MATNLARNI RAQAMLASHTIRISH VA MASHINALI O‘QITISHDA WORD2VEC METODINING AHAMIYATI

Botir Elov[1], Alayev Ruhillo[2], Alayev Narzullo[3]

[1]Alisher Navoiy nomidagi Toshkent davlat o‘zbek tili va adabiyoti universiteti. E-pochta: elov@navoiy-uni.uz
[2]Mirzo Ulug‘bek nomidagi O‘zbekiston Milliy universiteti. E-pochta: mr.ruhillo@gmail.com
[3]Alisher Navoiy nomidagi Toshkent davlat o‘zbek tili va adabiyoti universiteti tayanch doktoranti.

K E Y W O R D S

Matnlarini qayta ishlash, Word2Vec, so‘zlarni joylashtirish, word embedding, tokenizatsiya, o‘quv ma'lumotlari, one-hot encoding modeli, matnlarni raqamlashtirish, mashinali o‘qitish.

A B S T R A C T

Tabiiy tilni qayta ishlash (NLP) – tilshunoslik, kompyuter fanlari, sun'iy intellektning kompyuter va insonning o‘zaro ta'siri bilan bog‘liq bo‘lgan bo‘limi. U asosan, tabiiy tilni qayta ishlash va baholash uchun metod, algoritm va axborot tizimlarini loyihalash, ishlab chiqish masalalari bilan shug‘ullanadi. Hozirda NLP usullari vositasida katta hajmdagi til korpuslari va millionlab veb-sahifalar bir soniya ichida tahlil qilinadi. Shuningdek, NLP vazifalarini yechishda statistik va neyron tarmoqli metodlar qo‘l kelmoqda. Ko‘pgina NLP ilovalari chuqur neyron tarmoq usullaridan foydalanib, texnologik taraqqiyot, kompyuter quvvatining ortishi va katta hajmdagi til korpuslarining mavjudligi tufayli samarali ishlamoqda. Matnli ma'lumotlarining aksariyati strukturlanmagan, Internetda mavjud yoki turli manbalarda joylashgan. Matnli ma'lumotlar to‘g‘ri olingan, jamlangan, formatlangan va tahlil qilingan bo‘lsa, ahamiyatli va foydali bo‘la oladi. Matn tahlilini to‘g‘ri amalga oshirish kompaniya va tashkilotlarga turli yo‘llar bilan foyda keltirishi mumkin. Strukturalanmagan matnni tahlil qilish usullari matn tasnifi, hissiyotlarni tahlil qilish, NER obyektlarni aniqlash va mavzuni modellashtirish kabi vazifalarini qamrab oladi. NLPning ushbu vazifalari turli kontekstlarda qo‘llaniladi. Ularni bajarish uchun, birinchi navbatda, mashina inson tilini tushunishi, qayta ishlashi uchun nutq va matnlarni raqamli shaklga o‘tkazish zarur. Tabiiy tilni talqin qiluvchi, tushunuvchi aqlli tizimlarni ishlab chiqishda strukturlanmagan matnli ma'lumotlar bilan ishlash, ularni sun'iy intellekt metodlari vositasida qayta ishlash maqsadida raqamli shaklga o‘tkazish lozim. So‘zlarni joylashtirish – bu tabiiy tildagi leksik birliklarning umumiy semantikasi va lingvistik shablonlarini qamrab oluvchi so‘zlarning muayyan (fiksirlangan) uzunlikdagi vektor ko‘rinishlari. NLP tadqiqotchilari bunday tasvirlarni olishning turli usullarini taklif qilishgan. Jumladan, Word2ec 2013-yilda Google kompyaniyasi tadqiqotchilari tomonidan ishlab chiqilgan matnni qayta ishlashga va raqamlashtirishga mo‘ljallangan metod bo‘lib, uning asosiy maqsadi so‘zlarni vektorlar orqali ifodalashdan iborat. Word2vec metodi vostasida matndagi so‘zlarning semantikasi ma'no jihatdan kodlanadi. Ushbu maqolada Python tilidagi NumPy paketidan foydalangan holda word2vec metodi orqali o‘zbek tili matnlaridagi so‘zlarni raqamlashtirishni amalda qo‘llash masalasi tahlil qilinadi.

Kirish

Tabiiy tilni qayta ishlash (NLP) kompyuter va inson (tabiiy til) o‘rtasidagi o‘zaro muloqotni amalga oshiruvchi sun'iy intellektning alohida bo‘limi bo‘lib, NLP metodlari orqali (lug‘at yoki korpusdagi) so‘z va so‘z birikmalariga ishlov berishni soddalashtirish uchun raqamli vektorlar shaklida ifodalash talab etiladi. Tabiiy tilni modellashtirishning bunday vazifasi **so‘zlarni joylashtirish (word embedding)** deb ataladi [1].

Word Embedding – so‘zlarni sonli shaklda ifodalash usuli, u mashinali o‘rganish algoritmlariga o‘xshash ma'noga ega so‘zlarni tushunishga imkon beradi. Ushbu metod neyron tarmoqlar, ehtimollik modellari yoki so‘zlarning birgalikdagi matritsasida o‘lchamlarni kamaytirishdan orqali so‘zning sonli vektorlarini aniqlash vositasida **tabiiy tilni modellashtirishga** imkon beradi. So‘zlarni joylashtirish, shuningdek, *taqsimlangan semantik model (distributed semantic model)* yoki *vektor fazo modeli (vector space model)* deb ham ataladi [2]. Mazkur ta'rifda,

o'xshash so'zlarni turkumlash *(means categorizing similar words)* degan ma'no anglashiladi. Masalan, *olma, anor, banan* kabi mevalarni yaqin qo'yish kerak. *Kitob* esa bu so'zlardan ancha uzoqda bo'ladi. Kengroq ma'noda, so'zlarni joylashtirish metodi orqali *mevalar* to'plamidagi so'zlar, *kitob* so'zining vektor tasviridan uzoqroqda joylashgan vektorlarni yaratadi. So'zlarni joylashtirish metodi *hujjatlarni klasterlashda, matnlarni tasniflashda* va *tabiiy tilni qayta ishlash vazifalarida* yordam beradi [3].

Ko'pincha til korpusiga mos CBOW modelini shakllantirish uchun **so'zlarni joylashtirish vektorlari (word embedding vectors)**ning hajmi belgilanadi. Bunda ularning umumiy soni asosan lug'at hajmiga teng bo'ladi. Ushbu zich vektor maydonining o'lchovi an'anaviy Bag of Words (BOW) modellari yordamida qurilgan yuqori o'lchamli siyrak vektor maydonidan ancha kam. Hozirgi vaqtda so'zlarni joylashtirishning **CBOW** va **Skip-gram** kabi ikki xil arxitekturasi mavjud. Word2Vec modeli esa so'zlarni joylashtirish tasvirini yaratishda foydalaniladi.

