ISSN: 3030-3346

MATNLI MA’LUMOTLARNI WORD2VEC, GLOVE VA FASTTEXT CHUQUR O‘RGANISH USULLARI VOSITASIDA QAYTA ISHLASH

Botir Elov

texnika fanlari falsafa doktori, dotsent. Alisher Navoiy nomidagi Toshkent davlat o‘zbek tili va adabiyoti universiteti.
E-pochta: elov@navoiy-uni.uz

K E Y W O R D S

Matnli ma’lumotlarni qayta ishlash, Word2Vec, GloVe, FastText, chuqur o‘rganish usullari, sonli vektorlar, so‘zlar o‘xshashligi.

A B S T R A C T

Kompyuter so‘zlarni inson kabi tushunmaydi, ular raqamlar bilan ishlashni afzal ko‘radi. Biroq buni amalga oshirish uchun so‘zlar o‘rtasidagi semantik aloqalarni o‘zida saqlanadigan usulni tanlashni istaymiz va hujjatlarda so‘zlar nafaqat semantikani, balki kontekstni ham eng yaxshi ifodalash uchun sonli tasvirlar hisoblanadi. Kompyuterga so‘zlar va ularning ma’nolarini tushunishga yordam berish uchun biz o‘rnatish/joylashtirsh (embeddings) deb ataladigan usuldan foydalanamiz. So‘zlarni joylashtirish – bu tabiiy tilni qayta ishlashning maxsus sohasi bo‘lib, so‘zlarni sonli vektorlarga mos qo‘yishda so‘zning qurshovidan foydalanadi. Ushbu o‘rnatishlar so‘zlarni matematik vektor sifatida ifodalaydi. Ushbu o‘rnatishlar to‘g‘ri va aniq ishlab chiqilganda, o‘xshash ma’noga ega bo‘lgan so‘zlar o‘xshash raqamli qiymatlarga ega bo‘ladi. Bu esa kompyuterlarga turli so‘zlar orasidagi bog‘lanish va o‘xshashliklarni ularning raqamli ko‘rinishlariga asoslangan holda tushunish imkonini beradi. Bugungi kunda 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. NLP vazifasini hal qilishda samarali natijalarga erishish uchun so‘zlarni joylashtirish va chuqur o‘rganish modellarini tanlash juda muhim. Hozirda tabiiy tildagi matnni tahlil qilish, matn tasnifi, his-tuyg‘ularni tahlil qilish, NER obyektni tanib olish, mavzuni modellashtirish va boshqa NLP vazifalarini hal qilishda ushbu ML usullaridan keng miqyosida foydalanilmoqda. Ushbu maqolada o‘zbek tili korpusi matnlarini ushbu usullar vositasida qayta ishlash usullari, ularning arxitekturalari, so‘zlarni joylashtirish va chuqur o‘rganish modellarining Python tilidagi tadbig‘i keltiriladi. Shuningdek, NLP bo‘yicha so‘nggi tadqiqot tendentsiyalarining umumiy ko‘rinishini va matn tahlili vazifalarida samarali natijalarga erishish uchun ushbu modellardan qanday foydalanish batafsil keltiriladi. Matnni tahlil qilish vazifalarini bajarish uchun turli usullarni qiyosiy tahlil qilish asosida so‘zlarni joylashtirish va chuqur o‘rganish yondashuvini tanlash uchun zarur ma’lumotlar taqdim etiladi. Ushbu maqola so‘zlarni sonli ifodalashning turli yondashuvlari va chuqur o‘rganish modellarining asoslari, afzallik va qiyinchiliklarini o‘rganish uchun tezkor ma’lumot bo‘lib xizmat qilishi mumkin. Maqolada keltirilgan usullar o‘zbek tili matnlarini tahlil qilishda qo‘llanilishi va kelajakdagi NLP sohasidagi ilmiy tadqiqot uchun zarur vosita bo‘lib xizmat qiladi.

