# Machine Learning (CE 40717)

Fall 2024

Ali Sharifi-Zarchi

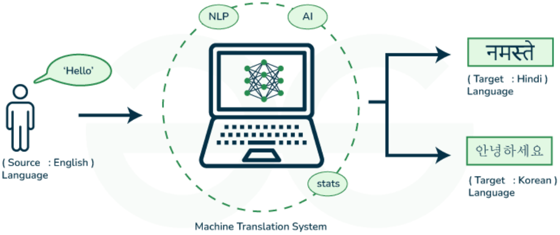
CE Department
Sharif University of Technology

November 9, 2024

# Natural Language Processing

- Language is central to human interaction; many of our daily activities revolve around text and language.
- Natural Language Processing (NLP) enables computers to understand and generate human language.

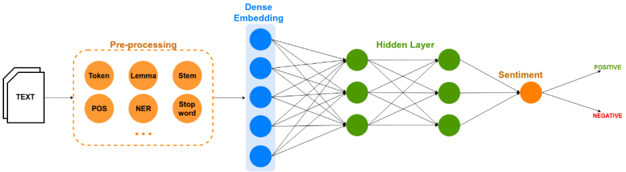
# Translation




- NLP helps translate text from one language to another.

Figure adapted from geeksforgeeks.org/machine-translation-of-languages-in-artificial-intelligence/

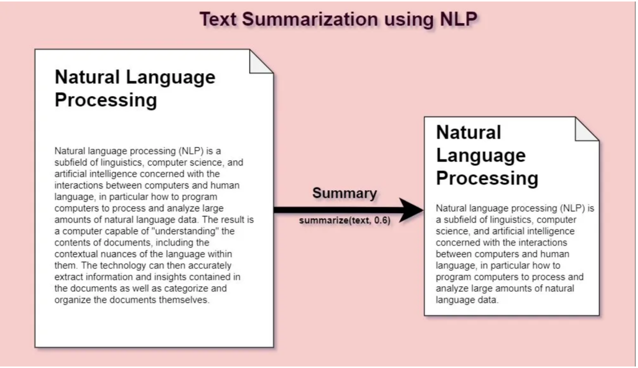
## Sentiment Analysis




- Determines the sentiment (e.g., positive or negative) expressed in a text.

Figure adapted from www.mdpi.com/2079-9292/9/3/483

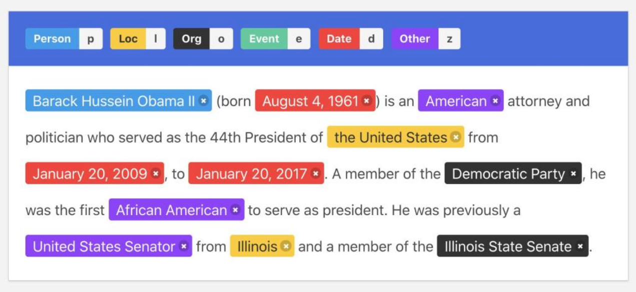
# Text Summarization




- Automatically generates a concise summary of longer text.

Figure adapted from turbolab.in/types-of-text-summarization-extractive-and-abstractive-summarization-basics/

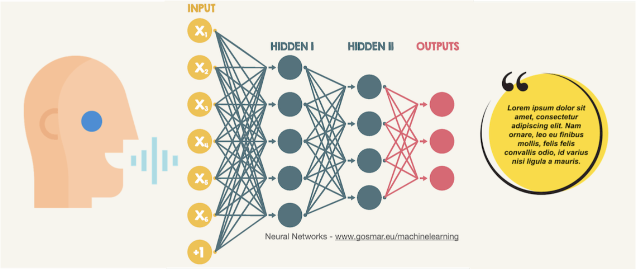
# Named Entity Recognition (NER)




- Identifies and classifies entities (e.g., names, dates) in text.

