific kind of network architecture, like a fancier feedforward network, but based on attention



# Problem with static embeddings (word2vec)

They are static!  The embedding for a word doesn't reflect how its meaning changes in context.

```
The chicken didn't cross the road because it was too tired
```

What is the meaning represented in the static embedding for "it"?

# Contextual Embeddings

- Intuition: a representation of meaning of a word should be different in different contexts!
- **Contextual Embedding**: each word has a different vector that expresses different meanings depending on the surrounding words
- How to compute contextual embeddings?
  - **Attention**

# Contextual Embeddings

`The chicken didn't cross the road because it`

What should be the properties of "it"?

`The chicken didn't cross the road because it was too` **`tired`**
`The chicken didn't cross the road because it was too` **`wide`**

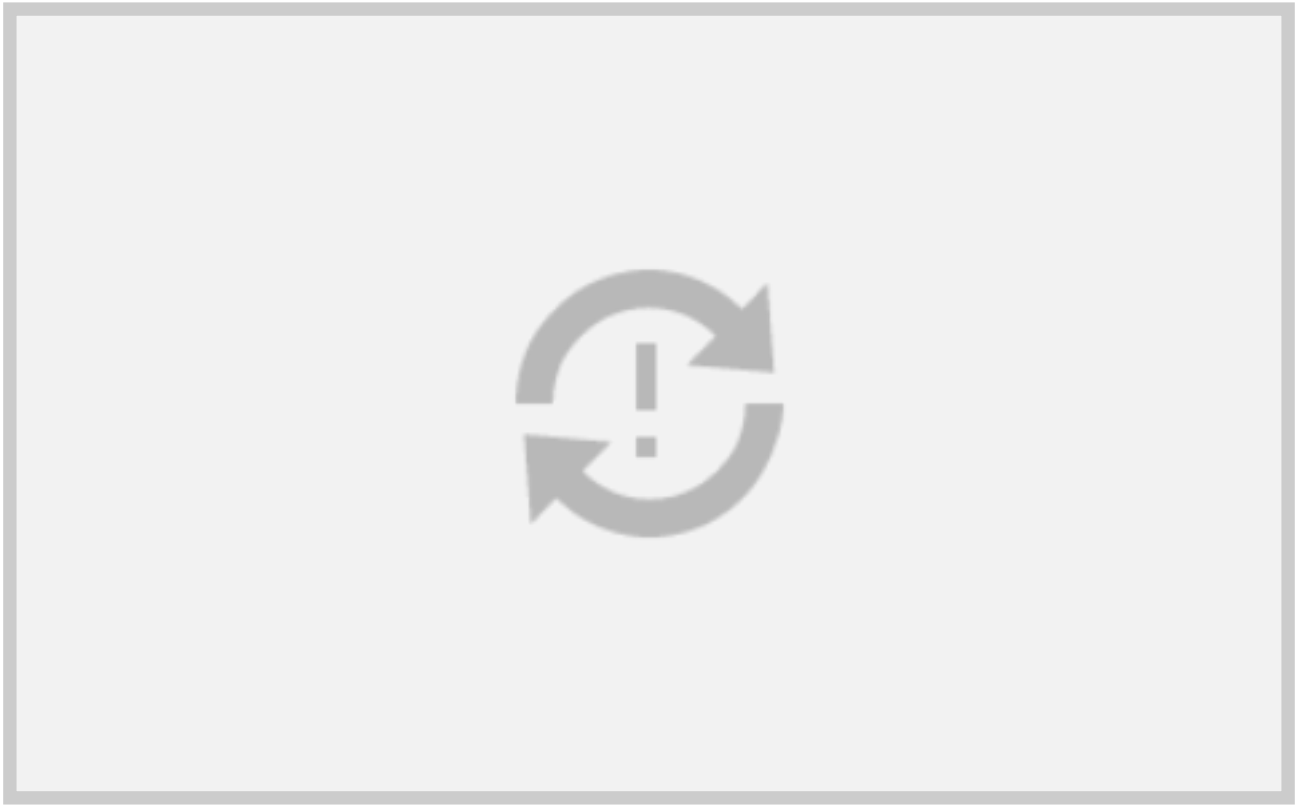
At this point in the sentence, it's probably referring to either the chicken or the street

# Intuition of attention

Build up the contextual embedding from a word by selectively integrating information from all the neighboring words

We say that a word "attends to" some neighboring words more than others

# Intuition of attention:



# Attention definition

A mechanism for helping compute the embedding for a token by selectively attending to and integrating information from surrounding tokens (at the previous layer).

More formally: a method for doing a weighted sum of vectors.

