**3**

**Word Embedding**

**Learning Objectives**

After completing this chapter, the readers are expected to

* Understand the motivation behind word embeddings and their significance in NLP.
* Learn the concept of distributional hypothesis and its role in vector semantics.

Word embeddings have revolutionised NLP by transforming how machines perceive and process human language. Word embeddings are dense vector representations of words that encode semantic and syntactic relationships into a multidimensional vector space. These linguistically informed vectors are used for various downstream activities, ranging from basic text analysis to advanced language understanding and generation tasks. In short, word embeddings bridge the gap between raw text and machine comprehension, enabling computers to understand the nuances of language and contextual meanings.

Consider a real-world scenario where word embeddings can be helpful. Let us take an e-commerce company trying to build a customer assistant chatbot. To build this chatbot, we need to ensure that the computer understands human language. However, the computer only understands numeric values. In computer vision, images can be represented by their numeric pixel values, allowing the computer to quickly analyse and process them. Similarly, let us try to describe the words of natural language in ASCII form by replacing each word’s characters with their ASCII values. For example, the words ‘ship’, ‘bank’, and ‘sell’ can be represented as:

This approach has several issues. First, the ASCII values alone do not carry any semantic information about the words or their associations. Second, there is no room for contextual

understanding, as the word ‘bank’ will have the same representation for both the *riverbank* and *the financial bank*. Additionally, this representation has variable and inconsistent word lengths, which could be problematic for methods requiring a fixed input size. In contrast, word embeddings provide a dense and linguistically rich representation of words in the vector semantic space. For instance, word embeddings can represent the above example words in a fixed-length (*d*) dense vector as:

**3.1 Distributional Hypothesis**

The Distributional Hypothesis is one of the fundamental hypotheses in linguistics and serves as the motivation behind word embeddings. The underlying idea that “a word is characterised by the company it keeps” was formulated by linguists like Joos (1950), Harris (1954), and Firth (1957). According to this hypothesis, words appearing in similar contexts tend to have similar meanings. For instance, two words, ‘student’ and ‘teacher’, often appear in similar contexts like ‘education’ and ‘school’. Suppose we have two sentences, ‘*The student asked a question*’ and ‘*The teacher answered the question.*’ Here, ‘student’ and ‘teacher’ are connected to the word ‘question’, while ‘asked’ and ‘answered’ are actions related to the question. This is where the distributional hypothesis comes in, allowing us to recognise these patterns and teaching machines a concrete sense of word meaning. Word embeddings are the subsequent step built on this, enabling computers to understand and generate human language.

**3.2 Vector Semantics**

Vector semantics is the standard method for representing word meaning in NLP, helping the model in understanding various aspects of word meaning. The idea was originated by Osgood et al. (1957). Combined with the vector representation and the distributional hypothesis, word vectors of similar words lie in the same distributional vector space, meaning two similar words are neighbours in the vector space. The concept of vector semantics involves representing a word as a point in a multidimensional semantic space derived from the distributions of neighbouring word embeddings. In Figure 3.1, we see that word embeddings of similar categories group into distinct clusters. Even within a category, the more closely related words are positioned closer together. For instance, in the sports cluster, ‘ball’ sports such as

Insert\_figure

Figure 3.1: A two-dimensional t-SNE projection of word vectors using GloVe-6B with 300-dimensional word embeddings. The relative positions of these word vectors in this two-dimensional vector space reflect the semantic meanings captured in the GloVe embeddings. Here, words of similar types are neighbours, which are clustered in the dotted ellipses. Notice how ‘apple’ lies between the cluster of ‘food’ and the ‘devices.’ Also, note that the axes values are arbitrary t-SNE components, and only the relative positions are meaningful.

