Włożenie słów w przestrzenie wektorowe

Word embeddings

Aleksandra Chazy, Paulina Helwin, Dagmara Skowron

May 2026

Spis treści

Wstęp

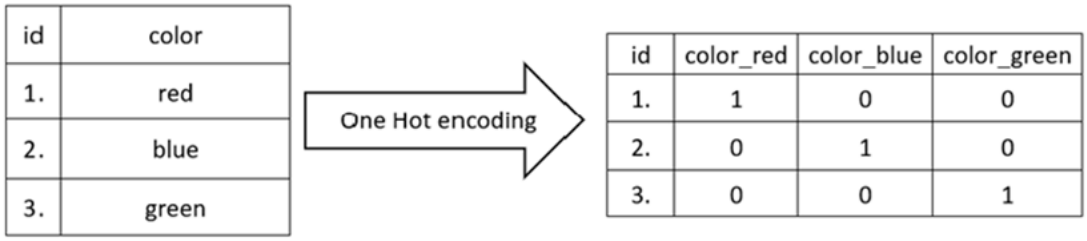
Celem pracy jest przedstawienie pojęcia word embeddings oraz wyjaśnienie, w jaki sposób słowa i teksty mogą być reprezentowane w komputerze za pomocą wektorów liczbowych. Praca ma pokazać, dlaczego tradycyjne metody reprezentacji tekstu, takie jak one-hot encoding, Bag of Words i TF-IDF, są niewystarczające w analizie znaczenia oraz kontekstu słów. Omówione zostaną klasyczne modele word embeddings, takie jak Word2Vec, GloVe i FastText, a także nowoczesne modele językowe, między innymi BERT i GPT. Dodatkowo praca przedstawia przykłady zastosowania embeddingów oraz prostą implementację w języku Python, pokazującą praktyczne wykorzystanie reprezentacji wektorowych.

1 Problem reprezentacji tekstu w komputerze

Jednym z podstawowych problemów przetwarzania języka naturalnego jest sposób przedsawiania tektu w formie zrozumiałej dla komputera. Człowiek naturalnie rozumie kontekst i znaczenie danego wyrażenia, natomiast komputer nie zna znaczenia słowa dopóki nie zotsanie ono zapisane w formie, którą umie przetwarzać, czyli jako liczba, zbiór liczb lub wektor. Większość algorytmów uczenia maszynowego działa na danych liczbowych. Surowy tekst nie nadaje się do takich operacji, dlatego rozwijano kolejne metody reprezentacji tekstu, która pozwoli komputerowi wykonywać obliczenia na języku.

1.1 One-hot encoding

One-hot encoding to jedna z najprostszych metod reprezentowania słów. Polega ona na stworzeniu słownika wszystkich słów występujących w zbiorze danych. Następnie każdemu słowu przypisuje się oddzielny indeks. Słowo jest reprezentowane jako wektor, który ma same zera i jedną jedynkę identyfikującą dane słowo. Sam wektor jest długości równej ilości słów w słowniku.



Rysunek 1: Schemat procesu One-Hot Encoding (wersja uproszczona).

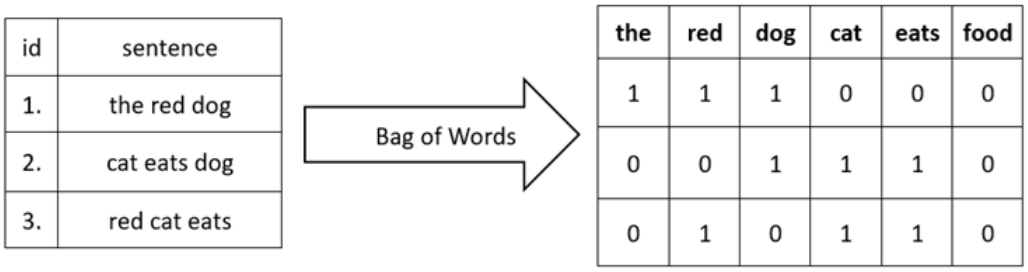
One-hot encoding pozwala jednoznacznie zakodować każde słowo, ale nie przekazuje informacji o jego znaczeniu. Wektory są od siebie niezależne, więc komputer nie wie, które słowa są podobne, a które całkowicie różne.