Kirish

Tabiiy tilni qayta ishlash (NLP) – tilshunoslik, kompyuter ilmlari va sun’iy intellektning kompyuter va inson o‘zaro ta’siri bilan bog‘liq bo‘lgan bo‘limi bo‘lib, asosan, tabiiy til ma’lumotlarini qayta ishlash va baholash uchun metodlar, algortimlar va axborot tizimlarini loyihalash hamda ishlab chiqish masalalari bilan shug‘ullanadi. NLP – statistik, mashinali o‘qitish va chuqur o‘rganish modellarini kompyuter lingvistikasi qoidalariga asoslangan tarzda inson tilini modellashtirish bilan umumlashtiradi. Hozirda NLP usullari vositasida katta hajmdagi til korpuslari va millionlab veb-sahifalar bir soniya ichida tahlil qilinishi mumkin. Shuningdek, NLP vazfalarini hal qilishda statistik va neyron tarmoqli metodlardan foydalanilmoqda. Ko‘pgina NLP ilovalari chuqur neyron tarmoq usullaridan foydalanib, texnologik taraqqiyot, kompyuter quvvatining ortishi va katta hajmdagi til korpuslarining mavjudligi tufayli samaradorlgi va aniqligi yuqori natija bermoqda [1, 2, 3].

Matnli ma’lumotlarining aksariyati strukturlanmagan, Internet tizimi bo‘ylab

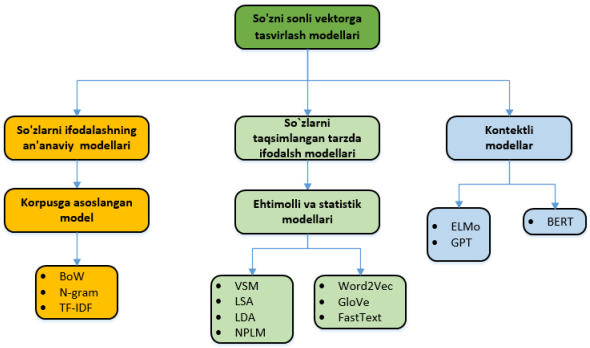
tarqalgan hamda turli manbalarda joylashgan. Matnli ma’lumotlar *to‘g‘ri olingan, jamlangan, formatlangan* va *tahlil qilingan* bo‘lsa, foydali bilimlarni berishi mumkin. Matn tahlilini to‘g‘ri amalga oshirish kompaniya va tashkilotlarga turli yo‘llar bilan foyda keltirishi mumkin. Matnni tahlil qilish vazifalarini bajarishning eng oson yo‘li *kalit so‘zlarni aniqlash* uchun qo‘lda belgilangan til qoidalaridan foydalanishdir. Ko‘p ma’noli so‘zlar (omonimlar) mavjud bo‘lganda, belgilangan qoidalarning ishlashi murakkablasha boshlaydi. Mashinali o‘rgatish, chuqur o‘rganish va tabiiy tilni qayta ishlash usullari matn tahlilida katta hajmdagi matndan ma’no chiqarish uchun ishlatiladi. Kompaniyalar ushbu tushunchalardan rentabellikni, iste’molchilarning qoniqishini, innovatsiyalarni va hatto jamoat xavfsizligini yaxshilash uchun foydalanishi mumkin. Strukturalanmagan matnni tahlil qilish usullari *matn tasnifi, hissiyotlarni tahlil qilish, NER obyektlarni aniqlash* va *mavzuni modellashtirish* va boshqa NLP vazifalarini o‘z ichiga oladi. NLPning ushbu vazifalarining har biri turli kontekstlarda qo‘llaniladi.

NLPning ushbu vazifalarini bajarish uchun, birinchi navbatda, mashinalar inson tilini tushunishi va qayta ishlashi uchun nutq va matnlarni raqamli shaklga o‘tkazish zarur. Tabiiy tilni talqin qiladigan va tushunadigan aqlli tizimlarni ishlab chiqishda strukturlanmagan matnli ma’lumotlar bilan ishlash, ularni sun’iy intellekt metodari vositasida qayta ishlash maqsadida raqamli shaklga o‘tkazish lozim [1]. Kundalik hayotda ijtimoiy tarmoqlarda, onlayn servislarda katta hajmdagi matnli ma’lumotlar hosil bo‘ladi. Ushbu strukturlanmagan ma’lumotlarni chuqur o‘rganish metodlari orqali qayta ishlash uchun undagi *“shovqin” (noisy)* larni tozalash, strukturlangan shaklga o‘tkazish va mashinali o‘rganish algoritmi tomonidan tushunilishi mumkin bo‘lgan vektorlashtirilgan sonli formatlarga o‘zgartirish talab etiladi. Tabiiy tillarni qayta ishlash, mashinali yoki chuqur o‘rganish tamoyillarini matnlarni qayta ishlashda qo‘llash bugungi kunda NLPning dolzarb vazifalaridan hisoblanadi.