‘

***3.2.1 Defining and Measuring Semantic Similarity***

We need a metric to measure the similarity between the *v* and *w* vectors of the *N* dimension. The cosine of the angle between the vectors is one such approach. It is based on the dot product:

The intuition is that two similar vectors will have higher dot product scores, while the score will be close to zero when they are dissimilar. However, one issue with dot products is that they tend to give more weight to longer vectors. The length of a vector is defined as:

The dot product favours longer vector lengths, but we need a metric that returns the similarity regardless of their length. Therefore, we must modify the dot product to normalise it by the vector’s length. It turns out that the normalised dot product is equivalent to the cosine between the two vectors. Based on this, the new normalised dot product between two vectors *a* and *b* is given as:

(3.3)

Now, we can compute the cosine similarity between two vectors *v* and *w* as:

**3.3 Types of Word Embedding**

Now, let us explore the different types of word embeddings, which are typically categorised into *frequency-based* and *prediction-based* methods. In frequency-based methods, raw frequencies or weighted frequencies of co-occurrences are used for word representation. We will discuss two count-based approaches: *co-occurrence-based* and *TF-IDF*. The resultant vectors from these approaches are sparse and of extremely high dimension.

In the prediction-based section, we include Word2Vec, GloVe, and FastText. These methods use prediction tasks as a proxy for learning word embeddings. The resultant vectors are shorter, denser, and capable of capturing more complex linguistic information. Word2Vec learns from word contexts in sentences, GloVe from word co-occurrence across a corpus, and FastText by incorporating subword information.

***3.3.1 Frequency-based Embeddings***

Frequency-based approaches in word embedding involve representing words based on their co-occurrence statistics in a large corpus of text. These methods are rooted in the idea that words that frequently appear in similar contexts tend to have similar meanings, an idea known as the distributional hypothesis.

*<H4>Co-occurrence Matrix*

In this approach, a matrix is created where each row represents a word, and each column represents a context (often another word in a specified window around the target word or the document where the word appears). The co-occurrence matrix of words can be either a word-document (term-document) matrix or a word-word (term-term) matrix.

Table 3.1: The term-document matrix for five words in four magazine documents. Each cell contains the number of times a word (row) appears in the document (column).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Fruit Market Overview | Tropical Fruit Guide | Tech Trends | Healthy Tech Lifestyle |
| apple | 10 | 0 | 0 | 8 |
| banana | 0 | 12 | 0 | 0 |
| fruit | 7 | 5 | 0 | 5 |
| technology | 0 | 0 | 15 | 3 |
| software | 0 | 0 | 10 | 4 |

**Example 3.1.** Let us determine whether ‘fruit’ is more similar to ‘apple’ or ‘technology’ using the example from Table 3.1.

The vector representations for the words are: ‘apple’ = [10*,* 0*,* 0*,* 8], ‘fruit’ = [7*,* 5*,* 0*,* 5], and ‘technology’ = [0*,* 0*,* 15*,* 3]. Using Equation (3.4), we calculate the cosine similarity by:



These results indicate that in this vector space representation, ‘fruit’ shows a significantly higher semantic similarity to ‘apple’ than to ‘technology’. This outcome aligns with the intuitive conceptual relationships among these terms.

*<H4> Term Frequency-Inverse Document Frequency (TF-IDF)*

1. **Term Frequency (TF**): TF measures how frequently a term occurs in a document. It is typically calculated as:

*TFt,d* = *ft,d* (3.5)

where *ft,d* is the raw frequency of term *t* in document *d*. Since the number of documents can be very large, a logarithmic scaling can be used to dampen the effect of large frequency differences:



Table 3.2: The TF matrix for terms ‘the’, ‘lazy’, ‘dog’, ‘quick’, and ‘fox’ with respect to four documents: Doc 1, Doc 2, Doc 3, and Doc 4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | the | lazy | dog | quick | fox |
| Doc 1 | 2 | 1 | 1 | 1 | 1 |
| Doc 2 | 1 | 1 | 1 | 0 | 0 |
| Doc 3 | 2 | 0 | 0 | 1 | 1 |
| Doc 4 | 1 | 1 | 0 | 0 | 0 |

2. **Inverse Document Frequency (IDF)**: IDF measures how important a term is across

the entire corpus. It is calculated as:



where *N* is the total number of documents in the corpus, and *nt* is the number of documents containing the term *t*.

**Vector Representation using TF-IDF.** The TF-IDF score of each term or vocabulary word in the term-document matrix can also be used to represent a document or the word. Consider the following example:

**Example 3.2.** Consider a small corpus of 4 documents:

1. Doc 1: ‘*The quick brown fox jumps over the lazy dog*’.
2. Doc 2: ‘*The lazy dog sleeps all day*’.
3. Doc 3: ‘*The quick brown fox hunts in the forest*’.
4. Doc 4: ‘*A lazy afternoon in the forest*’.