Mikolov tomonidan yozilgan *"Distributed Representations of Words and Phrases and their Compositionality"* va *"Efficient Estimation of Word Representations in Vector Space"* sarlavhali maqolalarda Word2Vec usuli haqida boshlang'ich tushunchalar tavsiflangan [4,5].

So'zlarni joylashtirish (word embeddings) – bu so'zning vektor ko'rinishi bo'lib, har bir so'zning boshqa so'zlar bilan semantik va sintaktik aloqasini hisobga olgan holda, belgilangan vektor o'lchami bilan ifodalanadi. Word2vec arxitekturasi bitta yashirin qatlamli neyron tarmoqdir. Yashirin qatlamning og'irligi so'zning yo'qotish funktsiyasi orqali aniqlanadi. **Word2vec** usuli orqali korpus tasviri 2 xil usulda amalga oshiriladi:

– **CBOW** – atrofdagi so'z konteksti asosida oraliq so'zni taxmin qilishga asoslangan. CBOW metodida kontekst/atrofdagi so'zlarni hisobga olgan holda qaysi so'z ko'proq mos kelishi haqida bo'sh joylarni to'ldirishga harakat qilinadi. Ushbu usul kichikroq ma'lumotlar to'plamlari bilan samarali natijani beradi [6].

– **Skip-Gram** – maqsadli so'zdan (CBOWga teskari) atrofdagi kontekst so'zlarini taxmin qilishga harakat qiladi. Kattaroq hajmdagi ma'lumotlar to'plamida yaxshiroq natija beradi. Biroq o'quv ma`lumotlar top`lamini qayta ishlashga ko'p vaqt sarflanadi [6,7,8,9].

Matn tahlili (qayta ishlash) vazifalarini bajarish uchun chuqur o'rganish usullari orqali so'zlarni joylashtirish (word embedding) samaradorligi o'rganiladi, muhim jihatlari umumlashtiriladi. Matnlarni qayta ishlash maqsadida ishlab chiqilgan mashinali o'rganish algoritmi *statistik, matematik* va *optimallashtirish* tamoyillariga asoslanadi [10,11,12]. Biroq ushbu tamoyillar orqali strukturlanmagan, boshlang'ich ishlov berilmagan matnni qayta ishlab bo'lmaydi. Til korpusidagi matnlarga mos sonli vektorlarni ishlab chiqishdan avval ularni boshlang'ich qayta ishlash va normallashtirish kerak.

Ma`lumotlarni tayyorlash

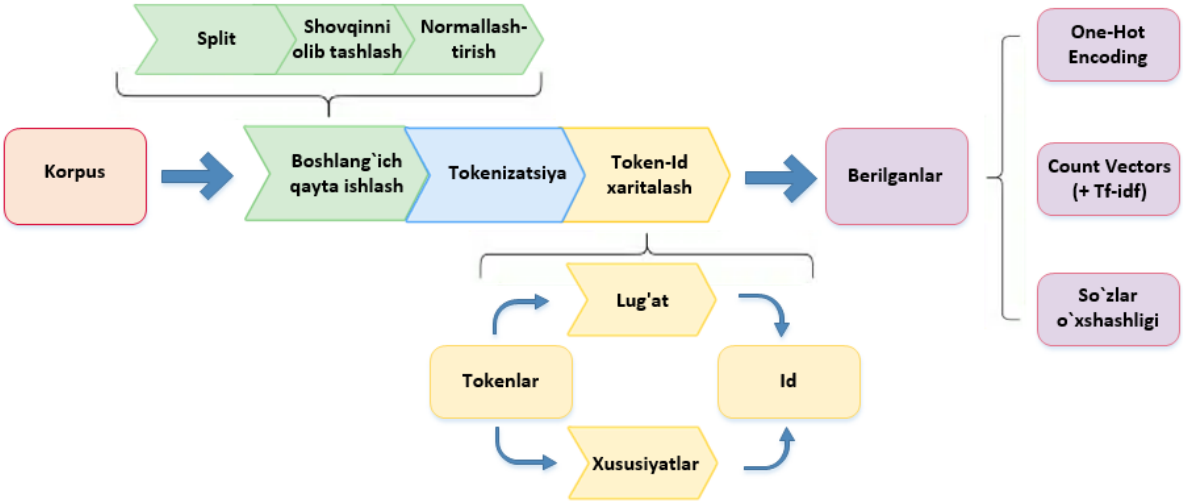
So'zlarning joylashuvini aniqlash maqsadida, bizga ayrim ma'lumotlar kerak bo'ladi. Bunda ijtimoiy tarmoq (online nashr)larda mavjud matndan foydalanish mumkin.

1. text = "'Kun bulutli bo'lsa, yomg'ir yog'ishi mumkin.
2. Ertaga havo bulutli bo'ladi.
3. Toshkent hayvonot bog'iga kichkina tog' echkisi olib kelindi.
4. Uch dona tuxumni idishda qaynatib oldi.
5. Sumalak, ko'k somsa, palov kabi bayram taomlari ulashildi.
6. Parranda va quyon go'shtidan taom tayyorlandi.
7. Darvoza oldida somsa sotilar ekan.
8. Tuyaqush boqish ham barakali biznesga aylandi."'

Tokenizatsiya

Matnlarni Word2vec modeli vositasida raqamlashtirish uchun avvalo ularga boshlang`ich ishlov berish kerak. Bu bosqich NLPda pipeline jarayoni deb yuritiladi[13]. Ushbu bosqich ko'plab NLP masalalarida qo'llaniladigan yondashuv bo'lib, matnni tokenizatsiyalash, ya'ni matnni kichikroq birliklarga ajratish (misol uchun so'zlarga) va tinish belgilarini olib tashlash kabi amallar bajariladi. B.Elov, Sh.Khamroeva va

Z.Xusainova tomonidan yozilgan *"The pipeline processing of NLP"* nomli maqolada o'zbek tili matnlarini boshlang`ich qayta ishlash haqida to'liq ma`lumot berilgan. Ko'plab NLP vazifalarida qayta ishlanmagan matn boshlang`ich qayta ishlanadi va mashinali o'rganish modeli uchun mos formatda ko'rinishlariga keltiriladi. Bunda ma'lumotlar *tokenizatsiya, nomuhim so'zlarini olib tashlash, tinish belgilarini olib tashlash, stemming, lemmatizatsiya* va boshqa bir qator boshlang`ich ishlov berish NLP vazifalari orqali qayta ishlanadi (1-rasm). Ushbu jarayonda ma'lumotlardagi mavjud "shovqin"lar tozalaanadi. Tozalangan ma'lumotlar NLP ilovasiga va mashinali o'rganish modelining kiritish talablariga muvofiq turli shakl (shablon)larda taqdim etiladi.