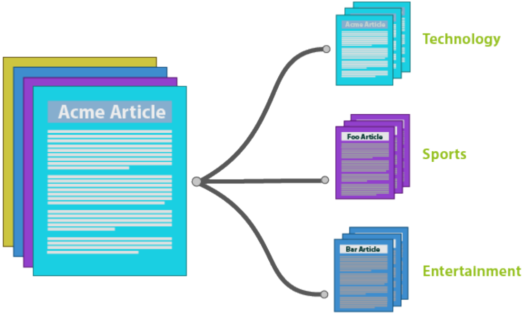
Figure adapted from analyticsvidhya.com/blog/2021/11/a-beginners-introduction-to-ner-named-entity-recognition/

# Speech Recognition




- Converts spoken language into text.

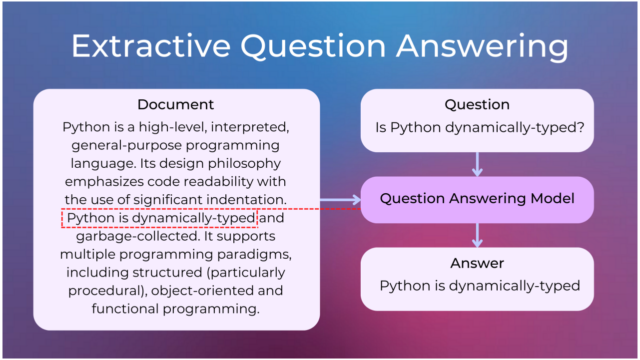
# Text Classification




- Categorizes text into predefined groups or topics.

Figure adapted from towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a

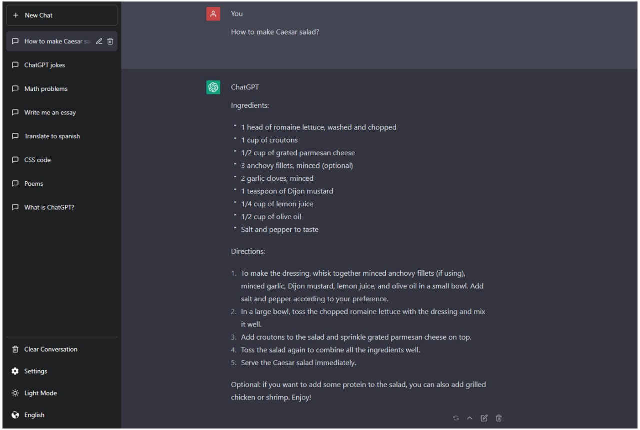
# Question Answering




- Answers questions based on a given text or dataset.

Figure adapted from nlplanet.org/course-practical-nlp/02-practical-nlp-first-tasks/17-question-answering

# Chatbots and Dialogue Systems




- NLP powers chatbots that can interact with users through text or speech.

Figure adapted from chatgpt.org/

## The Importance of Word Representation

- To process text effectively, the first step is to represent words in a way that models can understand.
- We need to transform words into vectors or dense representations to capture their meaning and relationships.
- This is crucial for enabling machines to understand and use language as humans do.

## Motivation

- Traditional models use methods like one-hot encoding, which lacks semantic understanding and cannot capture relationships between words.
- We need better word representations that are both dense and semantic.

## Definition of One-Hot Encoding

- One-hot encoding is a straightforward method for representing categorical data, such as words, as discrete vectors.
- Each word is represented as a binary vector with the same length as the vocabulary size.
- All vector elements are set to 0 except for one position, which is set to 1, identifying the word's unique position in the vocabulary.

# Example of One-Hot Encoding

- For example, given a vocabulary of 5 words:
  - `apple` = [1, 0, 0, 0, 0]
  - `banana` = [0, 1, 0, 0, 0]
  - `cherry` = [0, 0, 1, 0, 0]
  - `date` = [0, 0, 0, 1, 0]
  - `elderberry` = [0, 0, 0, 0, 1]
- The length of the one-hot vector depends on the number of unique words in the vocabulary.

# Strengths and Limitations of One-Hot Encoding

- **Strengths:**
  - One-hot encoding is a simple and intuitive representation that can be effective in certain models, especially smaller neural networks.
  - It requires minimal computation and works well for small vocabularies or categorical features in simpler tasks.
- **Limitations:**
  - One-hot encoding does not capture any semantic relationships between words.
  - The vectors are sparse, containing mostly zeros, which is inefficient for large vocabularies.
  - Similar words (like `hotel` and `motel`) appear completely unrelated in this representation.