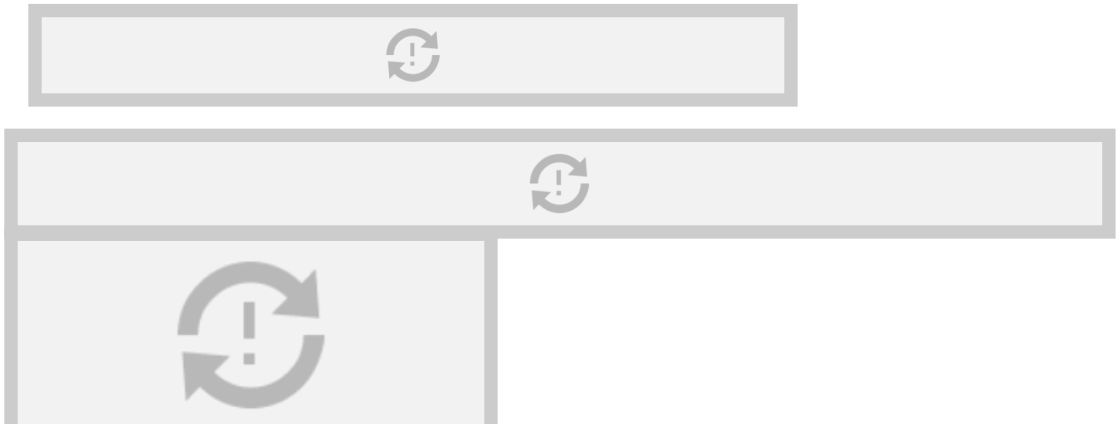
# Attention is left-to-right

Simplified version of attention: a sum of prior words weighted by their similarity with the current word

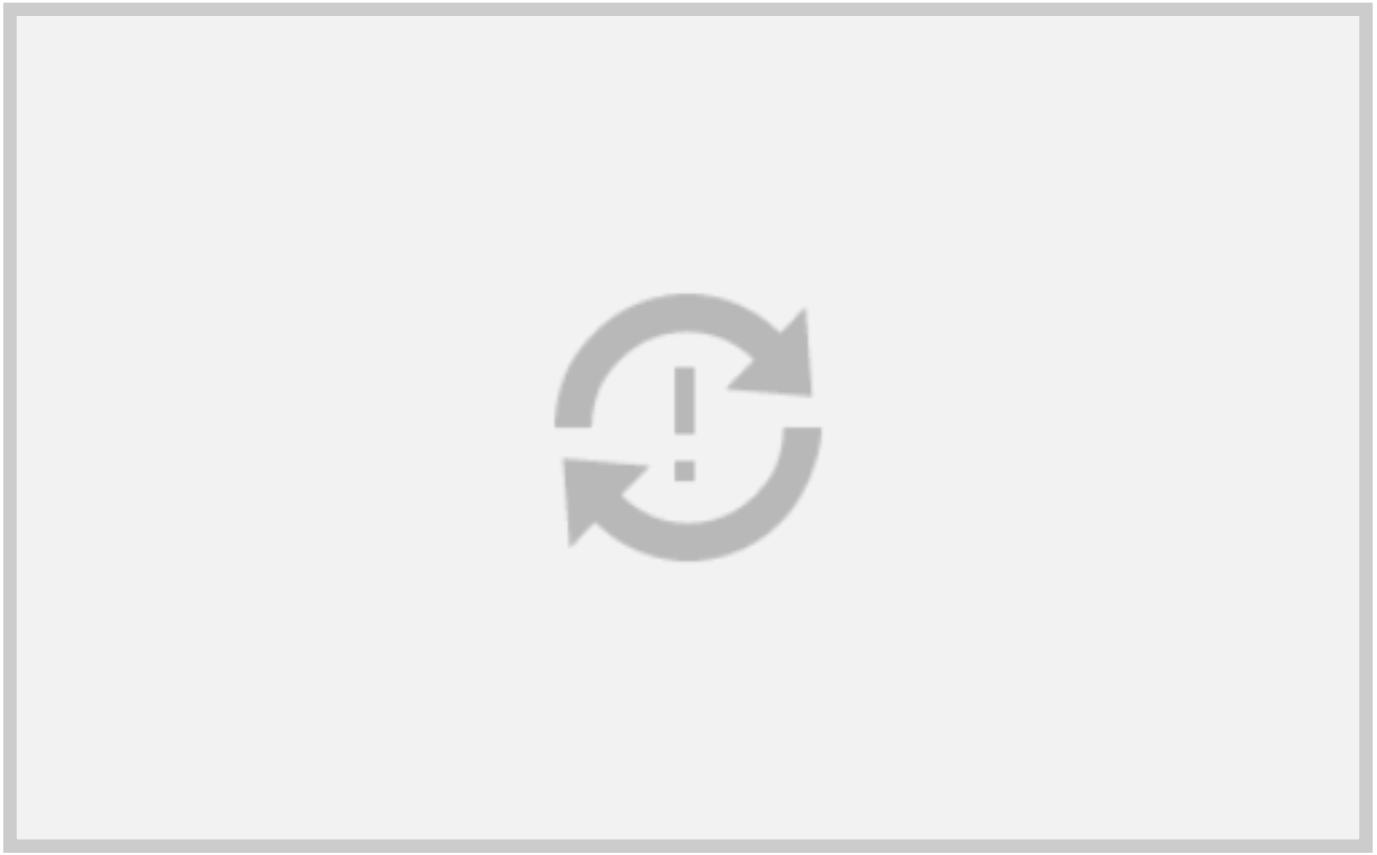
Given a sequence of token embeddings:

$$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \quad \mathbf{x}_5 \quad \mathbf{x}_6 \quad \mathbf{x}_7 \quad \mathbf{x}_i$$

Produce: $\mathbf{a}_i$ = a weighted sum of $\mathbf{x}_1$ through $\mathbf{x}_7$ (and $\mathbf{x}_i$)
Weighted by their similarity to $\mathbf{x}_i$



# Intuition of attention:



**x1 x2 x3 x4 x5 x6**

# An Actual Attention Head: slightly more complicated

High-level idea: instead of using vectors (like $x_i$ and $x_4$) directly, we'll represent 3 separate roles each vector $\mathbf{x}_i$ plays:

- **query**: As *the current element* being compared to the preceding inputs.
- **key**: as *a preceding input* that is being compared to the current element to determine a similarity
- **value**: a value of a preceding element that gets weighted and summed

# An Actual Attention Head: slightly more complicated

We'll use matrices to project each vector $\mathbf{x}_i$ into a representation of its role as query, key, value:

- **query**: $\mathbf{W}^Q$
- **key**: $\mathbf{W}^K$
- **value**: $\mathbf{W}^V$

# An Actual Attention Head: slightly more complicated

Given these 3 representation of $\mathbf{x}_i$

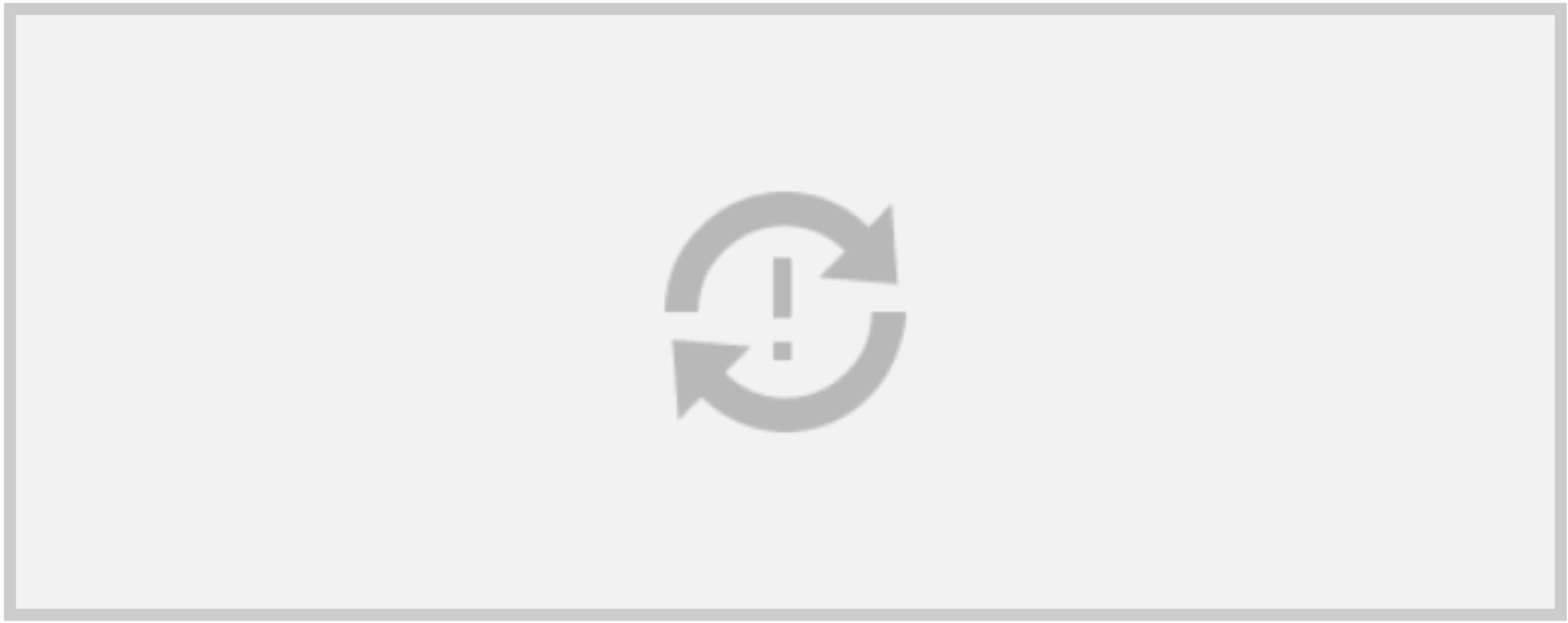
To compute similarity of current element $\mathbf{x}_i$ with some prior element $\mathbf{x}_j$