1-rasm. *Til korpusi matnlarini boshlang`ich qayta ishlash bosqichlari*

Yuqoridagi 1-rasmda korpus matnlarini ML modeli uchun turli kiritish formatlariga aylantirish jarayoni keltirilgan. Chapdan boshlab, korpus tokenlarni, matn qurilish bloklari to'plamini, ya'ni so'zlar, belgilar va boshqalarni olishdan oldin bir necha bosqichlardan o'tadi. ML modellari faqat sonli qiymatlarni qayta ishlashga asoslanganligi sababli, gapdagi tokenlar mos sonli qiymatlar bilan almashtiriladi. Keyingi qadamda, ular o'ng tomonda ko'rsatilgan turli xil kiritishli formatlarga aylantiriladi. Ushbu formatlarning har biri o'zining ijobiy va salbiy tomonlariga ega bo'lib, berilgan NLP vazifasining xususiyatlariga ko'ra strategik ravishda tanlanishi kerak. Quyida o'zbek tili matnlarini tokenlash funksiyasi keltirilgan:

```
import re
def tokenize(text):
   pattern = re.compile(r'[A-Za-z‘’]+[\w^\']*|[\w^\']*[A-Za-z]+[\w^\']*')
   return pattern.findall(text.lower())
```

Yuqorida keltirilgan namunaviy matndan foydalangan holda tokenlarni shakllantiramiz. tokenize(text) metodi orqali berilgan matndagi barcha tokenlar ro'yxati quyidagicha hosil qilinadi:

```
tokens = tokenize(text)
```

['kun', 'bulutli', 'bo'lsa', 'yomg'ir', 'yog'ishi', 'mumkin', 'ertaga', 'havo', 'bulutli', 'bo'ladi', 'toshkent', 'hayvonot', 'bog'iga', 'kichkina', 'tog'', 'echkisi', 'olib', 'kelindi', 'uch', 'dona', 'tuxumni', 'idishda', 'qaynatib', 'oldi', 'sumalak',...'boqish', 'ham', 'barakali', 'biznesga', 'aylandi']

Word2vec raqamli vektorlarini shakllantirishning yana bir muhim funksiyalardan biri – *tokenlar va indekslar o'rtasida map* (bog`lanish) yaratisdan iborat. Qaysidir ma'noda, ushbu funksiya orqali so'zlarni indekslarga va indekslarni so'zlarga osongina aylantirish imkonini beruvchi nazorat jadvalini shakllantiriladi. Bu, ayniqsa, keyinroq **one-hot kodlash** [6,14] kabi masalalarni hal qilishda foydali vosita bo'lib xizmat qiladi.

```
def mapping(tokens):
   word_to_id = {}
   id_to_word = {}
```

```
for i, token in enumerate(set(tokens)):
    word_to_id[token] = i
    id_to_word[i] = token
return word_to_id, id_to_word
```

Yuqorida keltirilgan mapping(tokens) metodi orqali **word-to-index** va **index-to-word** kabi bog`lanishlarning muvaffaqiyatli yaratilganini ko‘ramiz.

```
word_to_id, id_to_word = mapping(tokens)
word_to_id
```

{'taom': 0, 'olib': 1, 'idishda': 2, 'ertaga': 3, 'tuxumni': 4, 'kabi': 5, 'bo‘ladi': 6, 'bayram': 7, 'kichkina': 8, 'bulutli': 9, 'uch': 10, 'parranda': 11, 'go‘shtidan': 12, 'va': 13, 'darvoza': 14, 'sumalak': 15, 'taomlari': 16, 'ekan': 17, 'ham': 18, 'ulashildi': 19, 'dona': 20, 'bog‘iga': 21, 'mumkin': 22,
'havo‘: 23, 'hayvonot': 24,... 'kelindi': 42, 'boqish': 43, 'echkisi': 44, 'sotilar': 45, 'biznesga': 46}

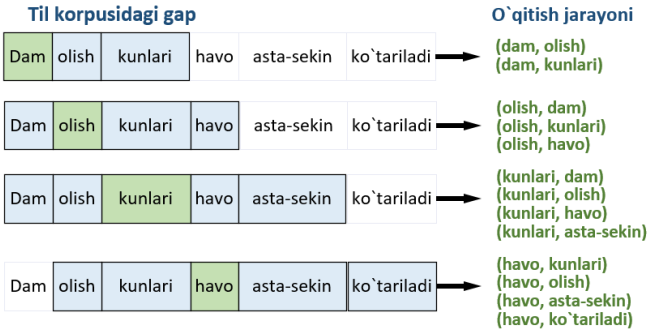
Yuqoridagi ro‘yxatdan qidiruv jadvalining o‘z navbatida *so‘zlar va identifikatorlar munosabatidan* iborat lug‘at obyekti sifatida tasvirlanini qayd etish mumkin. Ushbu qidiruv jadvalidagi har bir yozuv biz avval aniqlagan tokenize(text) funksiyasi yordamida yaratilgan tokenlardan iborat.

O‘quv ma'lumotlarini yaratish

Yuqorida bajarilgan amallar natijasida bizda *tokenizatsiyalangan matn* va *qidiruv jadvali* mavjud bo‘lib, keying bosqichda **training ma`lumolarini ishlab chiqish** lozim. Ushbu bosqich natijasida ma`lumotlat matritsa (jadval)lar ko‘rinishiga keltiriladi. Tokenlar hali ham satrlar ko‘rinishida bo‘lgani sababli, ularni **one-hot vektorizatsiya** (one-hot vectorization) yordamida raqamli ko‘rinishga olib kelish kerak [6,14,15]. Shuningdek, Word2vec nazorat ostida (o‘qituvchi bilan) o‘rgatish usuli hisoblanganligi sababli, **kiruvchi (input)** va **maqsadli** (**target**) o‘zgaruvchilar to‘plamini yaratish lozim.

Bu esa o‘z navbatida kiruvchi va maqsadli qiymatlarning qanday ko‘rinishga ega bo‘ladi degan savol kelib chiqishiga zamin yaratadi. Biz taxmin qilmoqchi bo‘lgan qiymatni va ehtimollarni hisoblash uchun modelga qanday ma'lumotlarni kiritish lozim? Ushbu savollarga javob berish uchun Word2vec metodi bilan qanday bog‘liqligi tushunish uchun so‘zlar joylashuvini tushunish kerak. Word2vec bu boshqa har qanday mashinali o‘qitish (ML, machine learning) masalalari kabi *kirish* va *chiqish* qiymatlarini puxta o‘qitish (training) natijasidir.