## Example: Similar Words, 0 Cosine Similarity

- Consider the following one-hot vectors:
  - `hotel` = [0, 0, 0, 1, 0]
  - `motel` = [0, 0, 0, 0, 1]
- Even though `hotel` and `motel` are semantically similar, their cosine similarity is 0 because their vectors are orthogonal.
- **Cosine Similarity:**

$$\cos(\theta) = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| \|\mathbf{v}_2\|}$$

- In this case, the dot product of the one-hot vectors is zero, leading to a cosine similarity of zero.

## Conclusion: Why Move Beyond One-Hot?

- While one-hot encoding is a simple and effective method for certain applications, it fails to capture word meanings or relationships.
- More advanced methods, such as word embeddings, address these limitations by representing words in a dense, meaningful vector space.

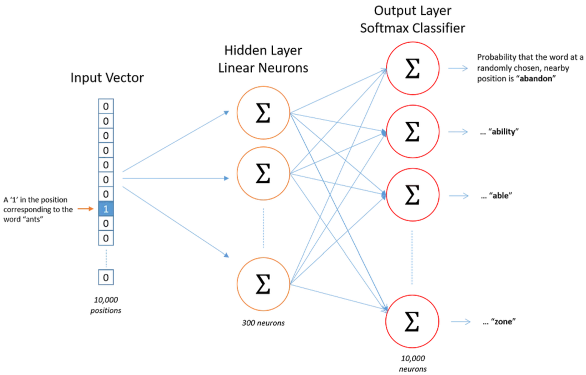
## Why Learn Word Vectors?

- To process text data, we need to represent words in a form that a machine can understand—numerical vectors.
- Word2Vec uses a neural network to learn word embeddings that capture semantic similarities.
- These embeddings allow words with similar meanings to be represented by vectors close to each other in a high-dimensional space.

# Word2Vec as a Neural Network

- Word2Vec operates like a shallow neural network, with an input, hidden, and output layer.
- It takes in a target word and learns to predict either the surrounding context words or the target word from a set of context words.
- Through training, the network adjusts weights to create meaningful vector representations of words.

# Word2Vec as a Neural Network




Word2Vec as a two layer neural network

Figure adapted from medium.com/@manansuri/a-dummys-guide-to-word2vec-456444f3c673

# Expected Outcome of Word2Vec

- Word2Vec aims to create a vector space where words with similar meanings or contexts are located close to each other.
- **Expected Result:** Semantically related words—such as “king” and “queen” or “dog” and “puppy”—should have similar vector representations.
- This proximity allows for various NLP tasks, such as:
  - **Synonym detection:** Identifying words with similar meanings.
  - **Analogy tasks:** Solving analogies by vector arithmetic (e.g., “king” - “man” + “woman” “queen”).
  - **Clustering of concepts:** Grouping related concepts together in the embedding space.
- By representing words in this way, Word2Vec enables models to make use of semantic relationships between words.

# Expected Outcome of Word2Vec


Hypothetical features to understand word embeddings

Figure adapted from medium.com/@manansuri/a-dummys-guide-to-word2vec-456444f3c673

# Word2Vec: Contextual Word Representation

- The core idea is based on distributional semantics: "You shall know a word by the company it keeps."
- Word2Vec uses two main algorithms for learning word vectors:
  - Continuous Bag of Words (CBOW)
  - Skip-gram Model

# CBOW: How It Works

- CBOW predicts the target word using the context (surrounding words) in a fixed window.
- For each word in the corpus, CBOW takes a set of context words and predicts the center word.
- Example: Given the context words {“the”, “brown”, “fox”, “over”}, CBOW predicts the center word “jumps.”
- CBOW tends to perform better on smaller datasets and is computationally more efficient.

## Skip-gram: How It Works

- Skip-gram is the reverse of CBOW. It predicts the surrounding context words given a target word.
- For each word $w_t$, the model predicts the words in the window of size $m$ around it (e.g., words $w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$).
- Example: If the center word is "jumps," Skip-gram predicts the context words "the," "brown," "fox," and "over."
- Skip-gram is better suited for larger datasets and can capture rare words more effectively.

