به نام خدا

# تمرین سری دوم درس مبانی پردازش زبان و گفتار دکتر مرضیه داود آبادی فراهانی

# فرزان رحمانی 99521271

## سوال اول

1. **Word embedding** و **One-hot encoding** دو رویکرد متفاوت برای نمایش داده‌های متنی به صورت عددی برای پردازش توسط مدل‌های یادگیری ماشین و NLP هستند.

**One-hot encoding** برداری ایجاد می کند که در آن هر بعد با یک کلمه در واژگان مطابقت دارد. بردار پراکنده(sparse) است، با 1 در بعد مربوط به خود کلمه، و 0 در تمام ابعاد دیگر. این یک نمایش ناکارآمد برای مجموعه واژگان بزرگ است.  
ویژگی های آن به طور خلاصه به شرح زیر است:

* + ساده و کارآمد برای مجموعه داده های کوچک.
  + هر کلمه را به عنوان یک بردار با یک "1" در موقعیت مربوط به کلمه و "0" در هر جای دیگر نشان می دهد.
  + نقطه ضعف: بردارهای پراکنده با ابعاد بالا ایجاد می کند، برای مجموعه واژگان بزرگ ناکارآمد است، و روابط معنایی بین کلمات را در نظر نمی‌گیرد.

**Word embedding** کلمات را به عنوان بردارهای متراکم از اعداد حقیقی در فضایی با ابعاد پایین تر نشان می دهد. کلمات مشابه معمولاً بازنمایی های برداری مشابهی دارند و روابط معنایی و نحوی را به تصویر می کشند. جاسازی ها به جای اینکه به صورت دستی مشخص شوند، از داده ها با روشی بدون نظارت یاد می گیرند.  
ویژگی های آن به طور خلاصه به شرح زیر است:

* + کلمات را به عنوان بردارهای متراکم در فضایی با ابعاد پایین تر نشان می دهد.
  + هر بعد جنبه ای از معنای کلمه را به تصویر می کشد.
  + کلماتی با معانی مشابه تمایل دارند که بردارهای نزدیک تری در این فضا داشته باشند.
  + برای مجموعه داده های بزرگ کارآمدتر است و امکان ثبت روابط معنایی و نحوی را فراهم می کند.

همچنین جواب chat gpt به شرح زیر بود که خالی از لطف نیست:

* **One-hot encoding:** In one-hot encoding, each word in a vocabulary is represented as a vector where all elements are zero except for the index corresponding to the word's position in the vocabulary, which is set to one. This creates a sparse representation where each word is represented as a unique vector.
* **Word embedding:** Word embedding, on the other hand, represents words as dense vectors of real numbers, typically with much lower dimensionality compared to the size of the vocabulary. Word embeddings are learned from data and capture semantic relationships between words based on their context.

1. Global Vectors for Word Representation (GloVe) الگوریتمی است که با ساختن یک ماتریس باهم‌آیی(co-occurrence matrix) از کلمات در یک مجموعه متنی(corpus)، جاسازی های کلمه(word embeddings) را ایجاد می کند. این ماتریس نشان می‌دهد که کلمات چقدر با کلمات متنی خود در مجموعه متنی همراه هستند. سپس GloVe از فاکتورسازی ماتریس(matrix factorization) برای نگاشت این اطلاعات آماری در فضای برداری استفاده می کند که در آن هر کلمه یک نمایش برداری دریافت می کند. بردارهای حاصل مستقیماً معنی و روابط کلمات را رمزگذاری می کنند.  
   در واقع، یک الگوریتم یادگیری بدون نظارت است که نمایش برداری یا جاسازی کلمات را ایجاد می کند. از داده های آماری باهم‌آیی کلمات در یک مجموعه متنی معین برای به دست آوردن روابط معنایی بین کلمات استفاده می کند. GloVe با استفاده از جفت‌های کلمه، یک ماتریس باهم‌آیی ایجاد می‌کند و سپس بردارهای کلمه را بهینه می‌کند تا تفاوت بین اطلاعات متقابل نقطه‌ای کلمات مربوطه و حاصل ضرب نقطه‌ای بردارها را به حداقل برساند.  
   همچنین در لینکی که داخل مراجع هست توضیحات دقیق تری آمده است که به شرح زیر است:

#### **Overview**

* Proposed in [GloVe: Global Vectors for Word Representation](https://aclanthology.org/D14-1162) by Pennington et al. (2014), Global Vectors for Word Representation (GloVe) embeddings are a type of word representation used in NLP. They are designed to capture not just the local context of words but also their global co-occurrence statistics in a corpus, thus providing a rich and nuanced word representation.
* By blending these approaches, GloVe captures a fuller picture of word meaning and usage, making it a valuable tool for various NLP tasks, such as sentiment analysis, machine translation, and information retrieval.

#### **How GloVe Works**

1. **Co-Occurrence Matrix:** GloVe starts by constructing a large matrix that represents the co-occurrence statistics of words in a given corpus. This matrix has dimensions of [vocabulary size] x [vocabulary size], where each entry (i,j) in the matrix represents how often word i occurs in the context of word j.
2. **Matrix Factorization:** The algorithm then applies matrix factorization techniques to this co-occurrence matrix. The goal is to reduce the dimensions of each word into a lower-dimensional space (the embedding space), while preserving the co-occurrence information.
3. **Word Vectors:** The end result is that each word in the corpus is represented by a vector in this embedding space. Words with similar meanings or that often appear in similar contexts will have similar vectors.
4. **Relationships and Analogies:** These vectors capture complex patterns and relationships between words. For example, they can capture analogies like “man is to king as woman is to queen” by showing that the vector ‘king’ - ‘man’ + ‘woman’ is close to ‘queen’.

1. Word2Vec یک شبکه عصبی دو لایه(یک لایه ورودی، یک لایه پنهان و یک لایه خروجی) است که متن را با "برداری کردن" کلمات پردازش می کند. ورودی آن یک پیکره متنی(corpus) و خروجی آن مجموعه ای از بردارها است: بردارهای ویژگی که کلمات را در آن پیکره متنی نشان می دهند. Word2Vec بردارهایی ایجاد می کند که نمایش های عددی ویژگی های کلمه، ویژگی هایی مانند زمینه تک تک کلمات، توزیع شده اند. این کار را بدون دخالت انسان انجام می دهد. با استفاه از داده‌های کافیو زمینه‌های کافی، Word2Vec می‌تواند بر اساس ظاهر گذشته حدس‌های بسیار دقیقی در مورد معنای یک کلمه بزند.  
   به طور خلاصه:

* این الگوریتم در دو معماری اصلی ارائه می شود: Continuous Bag of Words (CBOW) و Skip-gram.
* در معماری CBOW، مدل کلمه مورد نظر(target) را بر اساس context خود (کلماتی که آن را در یک پنجره خاص احاطه کرده اند) پیش بینی می کند.
* در معماری Skip-gram، مدل context word ها را پیش‌بینی می‌کند که به عوان ورودی یک کلمه هدف(target) داده می‌شود.
* Word2Vec با آموزش یک شبکه عصبی کم عمق بر روی مجموعه بزرگی از متن، جاسازی کلمات را می آموزد تا احتمال پیش بینی context را با یک کلمه هدف (یا برعکس) به حداکثر برساند.  
  همچنین در لینکی که داخل مراجع هست توضیحات دقیق تری آمده است که به شرح زیر است:

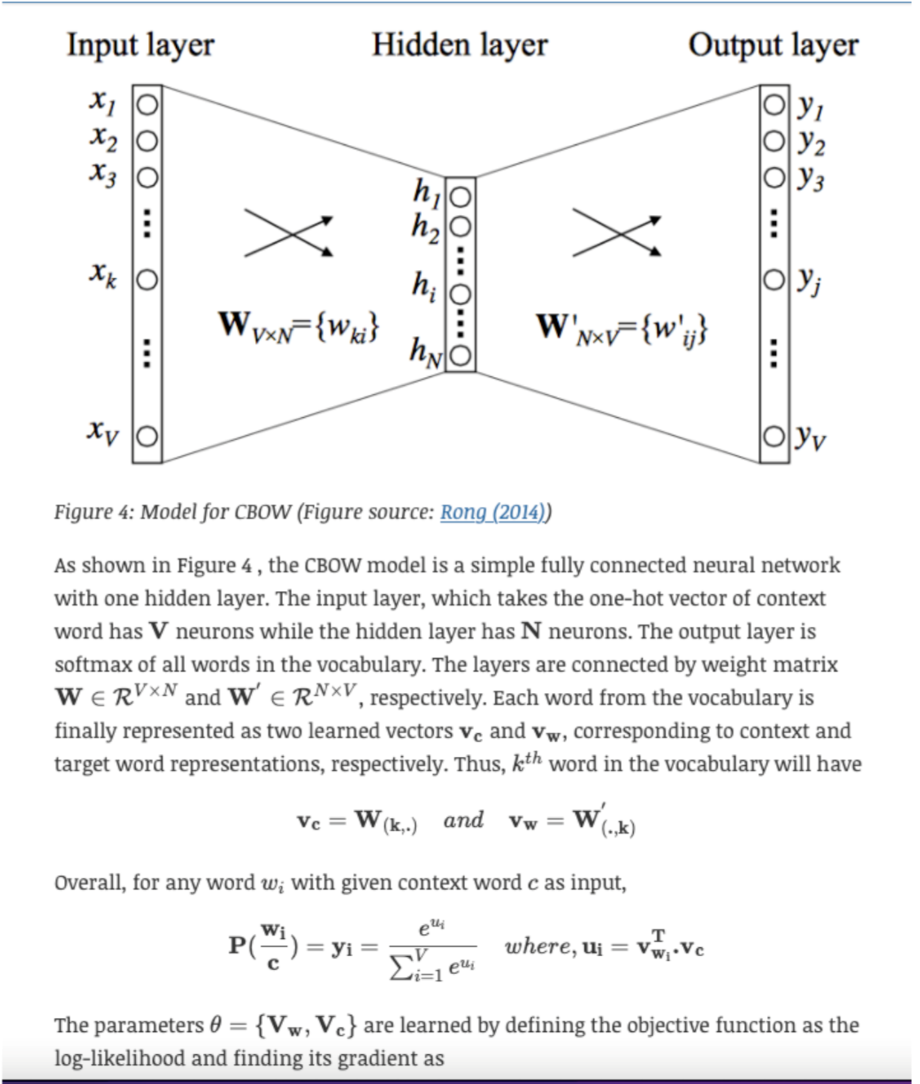
### **Word2Vec**

* Proposed in [Efficient Estimation of Word Representations in Vector Space](https://arxiv.org/abs/1301.3781) by Mikolov et al. (2013), the Word2Vec algorithm represented a significant advancement in the field of NLP at the time as a notable example of a word embedding technique.
* Word2Vec is renowned for its effectiveness in learning word vectors, which are then used to decode the semantic relationships between words. It utilizes a vector space model to encapsulate words in a manner that captures both semantic and syntactic relationships. This method enables the algorithm to discern similarities and differences between words, as well as to identify analogous relationships, such as the parallel between “Stockholm” and “Sweden” and “Cairo” and “Egypt.”
* Word2Vec’s methodology of representing words as vectors in a semantic and syntactic space has profoundly impacted the field of NLP, offering a robust framework for capturing the intricacies of language and its usage.

#### **Core Idea**

* Word2Vec employs a shallow neural network, trained on a large textual corpus, to predict the context surrounding a given word. The essence of Word2Vec lies in its ability to convert words into high-dimensional vectors. This representation allows the algorithm to capture the meaning, semantic similarity, and relationships with surrounding text. A notable feature of Word2Vec is its capacity to perform arithmetic operations with these vectors to reveal linguistic patterns, such as the famous analogy ‘king - man + woman = queen’.

#### **Word2Vec Architectures**

* Word2Vec offers two distinct architectures for training:
  1. **Continuous Bag-of-Words (CBOW)**: This model predicts a target word based on its context words. The input is a summation of the word vectors of the surrounding context words, with the output being the current word. This approach is depicted in the following image:
  2. **Skip-gram**: Conversely, the Skip-gram model predicts the surrounding context words from a given target word. The input is the target word, and the output is a softmax classification over the entire vocabulary to predict the context words.