We'll use dot product between $\mathbf{q}_i$ and $\mathbf{k}_j$.

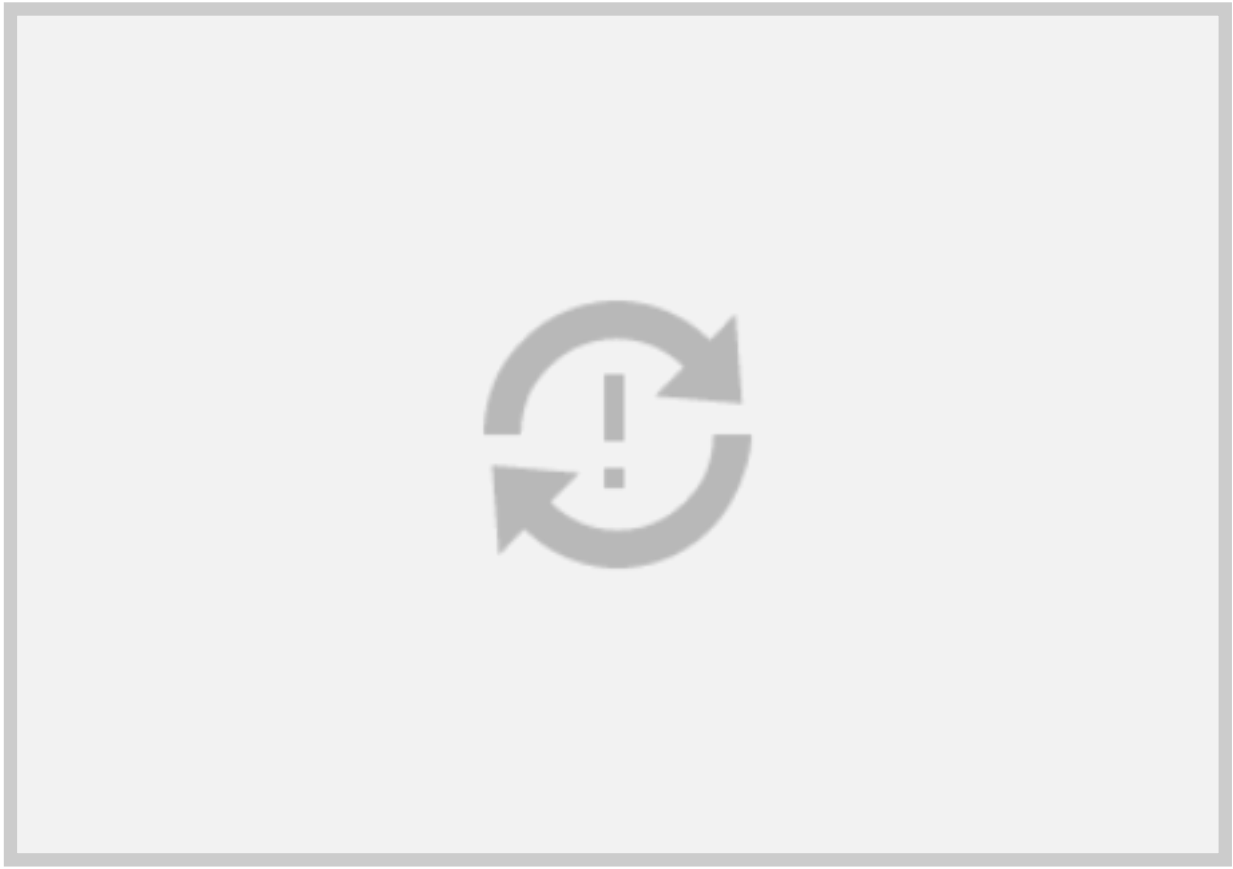
This will give us attention "$\alpha_{ij}$" and using this we calculate $\mathbf{v}_j$