1.2 Bag of Words

Ta metoda reprezentuje cały zbiór, zamiast pojdyńczych słów. Polega na zliczaniu ile razy każde słowo ze słownika występuje w danym tekście, ignorując kolejność ich występowania.

Najpierw powstaje słownik wszystkich słów występujących w danym zbiorze. Następnie tekst jest zamieniany na wektor liczb, w którym każda pozycja odpowiada jednemu

słowu ze słownika. Wartość na danej pozycji odpowiada ilości wystąpień danego słowa w dokumencie. Metoda ta była i nadal bywa stosowana do klasyfkacji dokumentów i analizy opinii.



Rysunek 2: Schemat procesu Bag of Words (wersja uproszczona).

1.3 TF-IDF

Kolejną metodą jest Term Frequency-Inverse Document Frequency. Nadaje ona większą wagę słowom, które często pojawiają się w danym dokumencie, ale rzadziej występują w innych. Składa się z dwóch części: TF, która określa, jak czesto dane słowo występuje w tekście oraz IDF, która mówi o tym, jak rzadko słowo występuje w danym zbiorze, czyli obniża wagę słów np. spójników. Podobnie jak Bag of Words nie uwzględnia kolejności występowania słów oraz ich znaczenia. Słowo jest reprezentowane przez liczbę:

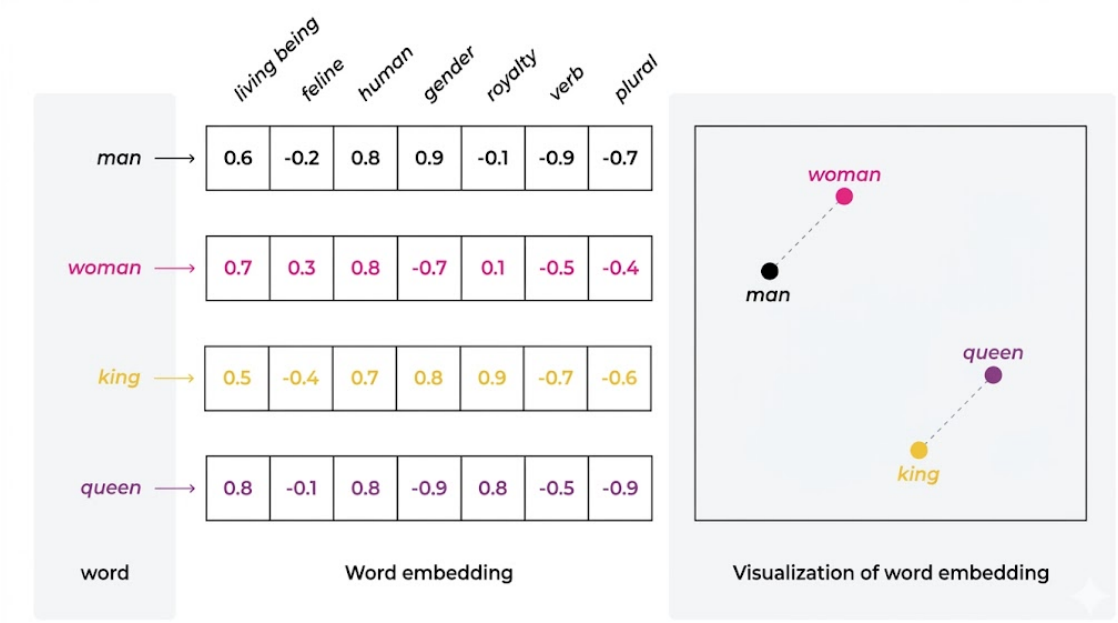
$$TF - IDF = TF \cdot IDF$$

Mimo, że TF-IDF jest bardziej zaawansowaną metodą niż Bag of Words, nadal posiada istotne ograniczenia. Problemem jest brak informacji o kontekście oraz brak analizy relacji między słowami. W rezultacie mogą niepoprawnie interpretować zdania, kóre zawierają te same słowa, ale mają inne znaczenie. Ograniczenia te doprowadziły do rozwoju bardziej zaawansowanych metod reprezentacji tekstu, określanych jako word embeddings.