So‘nggi yillarda NLPda chuqur o‘rganish usullari tobora ommalashib bormoqda. Kirish va chiqish qatlamlari o‘rtasida bir nechta yashirin qatlamlarga ega **sun’iy neyron tarmoqlari (Artificial neural networks, ANN) chuqur neyron tarmoqlari (deep neural networks, DNN)** deb nomlanadi. Ushbu tadqiqot doirasida so‘nggi yillarda nashr etilgan ko‘plab maqolalar ko‘rib chiqildi va tahlil qilindi. Chuqur o‘rganish modellari takrorlanuvchi neyron tarmoqlari (recurrent neural networks, RNN) va konvolyutsion neyron tarmoqlari (convolutional neural networks, CNN) kabi neyron tarmoqlari topologiyalari asosida tasniflanadi. RNN vaqt o‘tishi bilan shablonlarni aniqlaydi, CNN esa fazodagi shablonlarni aniqlay oladi.

Oldindan o‘rgatilgan so‘zlarni joylashtirish – bu tabiiy tildagi so‘zlarning umumiy semantikasi va lingvistik shablonlarini qamrab oluvchi so‘zlarning muayyan (fiksirlangan) uzunlikdagi vektor ko‘rinishlari. NLP tadqiqotchilari bunday tasvirlarni olishning turli usullarini taklif qilishdi. So‘zni joylashtirish bir nechta NLP ilovalarida foydali ekanligi A.Moreo va boshalar tominidan ko‘rsatilgan [4].

So‘zlarni joylashtirish usullarini **an’anaviy**, **taqsimlangan** va **kontekstli** modellarga ajratish mumkin (1-rasm).



1-rasm. *So‘zlarni joylashtirish yondashuvlari*

So‘zlarni joylashtirishning an’anaviy yondashuvi NLPda chastotaga asoslangan modellar deb ataladi va *so‘zlar paketiga (Bag of Words, BoW)* [5,6], *N-gram* va *TF-IDF (Term Frequency-Inverse Document Frequency)* [7] kabi modellarga bo‘linadi.

So‘zlarni joylashtirishning tasqimlangan yondashuvi NLPda so‘zlarni statistik joylashtirish deb ataladi va vektor fazo modeli *(vector space model VSM)* [8,9,10], *yashirin semantik tahlil (latent semantic analysis, LSA)* [11], *yashirin Dirixle ajratish (latent Dirichlet allocation,LDA)* [12], *neyron ehtimollik tili modeli (neural probabilistic language model, NPLM)* [13],

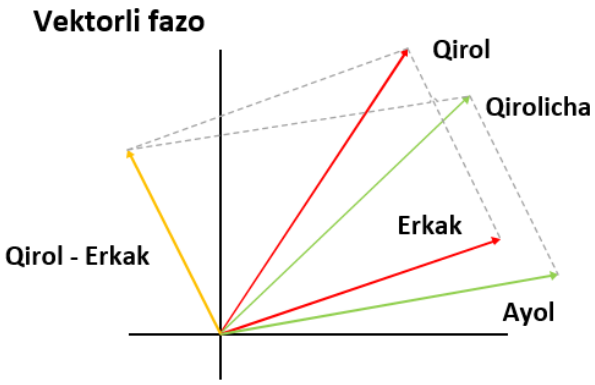
Word2Vec, *GloVe* va *Fasttext* modellaridan iborat [1,14,15].

So‘zlarni joylashtirishning kontekstli modellari *Embeddings from Language Models (ELMo)* [16], *Generative pre-training(GPT)* [17] va *Bidirectional encoder representations from transformers (BERT)* [18] kabi **avtoregressiv** va **avtokodlash** modellarga bo‘linadi.

Ushbu tadqiqotda 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 [19, 20, 21]. Biroq ushbu tamoyillar orqali strukturlanmagan, boshlang‘ich ishlov berilmagan matnni qayta ishlab bo‘lmaydi.