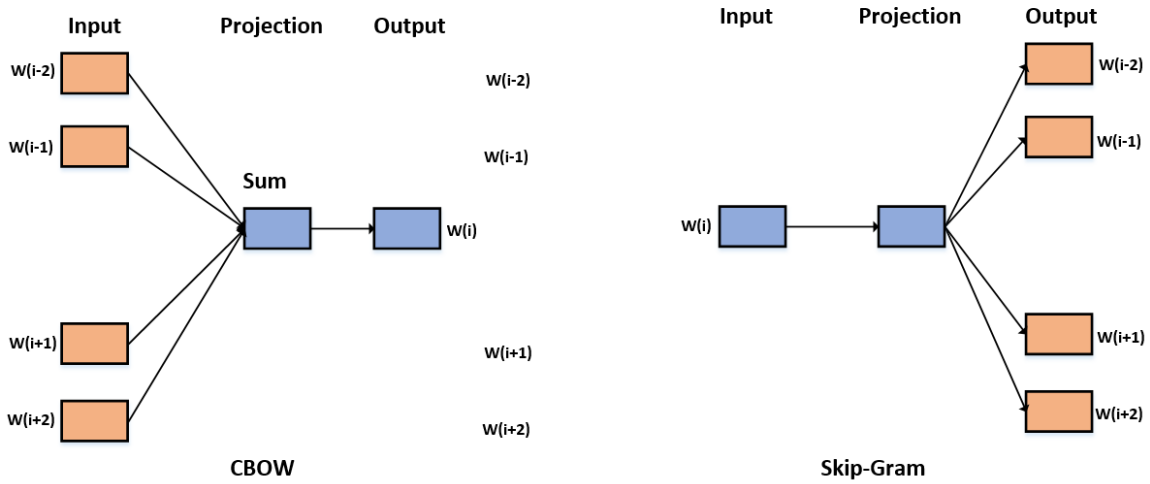
Bugungi kunda ushbu algoritmning ikkita turi mavjud: **CBOW** va **Skip-Gram**. *Skip-Gram usuli*da til korpusidagi har bir gapning so‘zlari qayta ishlanadi va qo‘shnilarini (ya’ni, uning kontekstini) bashorat qilish uchun joriy so‘zidan foydalanishga harakat qiladi. *CBOW (Continuous Bag Of Words) usulida* ushbu kontekstlardan foydalangan holda joriy so‘zi bashorat qilinadi. Har bir kontekstdagi so‘zlar sonini cheklash uchun **“oyna o‘lchami” (window size)** deb nomlangan parametr ishlatiladi.



3-rasm. Kontekstni tahlil qilish jarayoni

So‘zlarni ifodalash uchun foydalanadigan sonli vektorlar **neyron tarmoq asosida so‘zlarni joylashtirish (neural word embeddings)** deb ataladi. Word2vec usuli asosida til korpusidagi so‘zlar **“vektorlashtiradi”** va tabiiy tilni kompyuterda tushinish imkoniyati taqdim etiladi. So‘zga mos sonli vektorlar ustida matematik amallarni bajarish orqali ularni semantik yaqinligini aniqlash mumkin.

Neyron tarmoq asosida so‘zlar sonli ko‘rinishga ega bo‘ladi. Word2vec usuli avtokoderga o‘xshab, korpusdagi har bir so‘zni vektorda kodlaydi va ularga qo‘shni bo‘lgan boshqa so‘zlar asosida o‘rganadi. O‘rganish jarayoni ikki usuldan birida amalga oshiriladi. Yoki maqsadli so‘zni bashorat qilish uchun kontekstdan foydalanish (CBOW), yoki skip-gram deb ataladigan maqsadli kontekstni bashorat qilish uchun so‘zdan foydalanish. CBOW usuli katta hajmli ma’lumotlar to‘plamlarida aniqroq natijalar berganligi sababli, ushbu maqolada o‘zbek tili matnlarini qayta ishlash uchun Skip-Gramm usulining qo‘llanilishi ko‘rib chiqiladi.



4-rasm. CBOW va Skip-Gram usullari

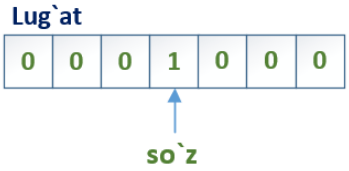
Agar so‘zga tayinlangan xususiyat vektoridan ushbu so‘z kontekstini aniq bashorat qilish imkoni mavjud bo‘lmasa, vektor komponentlari qayta ko‘rib chiqilishi yoki tahrirlanishi lozim. Korpusdagi har bir so‘zning konteksti uchun ML o‘qituvchisi xususiyat vektorini sozlash uchun xato signallarini yuboradi. Kontekstiga ko‘ra **o‘xshash** deb baholangan so‘zlarning vektordagi qiymatlari sozlash orqali bir-biriga yaqinlashadi. Skip-Gram modeli CBOWdan farqli o‘laroq, markaziy so‘zni yuqoridagi 4-rasmda ko‘rsatilgandek kirish sifatida ko‘rib chiqadi va kontekstdagi so‘zlarni taxmin qiladi.

One-hot encoding modeli

Yuqorida keltirilgan ma’lumotlarga ko‘ra neyron tarmog‘iga korpusdagi ba’zi juft so‘zlarni uzatish lozimligini anglash mumkin. Neyrom tarmoq ushbu juftliklarni qayta ishlashi uchun so‘zlarni matematik tarzda ifodalash kerak. Buning bir usuli – matndagi barcha so‘zlarning lug‘atini yaratish va keyingi qadamda lug‘at o‘lchovlariga mos tarzda har bir so‘zni vektor sifatida kodlashdan iborat. Vektorning har bir o‘lchovini lug‘atdagi so‘z sifatida tushinish mumkin. Shunday qilib, yuqorida keltirilgan usul orqali hosil qilingan vektorda nollar va lug‘atdagi mos keladigan so‘zni ifodalovchi 1 bo‘ladi. Ushbu kodlash usuli *one-hot encoding* deb ataladi.

One-hot encoding usuli orqali “*Dam olish kunlari havo harorati asta-sekin ko‘tariladi”* gapiga mos ravishda “*dam*”, “*olish*”, “*kunlari*”, “*havo*”, “*harorati*”, “*asta-sekin*”, “*ko‘tariladi*” so‘zlaridan tuzilgan lug‘at shakllantiriladi. “***havo*”** so‘zi quyidagi vektor bilan ifodalanadi: **[0,0,0,1,0,0,0]**.

One-hot encoding modeli matn korpusidagi bir so‘z uchun “**issiq**” **vektor (hot-vector)ni** shakllantiradi. Issiq vektor – bu so‘zning vektor ko‘rinishi bo‘lib, bunda vektor uzuligi lug‘at hajmi (jami unikal so‘zlar)ga bog‘liq. Raqamli vektorda so‘zni ifodalovchi o‘lchovdan tashqari barcha o‘lchovlarga 0 qiymati o‘rnatiladi:



5-rasm. “hot-vector” lug‘ati

Quyida keltirilgan dasturiy kodda yuqorida tavsiflangan Word2vec algoritmi yordamida malumotlarni o‘qitish jarayoni keltirilgan. Ushbu kod orqali tokenizatsiyalangan ma’lumotlardan so‘z juftliklari hosil qilinadi. Ushbu jarayonda birinchi va oxirgi tokenlar uchun kitruvchi tokenining chap yoki o‘ng tomonidagi so‘zlar olinmasligi mumkin. Bunday hollarda, biz bu so‘z juftliklarini hisobga olmaymiz va faqat *IndexErrors* xatolikni keltirib chiqarmaydigan holatlarni ko‘rib chiqamiz. Shu sababli, **X** va **Y** matritsalarni yuqorida ko‘rsatilganidek, kortej ko‘rinishida emas balki, alohida tarzda yaratamiz.

```
import numpy as np
np.random.seed(42)
```

```
def generate_training_data(tokens, word_to_id, window):
    X = []
    y = []
    n_tokens = len(tokens)
    for i in range(n_tokens):
        idx = concat(
            range(max(0, i - window), i),
            range(i, min(n_tokens, i + window + 1))
        )
        for j in idx:
            if i == j:
                continue
            X.append(one_hot_encode(word_to_id[tokens[i]], len(word_to_id)))
            y.append(one_hot_encode(word_to_id[tokens[j]], len(word_to_id)))
    return np.asarray(X), np.asarray(y)
```

Yuqoridagi 2 ta range() obyektlarini birlashtirish maqsadida ishlatilgan concat(*iterables) funksiyasi quyida keltirilgan:

```
def concat(*iterables):
    for iterable in iterables:
        yield from iterable
```

Quyidagi One-hot vektorizatsiya tokenlarida ishlatilgan kod keltirilgan bo'lib, bu jarayon har bir tokenni vektorizatsiya ko'rinishiga keltirish uchun muhimdir. Keyinchalik esa ushbu vektorlarni birlashtirish orqlai **X** va **Y** matritsalar hosil qilinadi.

```
def one_hot_encode(id, vocab_size):
    res = [0] * vocab_size
    res[id] = 1
    return res
```

Quyida keltirilgan kod orqali **window_size=2** o'lchamidagi oynadan iborat o'quv ma`lumotlarni hosil qilamiz.

```
X, y = generate_training_data(tokens, word_to_id, 2)
```

O'quv ma'lumotlar o'lchamini tekshiramiz:

```
X.shape
Y.shape
```

```
(190, 47)
(190, 47)
```

X va **Y** matrisalar **190** ta satr va **47** ta ustundan iborat. Bu yerda **190** o'quv ma`lumotlari soni. **window_size** parameter qiymatini kattaroq o'rnatish orqali o'quv ma`lumotlar sonini oshirish mumkin. **47** – bu bizning korpusimizning o'lchami yoki dastlabki matndagi unikal (noyob) tokenlar soni. Keyingi qadamda Word2vec metodining **so'zlarni joylashtirish** *tarmog`ini ishlab chiqish va joriy qilish* bosqichini amalga oshirish mumkin.

So'zlarni joylashtirish modeli

Ushbu bosqichda, berilgan tokenga yaqin bo'lgan kontekstli so'zlarni bashorat qiladigan neyron tarmoqni o'rgatish amalga oshiriladi. Umuman olganda, neyron tarmog`ining *chiqish qatlamida yakuniy vektor (embedding vector)* emas, balki *softmax qatlami* orqali hosil qilingan ba'zi **ehtimollik vektori (probability vector)** shakllantiriladi.

O'rtacha vazn matritsasining satrlari biz izlayotgan joylashish qiymatlari hisoblanadi! Agar modelni tashkil etuvchi og'irlik matritsalarining o'lchamlarini hisobga oladigan bo'lsak, bu jarayon yanada tushunarliroq bo'ladi. Quyida keltirilgan sodda bir misolda ushbu jarayonni tushuntiramiz. Matnli korpusda atigi 5 ta so'z bor deb faraz qilaylik va biz bu so'zlarni 3D vektor sifatida ifodalamoqchimiz. Xususan, quyida modelning birinchi **og'irlik qatlami (weight layer)** keltirilgan:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix} * \begin{pmatrix} 2 & 1 & 7 \\ 1 & 8 & 6 \\ 9 & 0 & 4 \\ 7 & 5 & 5 \\ 5 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 8 & 6 \\ 2 & 1 & 7 \\ 7 & 5 & 5 \\ \dots & \dots & \dots \end{pmatrix} \quad (1)$$

Amalga oshirilgan sinov natijalari shuni ko'rsatadiki, neyron tarmog`idagi kirish ma`lumotlari one-hot vektorlaridan iborat bo'lganligi sababli, vazn matritsasi samarali ravishda one-hot kodlangan vektorlarni boshqa o'lchamdagi zich vektorlarga, aniqrog'i, vazn matritsasidagi satr fazosiga o'tkazadigan qidiruv jadvali sifatida ishlaydi. Yuqoridagi misolda vazn matritsasi $R^5 \rightarrow R^3$ transformatsiyasi bo'lib hisoblanadi. Bu joylashtirish orqali so'zlarni zich vektor sifatida ifodalash amalga oshiriladi.

Ushbu jarayon esa joylashtirish (embedding)ning nima ekanligini ko‘rsatadi: biz yuqorida yaratilgan o‘quv ma`lumotlaridan foydalangan holda ushbu modelni o‘rgatishni boshlaganimizda, ushbu vazn matritsasining satrlari fazosi o‘quv ma`lumotlaridan foydalangan holda mazmunli semantik ma'lumotlar kabi natijani kutib qolamiz. O‘qitish jarayoning keyingi qadamida joylashuvlarni kirish qatlamida qabul qilib, chiqish qatlamini generatsiya qilishda undan foydalaniladi.

$$\begin{pmatrix} 1 & 8 & 6 \\ 2 & 1 & 7 \\ 7 & 5 & 5 \\ \dots & \dots & \dots \end{pmatrix} * \begin{pmatrix} 8 & 4 & 5 & 1 & 8 \\ 1 & 6 & 2 & 5 & 7 \\ 0 & 2 & 0 & 3 & 4 \end{pmatrix} = \begin{pmatrix} 16 & 64 & 21 & 59 & 84 \\ 17 & 28 & 12 & 26 & 51 \\ 61 & 68 & 45 & 47 & 111 \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix} \quad (2)$$

Oxirgi qadamda bizga faqat softmax qatlami kerak bo‘ladi. Qayta ishlangan ma`lumotlar chiqish qatlamiga uzatilganidan so‘ng, u elementlari yig‘indisi birga teng bo‘lgan ehtimollik vektorlariga aylanadi. Ushbu yakuniy natijani **kontekst ehtimollik**, ya'ni kirish so‘zi yonidagi oynada qanday so‘zlar bo‘lishi mumkinligini taxmin qilish deb hisoblash mumkin.

$$softmax\left[\underbrace{\begin{pmatrix} 16 & 64 & 21 & 59 & 84 \\ 17 & 28 & 12 & 26 & 51 \\ 61 & 68 & 45 & 47 & 111 \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix}}_{chiqish}\right] = \underbrace{\begin{pmatrix} 0.1 & 0.2 & 0.1 & 0.2 & 0.4 \\ 0.2 & 0.2 & 0.1 & 0.2 & 0.3 \\ 0.2 & 0.2 & 0.1 & 0.1 & 0.4 \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix}}_{ehtimollik} \quad (3)$$