2 Word embeddings

2.1 Słowa jako wektory

W metodach word embeddings każde słowo jest przedstawiane jako wektor [1]. Taka reprezentacja pozwala komputerowi wykonywać działania matematyczne na słowach i porównywać je ze sobą. Poniżej przedstawiono ilustrację opisującą shcemat procesu word embedding.



Rysunek 3: Schemat procesu Word embedding (wersja uproszczona).

Wartości w tych wektorach nie są nadawane ręcznie. Model uczy się ich automatycznie na podstawie dużych zbiorów tekstów. Jeżeli dwa słowa często występują w podobnych kontekstach, ich wektory będą do siebie podobne. Dzięki temu słowa *woman* i *man* mogą znajdować się blisko siebie w przestrzeni wektorowej.

2.2 Znaczenia kontekstów word embeddings

Kluczowym elementem word embeddings jest kontekst. Oznacza to, że model analizuje całe zdanie, w którym dane słowo występuje, wyłapując przy tym jego sens i kontekst. Wyrazy o podobnym znaczeniu będą miały zbliżoną interpretację wektorową. Jest to ważna różnica w porównaniu z metodami przedstawionymi w poprzednim rozdziale.

Model uczy się na podstawie dużych zbiorów tekstu, analizując które słowa często pojawiają się razem lub w podobnym otoczeniu. Dzięki temu potrafi rozpoznawać relacje między słowami.

Przykład:

- Zdanie 1: W moim domu zalęgły się myszy.
- Zdanie 2: Zapomniałem dzisiaj do pracy zabrać mojej myszy do komputera.

W obu zdaniach pada słowo mysz, ale w każdym ma ono inne znaczenie. Metody word embeddings są w stanie wyłapywać takie różnice.

Skoro word embeddings przedstawiają słowa jako wektory, pojawia się pytanie, jak takie wektory można analizować i wizualizować. Wektory embeddingowe mają zazwyczaj wiele wymiarów, więc ciężko przedstawić je na wykresie. Z tego powodu stosuje się redukcję wymiaru, która pozwala uprościć dane i pokazać relacje między słowami w bardziej czytelnej formie.

2.3 Redukcja wymiaru

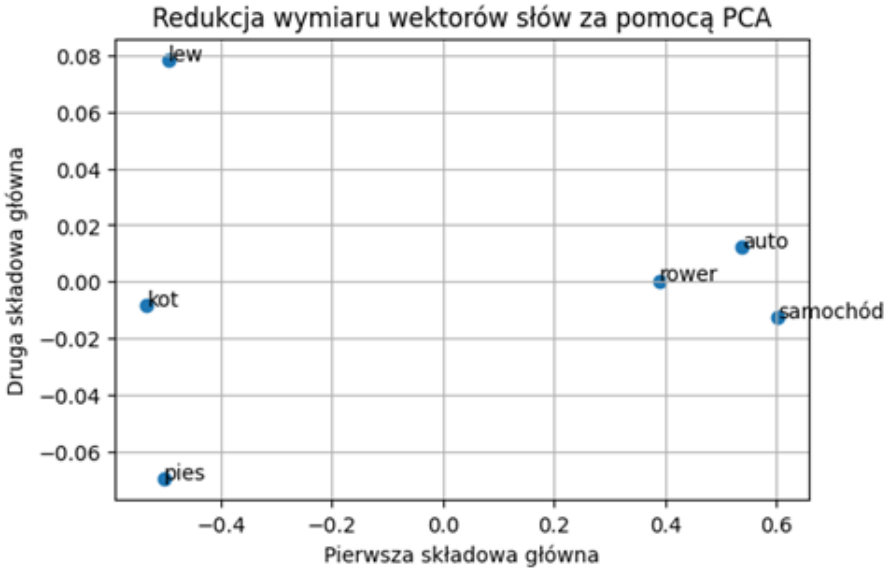
Redukcja wymiaru to proces zmniejszania liczby cech opisujących dane przy możliwie jak najmniejszej utracie istotnych informacji. W przypadku word embeddings proces ten polega na przekształceniu danych z przestrzeni o dużej liczbie wymiarów do przestrzeni prostszej, najczęściej dwu- lub trójwymiarowej.