So‘zlarni joylashtirish – matndagi so‘zlarning ma’nosini saqlashga harakat qiladigan **n** o‘lchovli taqsimlangan tasvir [19, 22, 23]. Chuqur o‘rganish modellari ma’lumotlarning ierarxik ko‘rinishlarini o‘rganish uchun bir nechta hisoblash qatlamlaridan foydalanadi. Bugungi kunda chuqur o‘rganish bilan ifodalangan so‘zni joylashtirish usuli katta e’tiborga sazovor bo‘lmoqda.

Ushbu maqolada AI chuqur o‘rganish modellari: *Word2Vec*[1], *GloVe*[2] va *FastText*[3] vositasida o‘zbek tili matnlarini qayta ishlash usullari ko‘rib chiqiladi va til korpusi ustida sinovdan o‘tkaziladi. Matnli ma’lumotlar uchun statistik hisoblashlarga asoslangan an’anaviy usullar NLPda “so‘zlar sumkasi” (Bag of Words, BoW) modeli sifatida talqin qilinadi [5,6]. An’anaviy usullarga *terminlar chastotalari (term frequencies)*, *TF-IDF (term frequency-inverse document frequency)* va *N-gramm* kabi usullarni misol sifatda keltirish mumkin [7]. Ushbu usullar matndan zarur xususiyatlarni ajratib olishning samarali usullari bo‘lsa-da, modelning strukturlanmagan so‘zlar sumkasini shakllantirishga asoslangan. Shu sababli statistik hisoblashlarga asoslangan usullar orqali matnning sonli formati shakllantirilganda, matndagi *semantik*, *struktural* va *kontekst* kabi qo‘shimcha ma’lumotlar yo‘qotiladi. Bu esa ba’zi NLP vazifalarida berilgan matn so‘zlariga mos sonli vektorlarni shakllantirishda (2-rasm) *so‘zlarni joylashtirish (word embeddings)ga* asoslangan yanada murakkab modellarni o‘rganish uchun yetarlicha asos bo‘ladi.



2-rasm. *So‘zlarni sonli vektor shaklida ifodalash*

Ilmiy tadqiqotlar tahlili

Matn tahlili uchun samarali va chuqur o‘rganish yondashuvini tanlash murakkab masala hisonlanadi. Chunki ma’lumotlar to‘plamining hajmi, turi va maqsadi turlicha bo‘lib, qoyilgan NLP vazifasiga bo‘gliq bo‘ladi. Tadqiqotchilar so‘zning ma’nosini samarali tasvirlash va qayta ishlash uchun turli xil so‘zlarni joylashtirish modellari taqdim etishgan. So‘zlarni joylashtirish modellari yillar davomida rivojlanib, lug‘atdan tashqari so‘zlarni samarali ifodalash va kontekstli so‘zning ahamiyatini tushunish uchun takomillashtirildi. Amalga oshirilgan ilmiy tadqiqotlar shuni ko‘rsatadiki, chuqur o‘rganish modeli ma’lumotlardan muhim shablonlarni olish orqali natijalarni muvaffaqiyatli bashorat qilishi mumkin [x1].

So‘zlarni joylashtirish – bu teglanmagan katta hajmli til korpusdan olingan *semantik*, *sintaktik* ma’nolarni ifodalash orqali so‘zlarning sonli vektor tasviridir [13]. Vektor tasvirlari vositasida matndagi semantik xususiyatlar va chuqur neyron tarmoqlardan foydalangan holda so‘zlar o‘rtasidagi lingvistik munosabatlarni aniqlashi tufayli tabiiy tillarni qayta ishlash (natural language processing, NLP) ilovalarida so‘zlarni joylashtirish keng qo‘llaniladi [24, 25, 26, 27]. So‘zlarni joylashtirish, odatda, mashinali

[1] https://en.wikipedia.org/wiki/Word2vec
[2] https://nlp.stanford.edu/projects/glove/
[3] https://research.fb.com/fasttext/

o‘rganish modellariga kirish ma’lumotlari sifatida qo‘llaniladi. Bu esa mashinali o‘rganish usullariga qayta ishlanmagan matn ma’lumotlarini kontekstuallashtirishga imkon beradi.