Let us consider these four documents and the terms[[1]](#footnote-1) ‘the’, ‘lazy’, ‘dog’, ‘quick’, and ‘fox’. In Table 3.2, each value is the TF score. For example, the TF for ‘*lazy*’ with respect to Doc 2 in Table 3.2 is computed by: *TF*(‘*lazy*’*,* Doc 2) = *count*(‘*lazy*’*,*Doc 2) = 1. Next, we compute the IDF score for each term q:

* IDF(‘the’) = log2(4*/*3) ≈ 0*.*415
* IDF(‘lazy’) = log2(4*/*3) ≈ 0*.*415
* IDF(‘dog’) = log2(4*/*2) = 1
* IDF(‘quick’) = log2(4*/*2) = 1
* IDF(‘fox’) = log2(4*/*2) = 1

We then calculate the TF-IDF score for each term. We do this by multiplying each TF value by its corresponding IDF, as presented in Table 3.3. For instance, in Table 3.3, the TF-IDF

Table 3.3: The TF-IDF matrix for the terms, ‘the’, ‘lazy’, ‘dog’, ‘quick’, and ‘fox’ with respect to the four documents: Doc 1, Doc 2, Doc 3, and Doc 4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | the | lazy | dog | quick | fox |
| Doc 1 | 0.830 | 0.415 | 1 | 1 | 1 |
| Doc 2 | 0.415 | 0.415 | 1 | 0 | 0 |
| Doc 3 | 0.830 | 0 | 0 | 1 | 1 |
| Doc 4 | 0.415 | 0.415 | 0 | 0 | 0 |

Each row vector in Table 3.3 represents the corresponding document. For instance, Doc 2 is represented as [ 0.415, 0.415, 1, 0, 0]. Each column vector can be used to represent the word vector. For example, the word ‘quick’ is represented as [1,0,1,0].

***3.3.2 Word2Vec***

In the previous sections, we explored frequency-based approaches to show how words can be represented in vector form using the raw counts from a co-occurrence matrix and the weighted values as in TF-IDF. However, these representations have limitations, particularly with vector dimensions scaling to the size of the vocabulary, |*V* |. For example, a corpus with 100,000 unique tokens would produce vectors with a dimension of 100,000, leading to large, often sparse vectors with most values being zeros. This sparsity increases computational and memory requirements, making word representation less efficient.

*<H4>Skip-Gram model*

The Skip-Gram model is one of the two primary models introduced by Word2Vec (Mikolov et al. 2013a,b). The fundamental idea behind the Skip-Gram model is to select a central word from a sentence and use it to predict its surrounding context words within a specified window. This process helps train a classifier to learn the embedding weights, which can later be used as dense vector representations of words.

For instance, consider the example sentence: “*I live in Abu Dhabi.*” If we choose the word ‘live’ as the centre word and set a context window of ±1, the model would attempt to predict the words ‘I’ and ‘in’ as the context words surrounding ‘live.’ This setup is illustrated in Figure 3.2. The Skip-Gram model thus leverages the prediction of context words to refine the word embeddings, with the aim of capturing semantic and syntactic relationships between words.

Insert\_figure

Figure 3.2: An illustration of the skip-Gram model to predict the context words ‘I’ and ‘in’ given the input centre word ‘life’. The centre word is converted to a one-hot encoder vector of dimension |*V* |, and the output final layers are the one-hot encoding vectors for the two predicted outputs. The model learns the weight matrices, *P* and *Q*, which are used as the embedding. Each row of the *P* matrix is a *d*-dimensional embedding.

embedding vectors, *p* and *q*, for every word in the vocabulary. The following steps outline the process for learning these embedding weights:

1. **Generation of One-hot Input Vector:** Initially, the centre word is represented as a one-hot vector. For a given centre word *x*, we create a |*V* |-dimensional one-hot vector, where |*V* | is the vocabulary size. This vector has a value of 1 at the index corresponding to the centre word *w* in the vocabulary and 0 elsewhere. Mathematically, it is represented as:

Here, index(*w*)refers to the index of the centre word *w* in the vocabulary.