Zmniejszenie wymiaru ma kilka ważnych zastosowań, przede wszytkim pozwala wizualizować embeddingi, czyli pokazywać słowa jako punkty na wykresie. Może także pomóc w analizie dużych zbiorów danych, ponieważ mniejsza liczba wymiarów oznacza prostszą interpretację. Jedną z popularnych metod jest PCA, czyli analiza składowych głównych [8]. Metoda ta szuka takich kierunków w danych, które zachowują najwięcej informacji o ich zmienności. Na ich podstawie tworzy krótszy opis wektorów. Pozwala to przedstawić dane np. na płaskim wykresie dwuwymiarowym.

Poniższy przykład pokazuje prostą wizualizację redukcji wymiaru wektorów słów z wykorzystaniem języka Python. Najpierw zdefiniowano przykładowe czterowymiarowe wektory dla kilku słów. Następnie za pomocą metody PCA wymiar wektorów zmniejszono do dwóch, aby można było przedstawić je na wykresie.

Implementacja:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

words = ["kot", "pies", "lew", "samochod", "auto", "rower"]
vectors = np.array([
[0.12, 0.45, 0.30, 0.80],
[0.10, 0.40, 0.35, 0.78],
[0.20, 0.50, 0.33, 0.82],
[0.90, 0.10, 0.75, 0.20],
[0.88, 0.12, 0.70, 0.25],
[0.76, 0.18, 0.65, 0.30]
])

pca = PCA(n_components=2)
reduced_vectors = pca.fit_transform(vectors)

plt.figure(figsize=(6, 4))
plt.scatter(reduced_vectors[:, 0], reduced_vectors[:, 1])

for i, word in enumerate(words):
plt.text(reduced_vectors[i, 0], reduced_vectors[i, 1], word)