Bu zamonaviy tabiiy tilni qayta ishlash vazifalarida keng qo‘llaniladigan kuchli vosita bo‘lib, *semantik tahlil* [28], *axborot olish* [29, 30, 31, 32, 33], *o‘zaro bo‘gliqlikni tahlil qilish* [34, 35, 36], *hissiyotlarni tahlil qilish* [37, 38], *savol-javob tizimlari* [39, 40, 41, 42], *matnni umumlashtirish* [43, 44] va *mashina tarjimasi* [39, 45, 46] kabi masalalarni hal qilishda muhim ahamiyat kasb etadi. Yuqori sifatli tasvirni o‘rganish ushbu vazifalar uchun juda muhim, ammo “so‘zni joylashtirishning yaxshi modeli nima” degan savol ochiq bo‘lib qolmoqda.

Chuqur o‘rganishga asoslangan his-tuyg‘ularni tahlil qilish[4], matnni chuqur o‘rganishga asoslangan tasniflash[5] va so‘zlar joylashuvini o‘rgatish va baholash bo‘yicha[6] olib borilgan ilmiy tadqiqotlar so‘zlarni joylashtirish va chuqur o‘rganish modellarining ishlashini solishtirishga qaratilgan. Ushbu tadqiqotlarda turli NLP vazifalarida so‘zlarni joylashtirish umumiy yondashuvlari ham taqdim etadi.

Turli NLP ilovalarida so‘zlarni joylashtirish usulidan foydalanishdagi muvaffaqiyat tufayli ba’zi mavjud tadqiqotlar so‘z semantikasini miqdoriy jihatdan ifodalashda so‘zlarni joylashtirishni baholaydi. Ularning aksariyati turli yondashuvlar orqali ishlab chiqilgan so‘zlarni joylashtirishni baholashga qaratilgan. Word2Vec [25], GloVe [47] va FastText [48] kabi tabiiy tilga bog‘liq bo‘lmagan chuqur o‘rganish usullarining ishlab chiqilishi, turli tillar bitta taqsimlangan yondashguv asosida bir xil vektor fazosiga joylashtirilgan tillararo so‘zlarni joylashtirishning rivojlanishiga olib keldi. Bunday usullar turli darajadagi kuchga ega bo‘lgan ikki tilli nazorat signallaridan (so‘zlar, gaplar yoki hujjatlar darajasida) foydalanadi [49].

So‘zlarni joylashtirishni o‘rganishda asosiy vazifa ularning sifatini baholash uchun ko‘rsatkichlarni topishdir. Qiu va Jiang o‘z ilmiy tadqiqotlarida ushbu ko‘rsatkichlar tashqi ko‘rsatkichlar bo‘lishi mumkinligini, joylashtirishlar NLPning quyi vazifalari uchun kirish xususiyatlari sifatida ishlatilishi va ularning ishlashini baholashda ichki o‘rnatishlar sintaktik yoki semantik xususiyatlarni o‘z-o‘zidan qanchalik yaxshi egallashini to‘g‘ridan-to‘g‘ri tekshirdilar [50].

Kam resursli tillar uchun teglangan ma’lumotlar yo‘qligi yoki kamligi sababli ushbu tillarga tashqi usullar har doim ham qo‘llanilmaydi. Shuningdek, quyi oqim modeli komponentlarini joylashtirish maydonida so‘zlarning semantik ifodalanishida yaxshilanishni ko‘rsatmasdan turib, muayyan vazifalarda yuqori samaradorlikka erishish uchun sozlash mumkin [51].

Baroni va boshqalar [52] to‘rtta model, ya’ni DISSECT[7], CBOW [25], Word2Vec[8], Distributional Memory model[9], Collobert va Weston model[10] yordamida **2,8 milliard** tokendan iborat korpus yordamida yaratilgan so‘zlarning birinchi tizimli bahosini taqdim etdilar; ushbu modellarni **5** toifadagi **14** ta ma’lumotlar to‘plamida sinovdan o‘tkazdilar. Jumladan *semantik bog‘liqlik, sinonimlarni aniqlash, kontseptsiyalarni toifalash, tanlov imtiyozlari* va *analogiya*. Ular Word2Vec modelidagi CBOW usulining deyarli barcha yuqorida keltirilgan NLP vazifalarni sifatli bajarganini qayd etdilar [53, 54, 55, 56].