**2. Get the Embedded Vector for the Centre Word:** Next, we multiply the one-hot vector *x* by the input word matrix *P* to obtain the embedding vector *pc* for the centre word. This operation can be expressed as:

*pc* =*PT x* ∈ ℝ*d*

Here, *PT* is the transpose of the input word matrix *P*, and *pc* is a *d*-dimensional vector that represents the embedding of the centre word. This multiplication effectively selects the column in *P* corresponding to the center word’s index, yielding its embedding vector.

3. **Getting the Similarity Between the Centre and Context Words:** The intuition behind the Skip-gram model is that the likelihood of a word appearing close to a target word depends on the similarity of their embedding vectors. To measure this similarity, we can use the dot product, where a higher dot product indicates a greater degree of similarity between the words. To compute the similarity score, we multiply the centre word vector *pc* by the output word matrix *Q,* resulting in the score vector *z:*

Each element in *z* represents the dot product score between the context word and the centre word, reflecting the model’s prediction of how likely each word is to appear in the context of the centre word.

**4. Turn Similarity Scores into Probability:** The similarity scores obtained in the previous step are not sufficient for our prediction task for several reasons. First, these scores are not probabilities; they range unboundedly between –∞ and +∞. Second, the scores do not account for relative scores across all the words in the vocabulary. For our task, we need a concrete probability that represents the likelihood of a context word given a centre word.

To achieve this, we use the *softmax function* to convert these scores into a probability

distribution:

Here, *ŷc*–*m*, …, *ŷc*–1, *ŷc*+1, …, *ŷc*+*m* represent the probabilities of observing each context word within a window of size *m* around the centre word. These probabilities correspond to the respective elements in the *ŷ* vector, making them suitable for the prediction task.

Now, Stochastic Gradient Descent (SGD) (as discussed in Chapter 2) can be used to update all the relevant word vectors *qc* and *pc*.

Figure 3.3: Illustration of the CBOW model - the context words {‘I’ and ‘in’} is used for predicting centre word ‘live’.

*<H4>Continuous Bag of Words (CBOW) Model*

The Continuous Bag of Words (CBOW) model is another approach proposed in the Word2Vec framework, contrasting with the Skip-Gram model. In the CBOW model, instead of using a centre word to predict its surrounding context words, the context words are used to predict the centre word. For example, in the sentence ‘*I live in Abu Dhabi*’, if the word ‘live’ is the centre word, the CBOW model would use the surrounding context words ‘I’ and ‘in’ to predict the centre word ‘live’. This approach is shown in Figure 3.3.

The key idea behind CBOW is that the model learns to represent words by predicting a target word from its neighbouring words. This differs from the Skip-Gram model, where the model learns to predict surrounding words based on a given centre word. The CBOW model is particularly useful when dealing with smaller datasets, as it tends to perform better in such scenarios due to the averaging effect of the context words, which provides a more stable learning signal compared to the Skip-Gram model.

1. **Obtain the Embedded Vector for the Centre Word:** We then multiply each one-hot vector *x* with the input word matrix *P* to get the embedded word vectors *pc*-*m* for the context words:



Here, each *pi* is the *d*-dimensional embedded vector for a context word.

3. **Averaging Context Vectors:** Unlike in the Skip-Gram model, where there is only a single input vector, the CBOW model has multiple input vectors. To combine these, we average the context vectors:



Here,  represents the aggregated context of the centre word.

4. **Generate the Score Vector *z:*** We multiply the averaged context vector  by the matrix *Q* to generate a score vector:



Each element in *z* corresponds to a word in the vocabulary and its compatibility score with the given context. Since the dot product of similar vectors is higher, this operation tends to bring similar words closer together, resulting in a higher score.

5**.** **Turn Similarity Scores into Probability:** We apply the softmax function to *z* to obtain a probability distribution over all words in the vocabulary.