And instead of summing up $\mathbf{x}_j$ , we'll sum up $\mathbf{v}_j$

# Final equations for one attention head



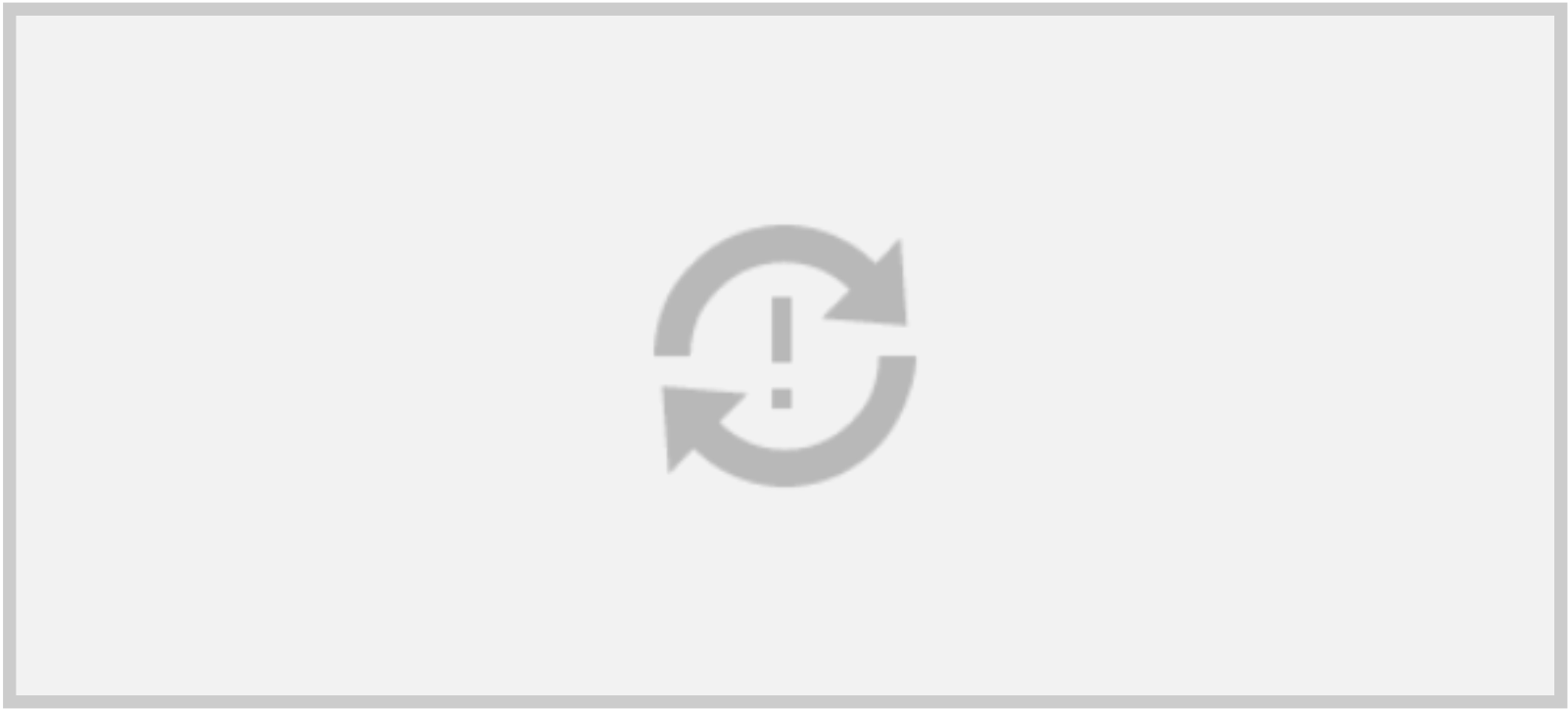
# Calculating the value of a3



# Actual Attention: slightly more complicated

- Instead of one attention head, we'll have lots of them!
- Intuition: each head might be attending to the context for different purposes
  - Different linguistic relationships or patterns in the context

# Multi-head attention



# Summary

Attention is a method for enriching the representation of a token by incorporating contextual information

The result: the embedding for each word will be different in different contexts!

Contextual embeddings: a representation of word meaning in its context.

We'll see in the next lecture that attention can also be viewed as a way to move information from one token to another.