2008-yilda Schnabel va boshqalar [22, 23] Word2Vec metodining CBOW usulini 10 ta ma’lumotlar to‘plamidagi boshqa joylashtirishlardan ustun ekanligini aniqladilar.

4 https://doi.org/10.1016/j.inffus.2020.06.002
5 https://doi.org/10.1109/CAIDA51941.2021.9425290
6 https://doi.org/10.1007/s41060-021-00242-8
7 https://www.sciencedirect.com/science/article/pii/S1532046418301825?pes=vor#fn5
8 https://www.sciencedirect.com/science/article/pii/S1532046418301825?pes=vor#fn6
9 https://www.sciencedirect.com/science/article/pii/S1532046418301825?pes=vor#fn7
10 https://www.sciencedirect.com/science/article/pii/S1532046418301825?pes=vor#fn8

Ular, shuningdek, ikkita quyi vazifaga, ya’ni *ot so‘z birikmalarini ajratish* va *his-tuyg‘ularni tasniflash* uchun kiritish xususiyatlari sifatida o‘rnatishdan foydalangan holda tashqi baholashni o‘tkazdilar. Ular ham CBOW natijalarini eng yaxshilar qatoriga kiritishdi.

Ghannay va boshqalar [21] shunga o‘xshash ichki baholashni o‘tkazdilar. Ular qo‘shimcha ravishda Word2Vec, Skip-gram modellarini [25], CSLM so‘zlarni joylashtirish [57], tobelikka asoslangan so‘zlarni joylashtirish [27] va to‘rtta NLP vazifasiga birlashtirilgan so‘zlarni joylashtirishni, shu jumladan, *POS teglash, chunking, NER obyektni aniqlash, eslatmani aniqlash (mention detection)* va ikkita *lingvistik vazifa*. Ular 4 milliard so‘zdan iborat **Gigaword** korpusida ushbu so‘zlarni joylashtirishni o‘rgatishdi va o‘zaro bog‘likka asoslangan so‘zlarni joylashtirish NLP vazifalarida eng yaxshi samaradorlikni sezilarli yaxshilanishga olib kelishini aniqladilar. Nayak va boshqalarning tadqiqoti [20] so‘zlarni joylashtirishni baholashda sintaktik va semantik xususiyatlarni sinab ko‘rish va baholash vazifalari real so‘z ilovalariga yaqinroq bo‘lishini tavsiya qilgan.

Bugungi kunda NLP sohasi ilmiy tadqiqotchilari tomonidan so‘zlarni joylashtirish modellarining samaradorligi va aniqligini sinab ko‘rish uchun turli baholash usullari taklif qilingan. A.Bakarovning ta’kidlashicha, baholash usullari uchun ikkita asosiy toifa mavjud – *ichki* va *tashqi* baholovchilar [19]. Tashqi baholovchilar quyi oqim vazifasiga kirish funksiyasi sifatida so‘z qo‘shishdan foydalanadi; shu vazifaga xos ishlash ko‘rsatkichlaridagi o‘zgarishlarni o‘lchaydi. Masalan, nutq qismini teglash (part-of-speech, POS) [58], atoqli obyektni aniqlash (amed-entity recognition, NER) [59], hissiyotlarni tahlil qilish [60] va mashina tarjimasi [61] kabi NLP vazifalari ushbu turga misol sifatida keltirish mumkin. Tashqi baholovchilar hisoblash jihatidan katta vaqt va resursni talab qilganligi sababli WE masalasida ular bevosita qo‘llanilmasligi mumkin. Ichki baholovchilar NLPning muayyan vazifalaridan qat’i nazar WE sifatini sinovdan o‘tkazadilar. Ushbu turdagi baholovchilar so‘zlar orasidagi sintaktik yoki semantik munosabatlarni bevosita o‘lchaydilar. Tanlangan so‘rovlar to‘plamidagi va semantik jihatdan bog‘liq maqsadli so‘zlardagi vektorlarni sinovdan o‘tkazish natijasida yig‘ilgan ballar beriladi.