 (3.7)

6**. Objective Function:** We need to come up with an objective function to match the predicted probability, with the actual probability  to learn the *Q* and *P* matrices. We use cross-entropy loss as our objective function to measure the difference between our predicted probability  and the true probability *y:*

 (3.8)

Now, let us see if cross-entropy provides us with a good measure of distance between the predicted and the actual probability. We can simplify Equation (3.8) for a single *yi* as:

 (3.9)

Let us consider a perfect prediction, i.e., ; here **. Thus, we do not face any penalty for a perfect prediction. Now, let us consider a case where the prediction is very bad and. Thus, we see that cross-entropy provides us with a good measure of distance. Now, we optimise this objective as below:

Now, Stochastic Gradient Descent can be used to update all the relevant word vectors *qc* and *pc.*

***3.3.3 Global Vectors for Word Representation (GloVe)***

Previously, we discussed word representation techniques using count-based methods (like co-occurrence matrices) and predictive methods (like Word2Vec). Count-based methods often fall short in tasks such as analogy, indicating that their vector representations are suboptimal. Additionally, their large dimensions scale up to the vocabulary size, which can be inefficient. Conversely, the Word2Vec model excels at capturing complex linguistic patterns beyond word similarity but does not fully utilise global co-occurrence statistics.

Global Vectors for Word Representation (GloVe) (Pennington et al. 2014) aims to combine the strengths of both approaches by leveraging the global statistical information captured by word-word co-occurrence matrices while also benefiting from the local context window approach used in Word2Vec. Specifically, GloVe addresses the limitations of both count-based and prediction-based (Word2Vec) representations by:

1. **Utilisation of Global Statistics**: GloVe constructs a global word-word co-occurrence matrix from the entire corpus, similar to the count-based method. This approach enables GloVe to efficiently leverage statistical information across the corpus, which might be lost in Word2Vec, as it learns word embeddings by making predictions within local context windows.
2. **Efficient Learning of Vector Space**: While count-based approaches use global statistics, they often produce suboptimal vector representations for tasks like analogy. In contrast, GloVe employs a weighted least squares training objective on global word-word co-occurrence counts, making more efficient use of the statistical data. In count-based approaches, frequent co-occurrences, such as stop words, tend to dominate, while rare co-occurrences are often overlooked. To address this, GloVe introduces a weighting function that balances the significance of both frequent and rare co-occurrences.

Table 3.4: An illustration of co-occurrence probabilities and ratios for the target words *comedy* and *horror* with *k* probe words as context from a hypothetical corpus.

|  |  |  |  |
| --- | --- | --- | --- |
| *k* | *P*(*k*|*comedy*) | *P*(*k*|*horror*) | *P*(*k*|*comedy*)*/P*(*k*|*horror*) |
| laughter | 0.0080 | 0.0009 | 8.89 |
| scary | 0.0005 | 0.0100 | 0.05 |
| popcorn | 0.0050 | 0.0045 | 1.11 |
| algebra | 0.0001 | 0.0001 | 1.00 |

We will cover the details of the weighted cost function in the subsequent section.

*<H4>The GloVe Model*

In this section, we will delve into the inner workings of the GloVe model. Before proceeding, let us establish some notations. GloVe uses a global word-word co-occurrence matrix, denoted by *X.* Each element in this matrix, *Xij,* represents the count of occurrences of word *j* given the context word *i.* Now, let us define *Xi =* Σ*k Xik* as the total count of all words in the context of word *i,* which is the sum of all co-occurrence counts across the entire corpus when *i* is the context word. Based on this, we can calculate the probability of word *j* given the context word *i* as follows:



Let us consider an example of how co-occurrence probabilities can be utilised to derive certain aspects of meaning.

**Example 3.3.** Let us consider two words *i* and *j* that represent movie genres: *i = comedy* and *j = horror.* We will examine the relationship of the words *i* and *j* by studying the ratio of their co-occurrence probabilities with some probe words *k.* In the example shown in Table 3.4, we use *k = laughter* for words related to *comedy* but not with *horror,* and the ratio of *Pik/Pjk* or *P*(*laughter|comedy*)*/P*(*laughter|horror*)is large (8.89) as expected. Similarly, *k = scary* will be more related to *horror,* and the ratio *Pik/Pjk* is low (0.05). And, the words that are either related to both *comedy* or *horror* or totally unrelated will have a ratio close to one. This example demonstrates that a ratio of probabilities can effectively distinguish words relevant to one genre but not the other (*laughter, scary*)and words relevant to either both (*popcorn*)or none (*algebra*)*.* In short, this approach of taking the ratio of co-occurrence probabilities provides a more effective and distinctive measure as against the raw probabilities.

*<H4> Benefits of FastText over Other Models*

Compared to traditional word-level representation, the subword information from the character-level *n*-gram representation makes FastText robust and flexible in handling OOV words, morphological structures, and rare words.