Ma`lumotlarni o‘qitish jarayonida, xususan, *xatolarni aniqlash* va *backpropogation (orqaga harakatlanish)* jarayonlarida ehtimollik vektorlarining bashoratini uning one-hot kodlangan maqsadli qiymatlari bilan taqqoslaymiz. Biz softmax bilan ishlatadigan xatolik funksiyasi *entropiya* bo‘lib, u quyidagi formula orqali hisoblanadi:

$$H(p,q) = -\sum_{x \in X} p(x) log\ (q(x)) \ (4)$$

H(p,q) qiymatni *maqsadli vektor* va *prognoz jurnalining* skalyar ko‘paytmasi deb qabul qilish mumkin. Ushbu muqobil formulada o‘zaro entropiya formulasini quyidagicha qayta yozish mumkin:

$$H(p,q) = -p * log(q) \quad (5)$$

Bu holda p – one-hot kodlangan vektor bo‘lganligi sababli, barcha qiymatlari nolga teng bo‘lgan p yakuniy natijaga ta'sir qilmaydi. Darhaqiqat, biz manfiy logarifm bashorati natijasiga ega bo‘lamiz. E'tibor bering, bashorat qiymati 1 ga qanchalik yaqin bo‘lsa, entropiya shunchalik past bo‘ladi va aksincha. Bu esa bizga kerakli holatlarga mos keladi, chunki biz bashorat qilingan ehtimollik imkon qadar 1 ga yaqin bo‘lishini xohlaymiz.

Shunday qilib, butun jarayonni umumlashtirib olaylik. Birinchidan, joylashuvlar - vazn matritsasi qatorlari bo‘lib, $\mathbf{W_1}$ bilan belgilanadi. O‘quv va backpropogation jarayonlari orqali $\mathbf{W_1}$ ning vaznini ikkinchi qatlamdagi vazn matritsasi bilan entropiyadan foydalangan holda moslashtiramiz va $\mathbf{W_2}$ kabi belgilaymiz. Umuman olganda, bizning modelimiz quyidagi ko‘rinishga ega bo‘ladi:

$$A_1 = XW_1$$
$$A_2 = XW_2 \quad (6)$$
$$Z = softmax(A_2)$$

Bu yerda, **Z** – bashoratning ehtimollik vektorlarini o‘z ichiga olgan matritsa. Yuqoridagi belgilashlar va formulalar asosida modelimizni qurish va o‘qitish bosqichini amalga oshiramiz.

Dasturiy kodni amalga oshirish

Ushbu modelni kodlash jarayonini boshlaymiz. Vazn matritsalari va kalitlaridan foydalangan holda, og‘irlik matritsalariga murojaat qilamiz. Avvaldan belgilangan nomenklaturaga muvofiq, biz ushbu o‘lchovlarni nazarda tutgan holda **"w1"** va **"w2"** parametrlardan foydalanamiz.

```
def init_network(vocab_size, n_embedding):
  model = {
    "w1": np.random.randn(vocab_size, n_embedding),
```

```
    "w2": np.random.randn(n_embedding, vocab_size)
  }
  return model
```

Keling, o‘n o‘lchovli joylashuvlarni yaratish uchun Word2vec modelimizda ba`zi o‘zgartirishlarni amalga oshiramiz. Ya`ni, endilikda har bir token – o‘n o‘lchovli fazoda vektorlar sifatida ifodalanadi. E'tibor bering, haqiqiy modellar odatda ancha kattaroq o‘lchamlardan foydalanadi. Ko‘p hollarda ushbu qiymat **300** atrofidagi sondan iborat.

```
model = init_network(len(word_to_id), 10)
```

Oldinga harakatlanish (Forward Propagation)

Oldinga harakatlanish qadimida (6) formulada keltirilgan uchta matritsani ko‘paytirish tenglamalarini NumPy paketidagi dekodlash amalga oshiriladi.

```
def forward(model, X, return_cache=True):
  cache = {}
  cache["a1"] = X @ model["w1"]
  cache["a2"] = cache["a1"] @ model["w2"]
  cache["z"] = softmax(cache["a2"])
   if not return_cache:
     return cache["z"]
  return cache
```

Orqaga harakatlanish (Back Propogation) qadamini bajarish uchun oraliq qatlamda ishlatilgan barcha o‘zgaruvchilar kerak bo‘ladi. Shu sababli, biz ularni *cache* deb nomlangan lug‘atda saqlaymiz. Ammo, agar bizga faqat *yakuniy bashorat vektorlarini (final prediction vectors*) kerak bo‘lsa, *return_cache* parametri qiymatini *False* bilan belgilaymiz. Ushbu qadamni bajarish uchun yuqorida qo‘llagan softmax(X) funksiyasini ham ishlab chiqish lozim. E'tibor bering, softmax(X) funktsiyasi kirish parametric sifatida vektor emas, matritsani qabul qiladi. Shuning uchun, oddiy takrorlanish amali yordamida jarayonlarni biroz o‘zgartirishimiz kerak bo‘ladi.

```
def softmax(X):
  res = []
  for x in X:
    exp = np.exp(x)
    res.append(exp / exp.sum())
  return res
```

Ushbu qadamni amalga oshirish orqali oldinga harakatlanish bosqichini yakunaymiz. Word2vec modelining keying bosqichariga o‘tishdan oldin, matritsalarning o‘lchamlarini tekshirib olish kerak. Berilgan matn Word2vec modelining *birinchi (first)* yoki *joylashtirish (embedding)* qatlamidan o‘tgandan so‘ng, matritsaning o‘lchami quyidagicha ko‘rinishga keladi:

```
(X @ model["w1"]).shape
```

```
(190, 10)
```

Bu kutilgan natija hisoblanadi. Chunki biz matndagi barcha 190 ta tokenni o‘n o‘lchovli vektorlarga aylantirishni nazarda tutib qo‘yganmiz. Keyinchalik, matn *ikkinchi (second) qatlamdan* o‘tganidan keyin o‘lchamni tekshiramiz.

```
(X @ model["w1"] @ model["w2"]).shape
```

```
(190, 47)
```

Bu safar bu **190x47** o‘lchanli matritsadan iborat bo‘ladi. Chunki natija **47** o‘lchovli bo‘lishi lozim bo‘lib, one-hot kodlashdan keyin asl o‘lchamlarga qaytamiz. Keyinchalik bu natija softmax qatlamiga uzatiladi va natijada ehtimollik vektorlari to‘plami hosil qilinadi.