#### **Training and Optimization**

* The training of Word2Vec involves representing every word in a fixed vocabulary by a vector and then optimizing these vectors to predict surrounding words accurately. This is achieved through stochastic gradient descent, minimizing a loss function that indicates the discrepancy between predicted and actual context words. The algorithm uses a sliding window approach to maximize the probability of context words given a center word.

#### **Embedding and Semantic Relationships**

* Through the training process, Word2Vec places words with similar meanings in proximity within the high-dimensional vector space. For example, ‘bread’ and ‘croissant’ would have closely aligned vectors, just as ‘woman’, ‘king’, and ‘man’ would demonstrate meaningful relationships through vector arithmetic.

1. کلمات با معانی متعدد(polysemy) می توانند برای جاسازی کلمات چالش هایی ایجاد کنند، زیرا یک نمایش برداری واحد ممکن است به اندازه کافی همه معانی آن را در بر نگیرد. در واقع با یک بردار نمی توان چندین بردار(چند معنی) را به شکل specific بازنمایی کنیم و این موضوع باعث می شود که این چندین معنی در مجموعه های آموزشی ترکیب شوند و بازنمایی عمومی تر که ظرافت های زبانی را در نظر نمیگیرند ایجاد کند.

برخی از رویکردها برای رفع این مشکل عبارتند از:

* Polysemy-aware embeddings: تکنیک‌هایی که به صراحت معانی متعدد یک کلمه را در فضای جاسازی آن مدل‌سازی می‌کنند. (Use different vectors for the multiple meanings by maintaining a dictionary of sense-specific vectors.)
* Contextual embeddings: مدل‌هایی مانند ELMo و BERT بازنمایی‌های کلمه‌ای را بر اساس context خود در یک جمله تولید می‌کنند و به آنها امکان می‌دهد معانی مختلف یک کلمه را در context ‌های مختلف دریافت کنند. در واقع آنها context-sensitive هستند یعنی آنها بازنمایی های پویای کلمه(dynamic word representations) را بر اساس زمینه(context) ارائه می دهند که منجر به درک دقیق تری از زبان می شود.   
  در زیر نحوه تولید embedding در BERT به طور خلاصه آمده است:  
  Use a pre-trained model and generate embeddings for desired words (rather than using pre-trained embeddings as in Word2Vec) since they are tailor-fit based on the context.

چالش هایی که این کلمات در بازنمایی های **Co-occurrence Based/Static Embedding Techniques** نظیر **Word2Vec and GloVe** ایجاد میکنند که به دلیل **Non-Contextualized Nature** است به شرح زیر هستند:

* **Single Vector Per Word**: Word2Vec and GloVe assigns a unique vector to each word, which remains static regardless of the word’s varying context in different sentences. This results in a representation that cannot dynamically adapt to different usages of the same word.
* **Combination of Contexts**: In cases where a word like “bank” appears in multiple contexts (“river bank” vs. “financial bank”), Word2Vec and GloVe does not generate distinct embeddings for each scenario. Instead, it creates a singular, averaged representation that amalgamates all the contexts in which the word appears, leading to a generalized semantic representation.
* **Lack of Disambiguation**: The model’s inability to differentiate between the multiple meanings of polysemous words means that words like “bank” are represented by a single vector, irrespective of the specific meaning in a given context.

همچنین در لینکی که داخل مراجع هست توضیحات بیشتری در این مورد آمده است که به شرح زیر است:

**Count-Based Techniques like TF-IDF and BM25 Does Not Account for Polysemy:** Words with multiple meanings (polysemy) are treated the same regardless of their context. For example, the word “bank” would have the same representation in “river bank” and “savings bank”, even though it has different meanings in these contexts.

However, it’s important to note that while **Word2Vec and GloVe(Co-occurrence Based/Static Embedding Techniques)** captures many semantic relationships, it also has limitations. For example, it doesn’t capture polysemy well (the same word having different meanings in different contexts) and sometimes the relationships it learns are more syntactic than semantic. More advanced models like BERT and GPT have since been developed to address some of these limitations.