plt.xlabel("Pierwsza skladowa glowna")
plt.ylabel("Druga skladowa glowna")
plt.grid(True)
plt.show()

```



Rysunek 4: Wizualizacja redukcji wymiaru.

W przykładzie wykorzystano słowa: „kot", „pies", „lew", „samochód", „auto" i „rower". Każdemu z nich przypisano czterowymiarowy wektor. Następnie zastosowano algorytm PCA, który przekształcił te wektory do przestrzeni dwuwymiarowej. Dzięki temu możliwe było przedstawienie słów jako punktów na wykresie.

Na wykresie można obserwować słowa jako punkty. W dobrze dobranych embeddingach słowa podobne znaczeniowo, takie jak „kot", „pies" i „lew" tworzą jedną grupę, natomiast słowa, takie jak „samochód", „auto" i „rower" znajdują się bliżej siebie. Pokazuje to, że redukcja wymiaru może pomóc w wizualnej analizie relacji między słowami.

3 Klasyczne modele word embeddings

3.1 Word2Vec

Pierwszy model, który sprawdził się na dużą skalę [3]. Ideą jest uczenie wektorów słów tak, żeby dobrze przewidywały kontekst oraz wykorzystywanie prostych sieci neuronowych.

Od strony technicznej Word2Vec jest płytką siecią neuronową, składającą się z warstwy wejściowej, jednej warstwy ukrytej (bez funkcji aktywacji, a więc operującej na transformacji liniowej) oraz warstwy wyjściowej z funkcją Softmax. Ostateczne reprezentacje wektorowe słów (embeddingi) są w rzeczywistości wyciągane bezpośrednio z macierzy wag warstwy ukrytej po zakończeniu procesu trenowania sieci.

Dodatkowo, algorytm ten stosuje podczas treningu optymalizację zwaną *Subsamplingiem*. Często występujące słowa (takie jak spójniki czy przyimki) niosą znacznie mniej informacji semantycznych niż słowa rzadkie. Word2Vec odrzuca je z pewnym prawdopodobieństwem obliczanym na podstawie częstotliwości ich występowania, co znacznie przyspiesza trening i poprawia jakość reprezentacji rzadszych wyrazów.

3.1.1 CBOW (Continuous Bag of Words)

W modelu CBOW analizowane jest otoczenie danego słowa. Model otrzymuje kilka słów znajdujących się przed i po słowie docelowym w formie wektorów one-hot, a następnie

poprzez warstwę ukrytą próbuje przewidzieć brakujące słowo w warstwie wyjściowej. Matematycznie, CBOW optymalizuje prawdopodobieństwo wystąpienia słowa docelowego pod warunkiem znanego kontekstu. Funkcję celu można zapisać jako maksymalizację logarytmu prawdopodobieństwa: $\log p(w_t|w_{t-c}, \ldots, w_{t+c})$. Dzięki temu uczy się, jakie wyrazy zwykle występują razem i które słowa pełnią podobną funkcję w zdaniu.

Wady:

- może gorzej odwzorowywać rzadkie słowa,
- uśrednia wektory kontekstu przed przewidywaniem, przez co traci część informacji o precyzyjnych relacjach,
- nie rozwiązuje problemu wieloznaczności (jedno słowo = jeden wektor).

3.1.2 Skip-gram

W modelu Skip-gram punktem wyjścia jest pojedyncze słowo. Odwraca on logikę CBOW – model uczy się przewidywać na podstawie słowa centralnego, jakie inne słowa (kontekst) mogą pojawić się w jego pobliżu. Funkcją celu w tym wariancie jest maksymalizacja średniego logarytmu prawdopodobieństwa przewidzenia słów kontekstowych na podstawie słowa centralnego w_t:

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

Standardowe obliczanie funkcji Softmax dla pełnego słownika na warstwie wyjściowej jest skrajnie nieefektywne obliczeniowo (wymaga sumowania po setkach tysięcy wyrazów w każdym kroku). Dlatego w praktyce stosuje się aproksymację zwaną **Negative Sampling** [4].

Zamiast aktualizować wagi dla wszystkich słów, Negative Sampling modyfikuje problem do klasyfikacji binarnej za pomocą regresji logistycznej. Sieć jest trenowana tak, aby odróżnić prawdziwą parę słów (słowo centralne i jego rzeczywisty kontekst) od kilku lub kilkunastu fałszywych par, stworzonych ze słowa centralnego oraz tzw. słów negatywnych (losowo dobranych wyrazów ze słownika, które nie występują w jego kontekście). Znacznie redukuje to koszty operacji na macierzach i pozwala modelowi skupić się na właściwych relacjach.

Wady:

- jest wolniejszy i bardziej kosztowny obliczeniowo niż CBOW,
- nadal tworzy jeden stały wektor dla każdego słowa (brak zrozumienia głębokiego kontekstu zdania),
- bardzo polega na odpowiednim doborze hiperparametrów (np. wielkości okna c).

Implementacja:

```
import nltk
from nltk.corpus import brown
from gensim.models import Word2Vec

nltk.download("brown")
sentences = brown.sents()