NLP vazifalarini hal qilishda chuqur o‘rganish (DL) usullaridan foydalanish juda yaxshi va samarali natijalarni berganligini qayd etish mumkin. DL usullaridagi asosiy kontseptsiya inson tomonidan o‘qilishi mumkin bo‘lgan gaplarni neyron tarmoqlarga etkazib berishdir. Shunda modellar ulardan qandaydir ma’lumotlarni olishlari mumkin bo‘ladi. Neyron tarmoqlari uchun matnli ma’lumotlarni qanday oldindan qayta ishlashni tushunish muhim. Neyron tarmoqlar raqamlarni kiritish sifatida qabul qilishi mumkin, ammo qayta ishlanmagan matnni emas. Shuning uchun biz bu so‘zlarni raqamli formatga aylantirishimiz kerak.

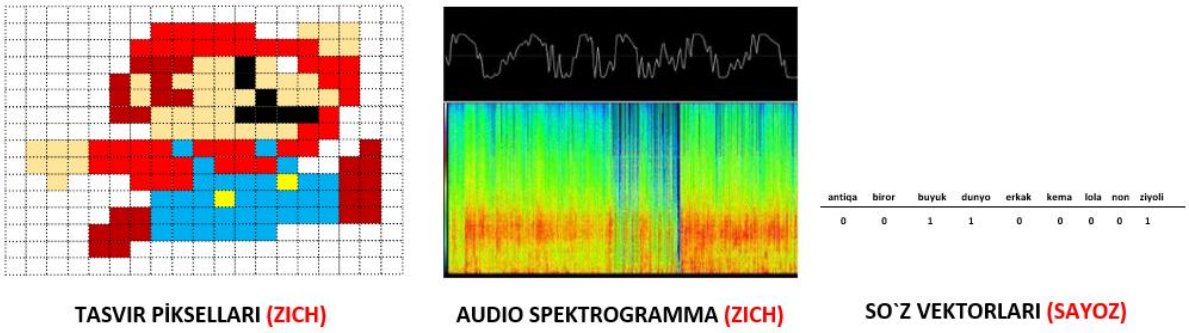
I. So‘zlarni joylashtirish (word embeddings) modellari

Og‘zaki nutq yoki tasvirni tanib olish tizimlarida barcha ma’lumotlar audio spektrogrammalar va tasvir piksellarining intensivligi kabi yuqori o‘lchamli ma’lumotlar to‘plami kabi zich xususiyat vektorlari shakliga ega bo‘ladi (3-rasm). Biroq strukturlanmagan matn ma’lumotlari haqida gap ketganda, ayniqsa, “so‘zlar sumkasi” kabi hisoblash modellarida o‘zining identifikatorlariga ega bo‘lishi mumkin bo‘lgan, so‘zlar o‘rtasidagi semantik aloqani saqlamaydigan so‘zlar to‘plami bilan ishlaymiz. Bu holda, matnli ma’lumotlar uchun juda katta o‘lchamli so‘z vektorlar shakllantiriladi va ulardan AI modellarida foydalanishda turli muammolar yuzaga keladi.

BoWga asoslangan modellar vositasida shakllantirilgan so‘zlar to‘plamidagi semantikani yo‘qotilishi va xususiyatlarning yetarli emasligi kabi kamchiliklarni bartaraf etish uchun *vektor fazo modellaridan (Vector Space Models, VSM)*[11] foydalanish mumkin. VSM orqali hosil qilingan modellarda matndagi semantik va kontekstual o‘xshashlik kabi xususiyatlar saqlanadi. Darhaqiqat, bir xil kontekstda paydo bo‘ladigan va ishlatiladigan so‘zlar semantik jihatdan bir-biriga o‘xshash bo‘lish ehtimoli katta bo‘ladi. Oddiy so‘zlar bilan aytganda, so‘zlar uni qurshab turgan kontekst bilan tavsiflanadi. Baroni tomonidan kontekstni hisoblash va bashorat qiluvchi

[11] https://en.wikipedia.org/wiki/Vector_space_model

semantik vektorlarni tizimli taqqoslash usullari tadqiq qilingan.



3-rasm. *Ovoz, rasm va matnga mos xususiyatlarni taqqoslash*

Olib borilgan tadqiqotlarga ko‘ra, bugun kunda kontekstli so‘z vektorlari uchun ikkita asosiy usul (yondashuv) mavjud:

- *hisoblash (statistika)ga asoslangan modellar (count-based methods);*
- *bashorat qilishga asoslangan modellar (predictive methods).*

Yashirin semantik tahlil (Latent Semantic Analysis, LSA)[12] kabi hisoblashga asoslangan usullar, bu so‘zlarning korpusdagi qo‘shni so‘zlari bilan qanchalik tez-tez uchrashining ba’zi statistik o‘lchovlarini hisoblash va ushbu o‘lchovlardan har bir so‘z uchun zich *so‘z vektorlari*ni yaratish uchun ishlatilishi mumkin.