Contextual Embeddings consider the order and context of words:

* ELMo (Embeddings from Language Models): Generates contextual embeddings from the internal states of a bi-directional LSTM.
* BERT (Bidirectional Encoder Representations from Transformers) Embeddings: Provides contextual embeddings based on the entire context of word usage.

1. یکی از رویکردهای رایج برای مدیریت کلمات خارج از واژگان، استفاده از FastText به جای Word2Vec یا GloVe است. FastText قادر است کلمات خارج از واژگان را با نگاه کردن به اطلاعات زیر کلمه (character ngrams) جاسازی کند. Gensim همچنین دارای یک پیاده سازی FastText است که استفاده از آن بسیار آسان است. روش دیگر جایگزینی کلمات نادر با توکن UNK (unknown) در زمان آموزش است. هنگامی که سیستم UNK را تولید می کند، UNK را با کلمه منبع تراز(align) کنیم، و این را با یک روش back-off ترجمه کنیم. یکی دیگر از ترفندهای رایج، به ویژه هنگام کار با راه حل های مبتنی بر جاسازی کلمه، جایگزینی کلمه با یک کلمه نزدیک از نوعی فرهنگ لغت مترادف است.

به بیان دیگر:

* واژه های خارج از واژگان(Out-of-vocabulary words) آنهایی هستند که در واژگان مورد استفاده برای آموزش word embedding وجود ندارند.
* یکی از رویکردهای رایج برای رسیدگی به کلمات OOV، اختصاص یک نشانه "ناشناخته" ویژه به آنها در طول آموزش است. ("unknown" token)
* برای ایجاد جاسازی برای کلمات OOV، می توانیم از تکنیک های زیر استفاده کنیم:
* Subword embeddings: شکستن کلمات OOV به واحدهای فرعی کوچک‌تر (به عنوان مثال، character n-grams or morphemes) و ترکیب جاسازی‌های آنها بر اساس جاسازی‌های این واحدهای زیر کلمه ای. (مانند FastText)
* Retrofitting: استفاده از اطلاعات معنایی از جاسازی های از پیش آموزش دیده برای ایجاد جاسازی برای کلمات OOV با اعمال شباهت با کلمات شناخته شده.
* Zero-shot learning: استنباط جاسازی برای کلمات OOV بر اساس شباهت آنها به کلمات شناخته شده در فضای جاسازی، حتی اگر در داده های آموزشی وجود نداشته باشند.
* استفاده از character embedding به جای word embedding.

جواب claude هم خارج از لطف نیست:

Out-of-vocabulary (OOV) words that were unseen in the training data cannot get pre-trained embeddings generated. Some ways to handle this:

1. Mapping to UNK (unknown) vector
2. Character-level models can generate representations for OOV words
3. Using pretrained subword embeddings like FastText
4. On-the-fly generation of embeddings for OOV words through knowledge transfer from similar words
5. Fine-tuning embeddings on the target task data

A common method is to train a character-level model that can compositionally generate embeddings for unseen words from their character n-grams. This embeds words based on their subword information.

همچنین توضیحات بیشتر راجع به FastText در ادامه آمده است:

#### **Overview**

* Proposed in [Enriching Word Vectors with Subword Information](https://arxiv.org/abs/1607.04606) by Bojanowski et al. (2017), fastText is an advanced word representation and sentence classification library developed by Facebook’s AI Research (FAIR) lab. It’s primarily used for text classification and word embeddings in NLP. fastText differs from traditional word embedding techniques through its unique approach to representing words, which is particularly beneficial for understanding morphologically complex languages or handling rare words.
* Specifically, fastText’s innovative approach of using subword information makes it a powerful tool for a variety of NLP tasks, especially in dealing with languages that have extensive word forms and in situations where the dataset contains many rare words. By learning embeddings that incorporate subword information, fastText provides a more nuanced and comprehensive understanding of language semantics compared to traditional word embedding methods.
* Here’s a detailed look at fastText with an example.