# Skip-gram Example

| Window Size | Text | Skip-grams |
|---|---|---|
| 2 | [ The **wide** road shimmered ] in the hot sun. | wide, the<br>wide, road<br>wide, shimmered |
| | The [ wide road **shimmered** in the ] hot sun. | shimmered, wide<br>shimmered, road<br>shimmered, in<br>shimmered, the |
| | The wide road shimmered in [ the hot **sun** ]. | sun, the<br>sun, hot |
| 3 | [ The **wide** road shimmered in ] the hot sun. | wide, the<br>wide, road<br>wide, shimmered<br>wide, in |
| | [ The wide road **shimmered** in the hot ] sun. | shimmered, the<br>shimmered, wide<br>shimmered, road<br>shimmered, in<br>shimmered, the<br>shimmered, hot |
| | The wide road shimmered [ in the hot **sun** ]. | sun, in<br>sun, the<br>sun, hot |

Different window sizes and samples drawn from context words and their target

Figure adapted from tensorflow.org/text/tutorials/word2vec

## Skip-gram: Objective Function

- The objective of Skip-gram is to maximize the likelihood of predicting context words $w_o$ given a center word $w_c$.
- The probability of a context word $w_o$ given a center word $w_c$ is defined as:

$$P(w_o|w_c) = \frac{\exp(v_{w_o} \cdot v_{w_c})}{\sum_{w \in V} \exp(v_w \cdot v_{w_c})}$$

- $v_{w_o}$ and $v_{w_c}$ are the word vectors for the context and center words, respectively.

## Skip-gram: Loss Function

- The goal is to minimize the negative log-likelihood over the entire training corpus:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} | w_t)$$

- Here, $T$ is the total number of words, and $m$ is the window size.
- Skip-gram adjusts the word vectors to maximize the probability of observing the context words around the center word.

## Skip-gram: Gradient Calculation

- To update the word vectors during training, we calculate the gradient of the objective function.
- The gradient with respect to the word vector $v_I$ is:

$$\frac{\partial \log P(w_o|w_I)}{\partial v_I} = u_o - \sum_x P(w_x|w_I)\, u_x$$

## Skip-gram: Gradient Calculation

- The detailed steps for the gradient calculation are:

$$\frac{\partial \log P(w_o|w_I)}{\partial v_I} = \frac{\partial}{\partial v_I} \log \frac{e^{u_o^T v_I}}{\sum_x e^{u_x^T v_I}}$$

$$= \frac{\partial}{\partial v_I} \left( \log e^{u_o^T v_I} - \log \sum_x e^{u_x^T v_I} \right)$$

$$= u_o - \frac{1}{\sum_x e^{u_x^T v_I}} \sum_x u_x e^{u_x^T v_I}$$

$$= u_o - \sum_x P(w_x|w_I)\, u_x$$

- The update rule for $v_{w_c}$ is:

$$v_{w_c} \leftarrow v_{w_c} + \eta \left( v_{w_o} - \sum_{w \in V} P(w|w_c)\, v_w \right)$$

- Here, $\eta$ is the learning rate.

## Skip-gram Example

- Consider the sentence: "The quick brown fox jumps over the lazy dog."
- If the center word is "jumps", the model predicts context words such as "quick", "brown", "fox", "over", and "the".
- The model iteratively adjusts word vectors to predict these context words, learning a meaningful representation for "jumps."