1. **Handling Out-of-Vocabulary (OOV) Words:** Let us consider an example word, ‘unfriendliness’, which is absent in the corpus. Hence, it is impossible to represent this word using word-level embedding as it is out of vocabulary. However, in FastText with 3-gram character-level subwords, ‘unfriendliness’ can be broken into the following trigrams:

<un, unf, nfr, fri, rie, ien, end, ndl, dli, lin, ine, nes, ess, ss>.

In FastText, this word can be represented as the average of these trigram embeddings. With this approach, even an out-of-vocabulary word can be represented, as there is a chance of having the embedding representation for these subwords from other words in the corpus.

**3.4 Bias in Word Embedding**

Word embeddings can capture semantic relationships that extend beyond mere vector similarity. One task that showcases this capability is the analogy task, where word embeddings can solve problems of the form “*a* is to *b* as *c* is to *d*.” Consider the following example for an analogy task:

king – man + woman ≈ queen*.*

This analogy captures the gender-based relationship between ‘king’ and ‘queen’. Similarly, consider another example:

Paris – France + Italy ≈ Rome*.*

Like the previous example, the word vectors establish the relationship between a country and its capital. These analogical reasoning capabilities also extend to various semantic and syntactic relationships, such as adjectives (e.g., good : better :: bad : worse), plurals (e.g., man : men :: woman : women), and country-currency relationships (e.g., USA : dollar :: Japan : yen).

However, this ability to perform analogy tasks also reveals some biases inherent in the training corpus. A study by Bolukbasi et al. (2016) found that the Word2Vec model trained on news data is biased towards gender in occupations. For instance, the occupation closest to the ‘woman’ vector was ‘homemaker’, while ‘man’ was associated with ‘computer programming’:

computer programming – man + woman ≈ homemaker*.*

This suggests an association of certain professions with specific genders, potentially reinforcing harmful stereotypes. Research has shown that these biases are not merely reflections of the training data but are often amplified beyond the input text statistics (Zhao et al. 2017). Additionally, these biases extend beyond gender; similar issues have been found with racial and ethnic biases. A study revealed that African-American names were associated with unpleasant words when tested on GloVe vectors (Caliskan et al. 2017), while European-American names had higher cosine similarity with pleasant words. Various approaches have been proposed to mitigate these biases, such as transforming the embedding space to remove gender stereotypes while preserving the notion of gender (Bolukbasi et al. 2016; Kumar et al. 2020) or altering the training paradigm (Zhao et al. 2018). However, addressing these biases remains an open challenge.

**3.5 Limitations of Word Embedding Methods**

We explored various vector representation methods for words, each with its limitations. Initially, count-based approaches result in sparse vectors, especially when there are zero cooccurrences, and their dimensionality scales with the vocabulary size. This often necessitates dimensionality reduction techniques like Singular Value Decomposition to manage the high dimensionality. In contrast, prediction-based approaches such as Word2Vec, GloVe, and FastText generate dense vectors with much smaller, arbitrarily chosen dimensions. However, selecting the optimal dimension requires balancing the quality of the embeddings and computational efficiency.

Another significant limitation is that traditional embedding methods struggle to represent out-of-vocabulary words. While FastText addresses this issue to some extent by incorporating subword-level information, it still faces challenges with entirely novel words.

Moreover, once trained, these embeddings remain static and must be retrained to adapt to language evolution, leading to semantic drift. For example, static embeddings trained on historical data may fail to relate the word ‘web’ with ‘internet’ as language usage changes over time. Additionally, these embeddings lack contextual information; even in prediction-based methods, only a small window of context words is available during training, which fails to capture the full nuances of context.

Finally, as discussed earlier, these embeddings are not only biased towards the training data but also tend to amplify these biases, further complicating their application.

**3.6 Applications of Word Embeddings**

We discovered how word embeddings can measure the similarity between two words and perform analogy tasks. However, the capabilities of word embeddings extend far beyond these basic functions. For instance, they can be used for document-level representation, as seen in Doc2Vec (Le and Mikolov 2014), which plays a crucial role in information retrieval for search engines and recommendation systems.