model = Word2Vec(
sentences,
vector_size=100,
window=5,
min_count=5,
sg=1,
epochs=10
)
print(model.wv.most_similar("city"))

```

W powyższej implementacji wykorzystano korpus Brown dostępny w bibliotece NLTK oraz model Word2Vec z biblioteki gensim. Celem implementacji było pokazanie, w jaki sposób skorzystać z modelu Word2Vec [2]. Każde zdanie jest listą słów, dzięki czemu może zostać bezpośrednio przekazane do modelu.

Parametr sg=1 oznacza użycie wariantu Skip-gram. Gdy zastąpimy jedynkę zerem, model będzie konstruowany metodą CBOW.

Na końcu użyto funkcji most similar(„city"), która zwraca słowa najbardziej podobne do słowa „city" według wytrenowanego modelu.

Tabela 1: Słowa najbardziej podobne do słowa „city" według modelu Word2Vec

Słowo podobne	**Wartość podobieństwa**
theater	0.702
county	0.686
studio	0.685
district	0.685

Powyższa tabela pokazuje, które słowa według wytrenowanego modelu są najbardziej zbliżone znaczeniowo do słowa „city" (to co zwraca zastosowana funkcja most similar). Wyniki okrojono do czterech słów z największym prawdopodobieństwem.

3.2 GloVe

GloVe, czyli Global Vectors for Word Representation to klasyczny model word embeddings, który tworzy reprezentacje wektorowe słów na podstawie statystyk współwystępowania wyrazów w dużym zbiorze tekstów [5]. Model ten został opracowany jako alternatywa dla metod takich jak Word2Vec i łączy podejście statystyczne z ideą reprezentacji wektorowych. Podstawowa różnica między GloVe a Word2Vec polega na sposobie uczenia modelu. Word2Vec analizuje kontekst słowa w jego otoczeniu, natomiast GloVe wykorzystuje informacje z całego zbioru. Oznacza to, że model bierze pod uwagę, jak często różne pary słów występują razem w całym zbiorze tekstów. W celu utworzenia modelu konstruowana

jest macierz, która zawiera informacje o tym, ile razy dane słowo pojawiło się w kontekście innego słowa. Na podstawie tych danych model uczy się takich wektorów, które odzwierciedlają zależności między słowami. Przykładowo słowa związane z miastem, takie jak „city", „district", „street" czy „building", mogą często pojawiać się w podobnych kontekstach. GloVe wykorzystuje takie zależności, aby umieścić ich wektory blisko siebie.

Wady:

- wymaga dużych zbiorów tekstu,
- tworzy statyczne embeddingi,
- nie uwzględnia zmiany znaczenia słowa w zależności od zdania,
- może mieć problem z nowymi słowami spoza słownika.

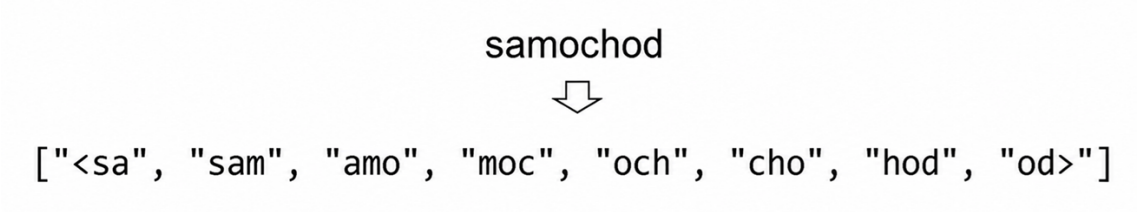
3.3 FastText

FastText to klasyczny model word embeddings, który można uznać za rozwinięcie podejścia Word2Vec [6]. Podobnie jak Word2Vec, służy do tworzenia reprezentacji wektorowych słów na podstawie ich kontekstu występowania w tekście. Główna różnica polega jednak na tym, że FastText nie traktuje słowa wyłącznie jako jednej całości, ale dzieli je na mniejsze fragmenty. W modelu FastText słowo reprezentowane jest jako zbiór krótszych części, najczęściej tzw. n-gramów znakowych. Oznacza to, że wyraz jest analizowany na poziomie fragmentów liter, a nie tylko jako pełne słowo. Dzięki temu model może lepiej radzić sobie ze słowami rzadkimi, odmianami wyrazów.

Wady:

- może być bardziej kosztowny obliczeniowo,
- nadal tworzy embeddingi statyczne,
- nie rozwiązuje w pełni problemu wieloznaczności,
- wymaga odpowiednio dużych danych treningowych.

Przykład:



Rysunek 5: Wizualizacja działania metody FastText.

4 Nowoczesne reprezentacje języka

4.1 BERT