# Skip-Gram Pseudocode

**Algorithm 1** Skip-Gram Model

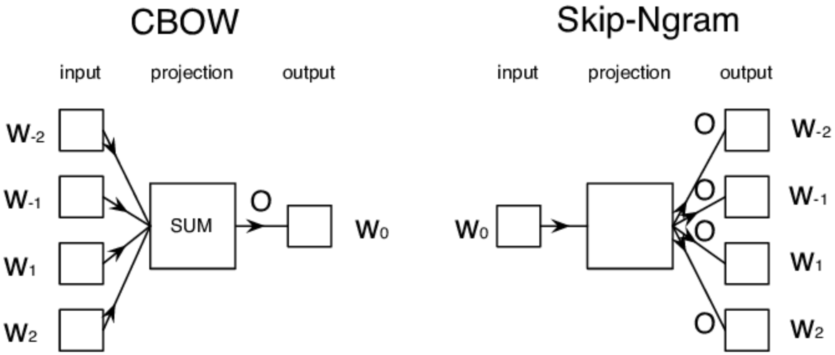
**Require:** Corpus $D$, window size $w$, embedding dimension $d$, learning rate $\alpha$, number of epochs $n$

```
Initialize word embeddings W and C randomly, where W maps words to embeddings and C maps context words to embeddings
for each epoch in 1 to n do
    for each sentence S in D do
        for each word w_t in S do
            Extract context words within window size w around w_t
            for each context word c of w_t do
                Compute dot product score = W(w_t) · C(c)
                Compute probability P(c|w_t) using softmax:
```

$$P(c|w_t) = \frac{\exp(\text{score})}{\sum_{c' \in \text{vocab}} \exp(W(w_t) \cdot C(c'))}$$

```
                Calculate loss L = −log P(c|w_t)
                Update W(w_t) and C(c) using gradient descent with learning rate η
            end for
        end for
    end for
end for
return Word embeddings W
```

# CBOW vs. Skip-gram




Difference between CBOW and Skip-gram

Figure adapted from researchgate.net/figure/llustration-of-the-Skip-gram-and-Continuous-Bag-of-Word-CBOW-models$_{f}ig1_2$81812760

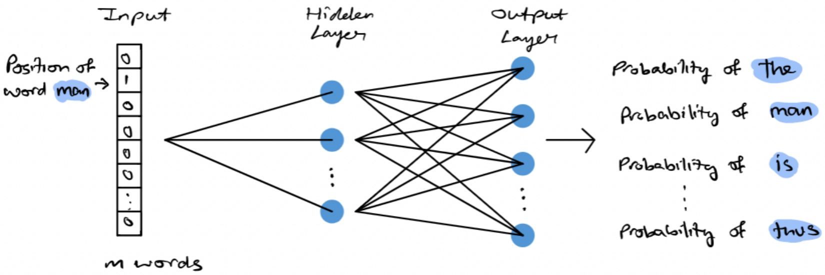
## CBOW vs. Skip-gram

- CBOW and Skip-gram are the two primary architectures for Word2Vec.
- **CBOW** predicts a target word given its surrounding context, making it efficient and effective for smaller datasets.
- **Skip-gram**, on the other hand, predicts the surrounding words for a given target word. It is well-suited for larger datasets and can handle rare words more effectively.
- In essence, the Skip-gram model captures more detailed word relationships and is robust in large vocabularies.

## Skip-gram as a Neural Network

- The Skip-gram model functions as a two-layer neural network.
- The input layer consists of a one-hot encoded target word vector, while the hidden layer is a dense embedding layer that learns the word representation.
- The output layer uses softmax to calculate the probability distribution over all words in the vocabulary, given the context.
- The Skip-gram model iteratively adjusts weights to maximize the likelihood of predicting the correct context words, ultimately learning meaningful word embeddings.

# Skip-gram as a Neural Network




Skip-gram Model as Neural Network

Figure adapted from towardsdatascience.com/word2vec-explained-49c52b4ccb71

## Why Do We Need Negative Sampling?

- The softmax function normalizes over all words in the vocabulary $V$, which can be very large (millions of words).
- This makes the calculation of $\sum_{w \in V} \exp(v_w \cdot v_{w_c})$ expensive, as it requires summing over all words in the vocabulary.
- **Solution**: Use **Negative Sampling** to only update a few randomly chosen "negative" words instead of the entire vocabulary.

## Skip-gram: Negative Sampling

- In **negative sampling**, we sample a few "negative" words that do not appear in the context of the target word.
- For each positive pair (center word and context word), we sample $k$ negative words that are not in the context.
- Instead of maximizing the probability of all words in the vocabulary, we only maximize the probability of the context words and minimize the probability of the sampled negative words.
- Example: If the center word is "cat", and the context word is "cute", we sample negative words like "computer", "sky", and "table" to minimize their probability in this context.