Orqaga harakatlanish (Back Propogation)

Orqaga harakatlanishni bosqichini amalga oshirish oldinga harakatlanish bosqichidan ko‘ra biroz murakkabroq. Yakuniy tenglama quyidagi ko‘rinishga ega:

$$\frac{\partial L}{\partial A_2} = Z - y$$

Model xatoligi qiymati ma`lumligi sababli, matritsa hisoblashlarining asosiy tamoyillarini inobatga olgan holda butun neyron tarmog`i bo‘ylab qo‘llash mumkin. Gradientning o‘lchami asl matritsaning o‘lchamiga teng bo‘lishi kerakligini hisobga olib, backward(model, X, y, alpha) funksiyasini quyidagicha aniqlaymiz:

```
def backward(model, X, y, alpha):
```

```
    cache  = forward(model, X)
    da2 = cache["z"] - y
    dw2 = cache["a1"].T @ da2
    da1 = da2 @ model["w2"].T
    dw1 = X.T @ da1
    assert(dw2.shape == model["w2"].shape)
    assert(dw1.shape == model["w1"].shape)
    model["w1"] -= alpha * dw1
    model["w2"] -= alpha * dw2
    return cross_entropy(cache["z"], y)
```

Orqaga harakatlabish jarayoni davomida xatolik qiymati jurnalini saqlash uchun backward() funksiyasining yakuniy qaytish qiymatini bashorat va maqsadli teglar o‘rtasidagi *o‘zaro entropiya (cross entropy)* yo‘qotilishiga aylantirish lozim. O‘zaro entropiyani yo‘qotish funktsiyasini quyidagi tarzda osongina amalga oshirish mumkin:

```
def cross_entropy(z, y):
    return - np.sum(np.log(z) * y)
```

Keyingi qadamda modelni o‘rgatish va sinovdan o‘tkazish mumkin.
Modelni sinovdan o‘tkazish

Ushbu maqolada kam miqdordagi o‘quv ma'lumotlariga ega matn ustida Word2vec modeli qo‘llanilganligi sababli, 50 ta *davr (epoch)*dan iborat iteratsiyani amalga oshiramiz. O‘quv jarayoni davomida *history* ro‘yxatida cross entropiya error funksiyasining qiymatini keshlaymiz. So‘ngra hosil qilingan natijalarni to‘g‘riligini aniqlash uchun grafikli tasvirni hosil qilamiz.

```
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'svg'
plt.style.use("grayscale")
n_iter = 50
learning_rate = 0.05
history = [backward(model, X, y, learning_rate) for _ in
range(n_iter)]
plt.plot(range(len(history)), history, color="skyblue")
plt.show()
```

Shunday qilib, yuqqoridagi natijalar asosida biz o‘quv ma`lumotlarini *joylashuv qatlami (embedding layer)*dan muvaffaqiyatli o‘tdi deb ishonch bilan aytishimiz mumkin.

```
learning = one_hot_encode(word_to_id["bayram"],
len(word_to_id))
result = forward(model, [learning],
return_cache=False)[0]
for word in (id_to_word[id] for id in
np.argsort(result)[::-1]):
    print(word)
```

Kabi, ulashildi, taomlari, palov, oldida, bayram, echkisi, quyon, va, ekan, ko‘k, parranda, sumalak, tog‘, kichkina, darvoza, olib, go‘shtidan, uch, bog‘iga, idishda, boqish, tayyorlandi, somsa, qaynatib, ..., havo, yog‘ishi, mumkin, bulutli,

So‘zlarni joylashtirish

Word2vec modelini o‘zbek tili matnlariga qo‘llashdan asosiy maqsad neyron tarmoq yaratish emas, balki so‘zlarni joylashtirishdan iborat. Ushbu maqolada so‘zlarni joylashtirishning asosiy mohiyati shundaki, birinchi vazn matritsasining satrlari aslida matn ma'lumotlar to‘plamidagi har xil tokenlarga mos keladigan one-hot kodlangan vektorlarning zichlik ifodasidir. Shunday qilib, bizning namunaviy matnimizga mos so‘zlar joylashuvini quyidagicha aniqlash mumkin:

```
model["w1"]
```

```
array([[-1.77834736, -1.71118313,  0.8128522 ,
0.83122324,  0.09011965,
        1.12515163, -0.15084759, -0.74274437,
0.78959857,  0.83444731],
       [ 0.66722351,  0.77871013, -0.28287462, -
1.50285936, -3.09483586,
        -0.27280923, -0.3795136 ,  0.09916834, -
0.58916751, -1.10779202],
       [ 0.70233125,  0.64151466, -0.82870607,
0.22333784,  0.63794264,
         0.31508777,  0.15733498,  1.86007226, -
2.02376076, -0.38043539],
       [ 0.46470001,  0.52754105, -1.31741452, -
2.01887553,  2.05612723,
         0.1504839 , -0.38557481, -0.56585359, -
0.07549121, -0.30120162],
       [ 0.45391041,  1.32545805, -0.89751218, -
0.5103604 , -1.03097668,
         1.11587855,  0.36703787,  1.5195217 , -
0.94875896, -1.48198889],
       [-2.04919849,  0.77607815,  0.28381557,
0.1580143 ,  0.27162906,
        -1.2489531 , -0.2447962 , -1.00847149,
0.97642302,  0.54529323],
```

```
      [ 2.25408424,  0.62351749,  0.11997246, -
1.47497836,  0.8544083 ,
       0.51370441, -0.83694609,  0.47156148,
0.51322539,  0.04087653],
      [-1.70313088,  1.17473888,  0.53098298,
1.33220002, -1.95771352,
       -0.45917187, -0.42343097, -1.31234154, -
0.02935387, -0.75051765],
      [ 1.39552486,  1.95556482,  1.05768466, -
0.69967713, -0.79754217,
       -0.82766484, -0.03610394, -0.83889428, -
0.60178189, -0.54935944],
      [ 1.59748133,  0.09875392, -0.59454853, -
0.29005016,  1.07594892,
       -2.11645766, -0.05887795, -0.76144644, -
0.0629483 ,  0.9140858 ],
      [ 0.39100414,  0.26508985, -2.08161107, -
0.28030012, -0.24576466,
        0.63652933,  0.96877408,  1.79325181,
0.85385862,  0.91585647],
      [-1.60875135, -0.49427592,  2.17006069,
1.30696819, -1.25923668,
        0.03020403, -0.79494606, -0.97729965,
0.92557562, -0.33019666],
      [-0.31029848, -1.12562356,  1.4555185 , -
1.10295027, -0.39108011,
...
       -0.13143597, -0.53615308, -0.78175882,
0.84430443, -1.19278228],
      [-1.56132266, -1.24714023, -0.59257249, -
1.2078075 , -0.27954628,
       -0.3679814 ,  0.60577809, -1.42842891, -
1.61673601, -0.36429139],
      [ 0.6885209 , -0.88250196,  0.22220893,
1.25926952, -0.57925017,
        1.504267  , -0.02739082,  1.46510496,
0.23868249, -0.87058806]])
```

Yuqoridagi WE vektori qiymatlarini tushuniy biroz murakkablikni hosil qiladi. Xususan, biz funksiya orqali so'z kiritish imkoniyatiga ega bo'lishni va chiqishda ushbu so'z uchun so'zlar joylashuvi vektorini hosil qilishimiz lozim. Quyida ushbu vazifani amalga oshiradigan funksiya keltirilgan:

```
def get_embedding(model, word):
    try:
        idx = word_to_id[word]
    except KeyError:
        print("`word` not in corpus")
    one_hot = one_hot_encode(idx, len(word_to_id))
    return forward(model, one_hot)["a1"]
```

Misol sifatida, **"bayram"** so'ziga mos o'n o'lchovli vektor quyidagi ko'rinishga ega:

```
get_embedding(model, "bayram")
```

```
array([-1.70313088,  1.17473888,  0.53098298,
1.33220002, -1.95771352,
       -0.45917187, -0.42343097, -1.31234154, -
0.02935387, -0.75051765])
```

Ushbu vektor tasodifiy tanlab olingan raqamlar to'plami emas, balki yuqorida tavsiflangan Word2vec algoritmi yordamida yaratilgan kontekst ma'lumotlaridan foydalangan holda matnni o'qitish natijasidir. Bu vektorlar qaysi so'zlar boshqa so'zlar bilan uchrashishini ko'rsatadigan mazmunli semantik ma'lumotlarni ifodalaydi.