BERT, czyli Bidirectional Encoder Representations from Transformers to jeden z najważniejszych modeli językowych wykorzystywanych we współczesnym przetwarzaniu języka naturalnego [7]. Najważniejszą cechą modelu BERT jest dwukierunkowe analizowanie tekstu. Oznacza to, że model bierze pod uwagę zarówno słowa znajdujące się przed analizowanym wyrazem, jak i słowa występujące po nim. Dzięki temu może lepiej rozpoznawać znaczenie słowa w konkretnym zdaniu.

BERT różni się od modeli takich jak Word2Vec, GloVe czy FastText tym, że nie przypisuje słowu jednej stałej reprezentacji. W klasycznych embeddingach dane słowo ma zawsze ten sam wektor, niezależnie od zdania. W BERT reprezentacja słowa jest tworzona dynamicznie, czyli zależy od tekstu, w którym słowo zostało użyte.

Model BERT jest trenowany na dużych zbiorach tekstowych. Proces ten polega na tym, że część słów w zdaniu zostaje ukryta, a model próbuje je odgadnąć na podstawie pozostałych słów. Dzięki temu uczy się zależności między wyrazami oraz znaczenia wynikającego z kontekstu.

4.2 GPT i modele LLM

Generative Pre-trained Transformer, to rodzina nowoczesnych modeli językowych opartych na architekturze Transformer. Modele GPT należą do grupy dużych modeli językowych, określanych jako LLM (Large Language Models). Ich głównym zadaniem jest przewidywanie kolejnych słów lub fragmentów tekstu na podstawie wcześniejszego kontekstu.

GPT działa głównie w sposób generatywny. Oznacza to, że model tworzy dalszą część wypowiedzi na podstawie tekstu, który otrzymał wcześniej. Dzięki temu może generować odpowiedzi, streszczenia , tłumaczenia, kod komputerowy czy inne formy tekstu.

4.3 Różnice między BERT a GPT

Oba modele są oparte na architekturze Transformer, ale różnią się sposobem działania oraz głównym zastosowaniem. BERT jest nastawiony głównie na rozumienie testu, a GPT na jego generowanie. Ważną różnicą jest również fakt, że BERT działa dwukierunkowo, natomiast GPT analizuje konekst tylko w jedną stronę. Modele różnią się również metodą trenowania. BERT uczy się przewidywać ukryte słowa na podstawie całego kontekstu zdania, natomiast GPT uczy się przewidywać kolejne słowo na podstawie wcześniejszego tekstu. Dlatego BERT lepiej sprawdza się w rozumieniu tekstu, a GPT w jego generowaniu.

5 Zastosowania word embedding

5.1 Wyszukiwanie informacji

Jednym z zastosowań word embeddings jest wyszukiwanie informacji. Tradycyjne wyszukiwarki opierały się głównie na dopasowaniu słów kluczowych, czyli szukały dokumentów zawierających dokładnie te same wyrazy, które podał użytkownik. Takie podejście ma jednak ograniczenia, ponieważ nie zawsze uwzględnia znaczenie słów.

Word embeddings pozwalają wyszukiwać treści na podstawie podobieństwa znaczeniowego. System może rozpoznać, że słowa „auto” i „samochód” są do siebie podobne, mimo że są zapisane inaczej. Zastosowanie word embeddings w wyszukiwaniu informacji zwiększa

trafność wyników i pozwala lepiej obsługiwać zapytania użytkowników. Dzięki reprezentacjom wektorowym wyszukiwarka może działać bardziej elastycznie i uwzględniać sens zapytania, a nie tylko dosłowne występowanie słów.

5.2 Tłumaczenie maszynowe

Kolejnym zastosowaniem jest tłumaczenie maszynowe. Modele tłumaczące muszą rozpoznawać znaczenie słów w kontekście, ponieważ jedno słowo może mieć różne tłumaczenia w zależności od zdania. Reprezentacje wektorowe pomagają modelowi określić, które słowa lub wyrażenia są do siebie podobne znaczeniowo. Szczególnie ważne są tutaj nowoczesne modele kontekstowe, ponieważ potrafią analizować znaczenie słowa na podstawie całego zdania.