Word embeddings trained on multilingual data have demonstrated strong cross-lingual capabilities, where similar words from different languages are positioned close to each other in the vector space. For example, a sentiment classifier model trained on English data can classify a French movie review by mapping French words to a shared English-French embedding space. Furthermore, cross-lingual aligned embedding vectors enable unsupervised machine translation without parallel corpora by using the nearest neighbours of source words in the target language as an initial word-level translation.

Word embeddings have also paved the way for contextual models like BERT (Devlin et al. 2018), which assigns dynamic embeddings based on context, unlike traditional word embeddings that offer static representations regardless of context. Recently, more advanced embeddings have been instrumental in revolutionising Retrieval-Augmented Generation (RAG) systems, significantly enhancing their capabilities.

**Additional Resources**

1. **Important Articles/Books**

* Efficient Estimation of Word Representations in Vector Space (Mikolov et al. 2013a)
* Enriching Word Vectors with Subword Information (Bojanowski et al. 2017)GloVe: Global Vectors for Word Representation (Pennington et al. 2014)
* Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings (Bolukbasi et al. 2016)
* Speech & language processing (Jurafsky 2000)

**Exercises**

***True/False Questions***

1. Distributional hypothesis states that words with similar meaning tend to occur in similar context. (True/False)
2. Cosine similarity is favoured over Euclidean distance for measuring the similarity between the word embeddings since it is invariant to magnitude of the vector. (True/False)
3. In the context of bias in word embeddings, the debiasing technique of transforming the bias direction from the word vectors always completely eliminate all forms of bias. (True/False)
4. The primary goal of a prediction-based method is to predict the context or the target words. (True/False)
5. FastText is capable to representing out-of-vocabulary words. (True/False)

***Multiple Choice Questions***

1. What does cosine similarity measure in the context of word embeddings?

1. The Euclidean distance between two-word vectors.
2. The angle between two-word vectors.
3. The magnitude difference between two-word vectors.
4. All of the above.

2. Cross-entropy loss is chosen instead of least squares as the measure of distance or loss between the actual one-hot and the predicted one-hot vectors in the Skip-Gram model because:

1. Cross-entropy allows for easier gradient updates.
2. Cross-entropy is computationally less expensive than least squares.
3. Cross-entropy is better suited for the probabilistic nature of the Skip-gram’s prediction task.
4. Least squares would cause the model to overfit the training data.

***Short Questions***

1. What are the issues of representing words by their character level ASCII values?
2. Why is the distributional hypothesis the basis of word embeddings?
3. What are the differences between count-based and prediction-based vector representation?
4. Why a logarithms is used for calculating the scores in TF-IDF?
5. When do you think the TF-IDF will be more suitable than Word2Vec?
6. Why softmax is needed after calculating the similarity score of the centre word and the output word matrix in Skip-Gram?

***Long Questions***

1. Formalise and explain why the cosine similarity is suitable for measuring the similarity between two-word embeddings?
2. Consider a small corpus of 5 documents, with 100, 80, 90, 120, and 95 words. The word ‘algorithm’ appears 3, 0, 2, 1, and 0 times in the documents, respectively. Now, calculate the TF-IDF score for the ‘algorithm’ in the first and the fourth documents. Consider using log base 2 for the calculation and use Equation (3.5) for calculating the TF.
3. Give a suitable example to prove why cross-entropy is a good fit for measuring the distance or loss between the predicted and actual one-hot vectors in the Skip-Gram model.

**Bibliography**

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146.  
URL https://doi.org/10.1162/tacl\_a\_00051

Bolukbasi, T., Chang, K.-W., Zou, J. Y., Saligrama, V., and Kalai, A. T. (2016). Man is to computer programmer as woman is to homemaker? debiasing word embeddings. *Advances in neural information processing systems*, *29*.

Caliskan, A., Bryson, J. J., and Narayanan, A. (2017). Semantics derived automatically from language corpora contain humanlike biases. *Science*, *356*(6334), 183–186.  
URL https://www.science.org/doi/abs/10.1126/science.aal4230

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dierk, S. (1972). The smart retrieval system: Experiments in automatic document processing—gerard salton, ed.(englewood cliffs, nj: Prentice-hall, 1971, 556 pp., 15.00). *IEEE Transactions on Professional Communication*, (1), 17–17.

1. We have considered only these five terms as the vocabulary for simplicity. [↑](#footnote-ref-1)