# Skip-gram: Negative Sampling

### Skipgram

| | | | | |
|---|---|---|---|---|
| shalt | not | make | a | machine |

| input | output |
|---|---|
| make | shalt |
| make | not |
| make | a |
| make | machine |

### Negative Sampling

| input word | output word | target |
|---|---|---|
| make | shalt | 1 |
| make | aaron | 0 |
| make | taco | 0 |

Negative sampling

Figure adapted from jalammar.github.io/illustrated-word2vec/

# Why Skip-gram?

- Skip-gram is preferred for large datasets because it handles rare words more effectively than CBOW.
- It captures detailed information about the surrounding words, leading to better word representations in the vector space.
- Word2Vec embeddings from Skip-gram have been widely adopted in various NLP tasks such as machine translation, sentiment analysis, and document classification.

1. Introduction

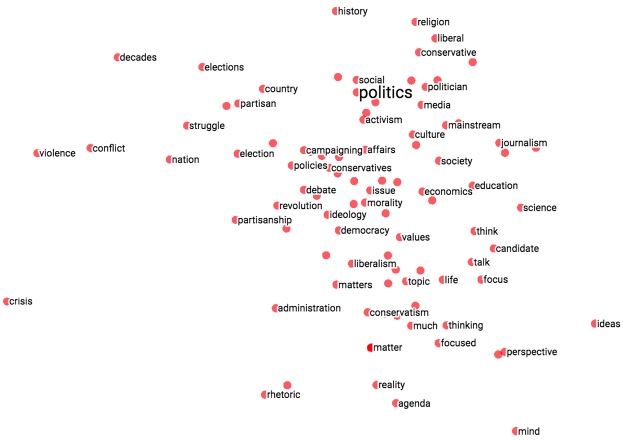
2. Word2Vec
   - Continuous Bag of Words (CBOW)
   - Skip-gram Model
   - Word Embedding Visualization
   - Word Analogy

3. References

## Visualizing Words in 2D

- After training the model, words are mapped into a high-dimensional vector space.
- Using techniques like PCA or t-SNE, these vectors can be reduced to 2D for visualization, where similar words appear closer together.

# Visualizing Words in 2D




Words represented in a 2D space after dimensionality reduction

## Word Analogy: Vector Arithmetic in Word2Vec

- Word2Vec embeddings can solve analogy tasks by performing vector arithmetic.
- The analogy task takes the form:

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

- The analogy is solved by finding the word vector closest to $\mathbf{v}_{king} - \mathbf{v}_{man} + \mathbf{v}_{woman}$.

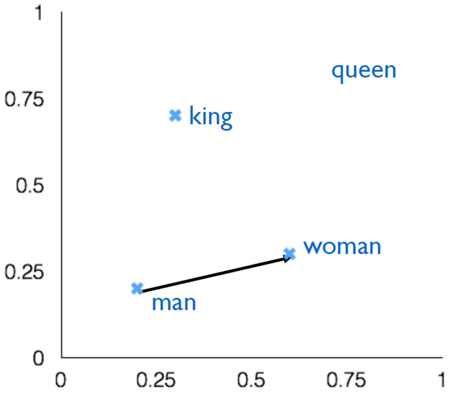
## Word Analogy Example

- Example:

$$\text{king}[0.30, 0.70] - \text{man}[0.20, 0.20] + \text{woman}[0.60, 0.30] \approx \text{queen}[0.70, 0.80]$$

- This means the vector difference between "king" and "man" is similar to the difference between "queen" and "woman."

# Word Analogy Example




Word analogy example

## Word Analogy Formula

- The formal formula for solving word analogies is:

$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

- This finds the word $x_i$ whose vector is closest to the result of the vector arithmetic.

1. Introduction
2. Word2Vec
3. References

[1] M. Soleymani, "Machine learning." Sharif University of Technology.

[2] M. Soleymani, "Modern information retrieval." Sharif University of Technology.

[3] C. Manning, "Natural language processing with deep learning." Stanford.

[4] E. Asgari, "Natural language processing." Sharif University of Technology.