5.3 Klasyfikacja tekstu i analiza opinii

Word embeddings są wykorzystywane także w klasyfikacji tekstu. Polega ona na przypisaniu tekstu do odpowiedniej kategorii, na przykład rozpoznaniu, czy wiadomość jest spamem, określeniu tematu artykułu albo przypisaniu dokumentu do właściwego działu. Dzięki reprezentacjom wektorowym model może analizować nie tylko pojedyncze słowa, ale również ogólny sens wypowiedzi.

Podobnie działa analiza opinii. Jej celem jest określenie, czy dana wypowiedź ma charakter pozytywny, negatywny czy neutralny. Może być stosowana na przykład do analizy recenzji produktów, komentarzy internetowych lub opinii klientów. Embeddingi pomagają modelowi lepiej rozpoznawać znaczenie słów w kontekście, co poprawia skuteczność takiej analizy.

5.4 Chatboty i systemy rekomendacji

Wykorzystuje się je także chatbotach. Dzięki nim model może lepiej rozumieć pytania użytkownika, analizować kontekst rozmowy i generować odpowiedzi pasujące do wcześniejszej wypowiedzi. Nowoczesne modele, takie jak GPT, korzystają z reprezentacji języka, aby prowadzić bardziej naturalną rozmowę.

Embeddingi stosuje się również w systemach rekomendacji. Jeżeli opisy produktów, filmów, artykułów lub zainteresowania użytkownika zostaną przedstawione jako wektory, system może porównywać ich podobieństwo. Dzięki temu możliwe jest polecanie treści podobnych do tych, które użytkownik wcześniej oglądał, czytał lub oceniał pozytywnie.

Podsumowanie

W pracy omówiono problem reprezentacji tekstu w komputerze oraz przedstawiono metody zamiany słów na postać liczbową. Proste techniki, takie jak one-hot encoding, Bag of Words i TF-IDF, pozwalają analizować tekst, ale nie uwzględniają dobrze znaczenia słów ani ich kontekstu. Word embeddings rozwiązują część tych problemów, ponieważ przedstawiają słowa jako wektory liczbowe, dzięki czemu możliwe jest porównywanie ich podobieństwa. W pracy opisano klasyczne modele, takie jak Word2Vec, GloVe i FastText, a także nowoczesne modele językowe, takie jak BERT i GPT. Przedstawione przykłady pokazują, że reprezentacje wektorowe są ważnym narzędziem w przetwarzaniu języka naturalnego. Znajdują zastosowanie między innymi w wyszukiwaniu informacji, tłumaczeniu maszynowym, chatbotach, klasyfikacji tekstu i systemach rekomendacji.

Bibliografia

[1] TensorFlow, *Word embeddings*, https://www.tensorflow.org/text/tutorials/word_embeddings

[2] TensorFlow, *Word2Vec tutorial*, https://www.tensorflow.org/text/tutorials/word2vec

[3] Mikolov, T., Chen, K., Corrado, G., Dean, J., *Efficient Estimation of Word Representations in Vector Space*, 2013, https://arxiv.org/abs/1301.3781

[4] Goldberg, Y., Levy, O., *word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method*, 2014, https://arxiv.org/abs/1402.3722

[5] Stanford NLP Group, *GloVe: Global Vectors for Word Representation*, https://nlp.stanford.edu/projects/glove/

[6] Bojanowski, P., Grave, E., Joulin, A., Mikolov, T., *Enriching Word Vectors with Subword Information*, 2017, https://arxiv.org/abs/1607.04606

[7] Devlin J., Chang M.-W., Lee K., Toutanova K., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2018. https://arxiv.org/abs/1810.04805

[8] Scikit-learn Documentation, *PCA – sklearn.decomposition.PCA*. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html