XULOSA

Matnli ma'lumotlarni qayta ishlashda ularni raqamli yoki sonli ko'rinishga keltirish talab etiladi. Mazkur vazifani amalga oshirish uchun so'zlar o'rtasidagi semantik aloqani o'zida saqlaydigan usulni tanlash, hujjatlarda so'zlar nafaqat semantika, balki kontekstni ham ifodalashi uchun sonli tasvirlarni hosil qilish lozim. Kompyuterning so'z va uning ma'nolarini tushunishi uchun o'rnatish/joylashtirsh (embeddings) deb atalgan usuldan foydalaniladi. So'zlarni joylashtirish tabiiy tilni qayta ishlashning alohida yo'nalishi bo'lib, so'zlarni sonli vektorga moslashda uning qurshoviga asoslaniladi. Ushbu qoidalar so'zlarni matematik vektor sifatida ifodalaydi. Mazkur qoidalar to'g'ri va aniq ishlab chiqilsa, o'xshash ma'noga ega bo'lgan so'zlar o'xshash raqamli qiymatlarga ega bo'ladi. Bu esa kompyuterga turli so'zlar orasidagi bog'lanish va o'xshashlikni ularning raqamli ko'rinishiga asoslangan holda tushunish imkonini beradi. So'zlarni joylashtirishni o'rganishning Word2Vec, GloVe va FastText kabi mashinali o'qitish (Machine Learning, ML)ning chuqur o'rganish (Deep Learning, DL) usullari navjud. Ushbu maqolada Word2vec metodining nisbatan oddiy tadbig'i keltirildi, asosiy tamoyillar aniq misollar yordamida ko'rsatildi. Word2vec metodining asosiy g'oyasi neyron tarmog'ini vazn matritsasi ko'rinishiga keltirishdan iborat. Shuning uchun joylashuv qatlam (embedding layerl)lari Python tilidagi TensorFlow yoki PyTorch kabi mashhur neyron tarmoq paketlari orqali tatbiq qilindi. Maqolada matematik hisob-kitoblar nisbatan kichik matn korpusi ustida bajarildi. Modelning aniqligi va samaradorligini

oshirish uchun katta hajmdagi matli korpuslardan foydalanish maqsadga muvofiq. Katta hajmdagi matnli ma'lumotlar ustida Word2vec va GloVe kabi metodlarni qo'llash sonli vektorlar yordamida matn semantikasini kodlash va tabiiy tildagi *matnni tahlil qilish, matn tasnifi, his-tuyg'ularni tahlil qilish, NER obyektni tanib olish, mavzuni modellashtirish va* NLPning boshqa vazifalarini yechishda foydalaniladi.

Foydalanilgan adabiyotlar

1. Sabharwal, N., & Agrawal, A. (2021). Introduction to Word Embeddings. In *Hands-on Question Answering Systems with BERT*. https://doi.org/10.1007/978-1-4842-6664-9_3
2. Tan, M., Zhou, W., Zheng, L., & Wang, S. (2012). A Scalable Distributed Syntactic, Semantic, and Lexical Language Model. *Computational Linguistics*, *38*(3). https://doi.org/10.1162/COLI_a_00107
3. Elsafoury, F., Wilson, S. R., & Ramzan, N. (2022). A Comparative Study on Word Embeddings in Social NLP Tasks. *SocialNLP 2022 - 10th International Workshop on Natural Language Processing for Social Media, Proceedings of the Workshop*. https://doi.org/10.18653/v1/2022.socialnlp-1.5
4. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2006). Distributed Representations of Words and Phrases and their Compositionality. *Neural Information Processing Systems*, *1*.
5. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*.
6. Elov, B., Hamroyeva, Sh., Alayev , R., Xusainova , Z., & Yodgorov , U. (2023). O'zbek tili korpusi matnlarini qayta ishlash usullari. *Digital transformation and artificial intelligence*, 1(3), 117–129. Retrieved from https://dtai.tsue.uz/index.php/dtai/article/view/v1i317
7. Elov B., Aloyev N., Xusainova Z., Yuldashev A. O'zbek tili korpusi matnlarini qayta ishlash Word2Vec, GloVe, ELMO, BERT usullari // Труды XI Международной конференции «Компьютерная обработка тюркских языков» «TURKLANG 2023». Бухара, 20-22 октября 2023 г.
8. Bamler, R., & Mandt, S. (2017). Dynamic Word Embeddings via Skip-Gram Filtering. *Proceedings of ICML 2017*.
9. Preethi Krishna, P., & Sharada, A. (2020). Word Embeddings - Skip Gram Model. In *ICICCT 2019 – System Reliability, Quality Control, Safety, Maintenance and Management*. https://doi.org/10.1007/978-981-13-8461-5_15
10. Bakarov A. (2018) Asurvey of word embeddings evaluation methods. arXiv preprint https://doi.org/10.48550/arXiv.1801.09536
11. Nayak, N., Angeli, G., & Manning, C. D. (2016). Evaluating word embeddings using a representative suite of practical tasks. *Proceedings of the Annual Meeting of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/w16-2504
12. Ghannay, S., Favre, B., Estève, Y., & Camelin, N. (2016). Word embeddings evaluation and combination. *Proceedings of the 10th International Conference on Language Resources and Evaluation, LREC 2016*.
13. B.ELov, Sh.Khamroeva, Z.Xusainova (2023). The pipeline processing of NLP. *E3S Web of Conferences 413, 03011, INTERAGROMASH 2023.* https://doi.org/10.1051/e3sconf/202341303011
14. Rodríguez, P., Bautista, M. A., Gonzàlez, J., & Escalera, S. (2018). Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, *75*. https://doi.org/10.1016/j.imavis.2018.04.004
15. Karthiga, R., Usha, G., Raju, N., & Narasimhan, K. (2021). Transfer Learning Based Breast cancer Classification using One-Hot Encoding Technique. *Proceedings - International Conference on Artificial Intelligence and Smart Systems, ICAIS 2021*. https://doi.org/10.1109/ICAIS50930.2021.9395930