#### **Core Features of FastText**

1. **Subword Information**: Unlike traditional models that treat words as the smallest unit for training, fastText breaks down words into smaller units - subwords or character n-grams. For instance, for the word “fast”, with a chosen n-gram range of 3 to 6, some of the subwords would be “fas”, “fast”, “ast”, etc. This technique helps in capturing the morphology of words.
2. **Handling of Rare Words**: Due to its subword approach, fastText can effectively handle rare words or even words not seen during training. It generates embeddings for these words based on their subword units, allowing it to infer some meaning from these subcomponents.
3. **Efficiency in Learning Word Representations**: fastText is efficient in learning representations for words that appear infrequently in the corpus, which is a significant limitation in many other word embedding techniques.
4. **Applicability to Various Languages**: Its subword feature makes it particularly suitable for languages with rich word formations and complex morphology, like Turkish or Finnish.
5. **Word Embedding and Text Classification**: fastText can be used both for generating word embeddings and for text classification purposes, providing versatile applications in NLP tasks.

#### **Example of FastText**

* Consider the task of building a sentiment analysis model using word embeddings for an input sentence like “The movie was breathtakingly beautiful”. In traditional models like Word2Vec, each word is treated as a distinct unit, and if words like “breathtakingly” are rare in the training dataset, the model may not have a meaningful representation for them.
* With fastText, “breathtakingly” is broken down into subwords (e.g., “breat”, “eathtaking”, “htakingly”, etc.). fastText then learns vectors for these subwords. When computing the vector for “breathtakingly”, it aggregates the vectors of its subwords. This approach allows fastText to handle rare words more effectively, as it can utilize the information from common subwords to understand less common or even out-of-vocabulary words.

مراجع:

<https://aman.ai/primers/ai/word-vectors/>   
<https://www.geeksforgeeks.org/pre-trained-word-embedding-using-glove-in-nlp-models>   
[python - How to handle words that are not in word2vec's vocab optimally - Stack Overflow](https://stackoverflow.com/questions/54709178/how-to-handle-words-that-are-not-in-word2vecs-vocab-optimally)  
<https://chat.openai.com/>   
<https://bard.google.com/>   
<https://claude.ai/chats>

## سوال دوم

برای ساختن ماتریس co-occurrence برای متن داده شده با اندازه پنجره 2، باید شمارش کنیم که هر کلمه چند بار با هر کلمه دیگری در پنجره مشخص شده اتفاق می افتد.

متن این است: "I love computer science and I love NLP even more."

بیایید ابتدا واژگانی از کلمات منحصر به فرد ایجاد کنیم:

Vocabulary = ['I', 'love', 'computer', 'science', 'and', 'NLP', 'even', 'more']

ماتریس co-occurrence یک ماتریس مربعی با ابعاد برابر با اندازه vocabulary خواهد بود.

در ادامه ماتریس co-occurrence با اندازه پنجره 2 آمده است:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | I | love | computer | science | and | NLP | even | more |
| I | 0 | 2 | 1 | 1 | 1 | 1 | 0 | 0 |
| love | 2 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| computer | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| science | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| and | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| NLP | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| even | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| more | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

ماتریس co-occurrence نشان می‌دهد که کلمات چقدر در متن کلمات دیگر(context) در اندازه پنجره داده شده ظاهر می‌شوند. از این اطلاعات آماری می توان برای آموزش مدل های جاسازی کلمه مانند GloVe یا Word2Vec استفاده کرد.

توضیحات:

* اندازه پنجره 2 کلمه مرکزی و دو کلمه قبل و دو کلمه بعد از آن را در نظر می گیرد.
* ما همزمانی یک کلمه با خودش (مورب) را در نظر نمی گیریم.
* مقدار در هر ردیف i، ستون j نشان دهنده تعداد دفعاتی است که کلمه i در کنار کلمه j در پنجره اندازه 2 ظاهر می شود.

مراجع:

<https://chat.openai.com/>   
<https://bard.google.com/>   
<https://claude.ai/chats>

## سوال سوم

داخل نوتبوک پیاده سازی شده است.  
مراجع:

<https://chat.openai.com/>   
<https://bard.google.com/>   
<https://claude.ai/chats>

## سوال چهارم

داخل نوتبوک پیاده سازی شده است.

مراجع:

<https://chat.openai.com/>   
<https://bard.google.com/>   
<https://claude.ai/chats>

پایان