Bashorat qilishga asoslangan modellar *sun’iy neyron tarmog‘iga*[13] asoslangan til modellari korpusdagi so‘zlar ketma-ketligiga qarab qo‘shni so‘zlardan ma’lum so‘zlarni bashorat qilishga harakat qiladi; bu jarayonda u bizga zich so‘zlarni joylashtirish imkonini beruvchi taqsimlangan tasvirlarni o‘rganadi. Ushbu maqolada bashorat qilish usullari asosida o‘zbek tilidagi matnli ma’lumotlarni qayta ishlash usullari ko‘rib chiqiladi.

O‘zbek tilidagi matnli ma’lumotlarni qayta ishlash va undan mazmunli xususiyatlarni ajratib olish uchun Python tili paketlaridan foydalangan holda, zarur funksiyalarni ishlab chiqish va til korpusida sinovdan o‘tkazish masalasini ko‘rib chiqamiz. Maqolada keltirilgan dasturiy kodlardan orqali hosil qilingan sonli vektorlardan murakkab NLP vazifalari hisoblangan mashinali o‘rganish tizimlarida foydalanish mumkin. Ushbu maqolada ishlatilgan barcha kodlardan boshqa NLP ilovalarida foydalanish uchun mualliflar tomonidan shakllantirilgan GitHub repozitoriysiga murojaat qilish mumkin.

```
import pandas as pd
import numpy as np
import re
import nltk
import matplotlib.pyplot as plt
pd.options.display.max_colwidth = 200
%matplotlib inline
```

Birinchi qadamda tahlil o‘tkazadigan bir nechta hujjatlarni aniqlaymiz:

```
corpus = ['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, ko‘k somsa, palov kabi bayram taomlari ulashildi.',
        'Parranda va quyon go‘shtidan taom tayyorlandi.',
        'Darvoza oldida somsa sotilar ekan.',
        'Tuyaqush boqish ham barakali biznesga aylandi.']
labels = ['ob-havo‘, 'ob-havo‘, 'hayvonlar', 'ovqat', 'ovqat', 'hayvonlar', 'ob-havo‘, 'hayvonlar']
corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus,
                          'Category': labels})
corpus_df = corpus_df[['Document', 'Category']]
```

[12] https://en.wikipedia.org/wiki/Latent_semantic_analysis

[13] http://www.scholarpedia.org/article/Neural_net_language_models

corpus_df

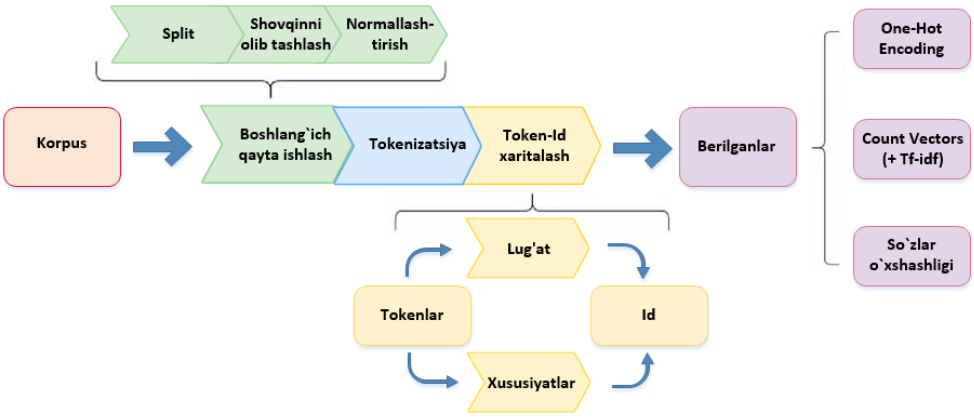
№	Document	Category
0	Kun bulutli bo‘lsa, yomg‘ir yog‘ishi mumkin.	ob-havo
1	Ertaga havo bulutli bo‘ladi.	ob-havo
2	Toshkent hayvonot bog‘iga kichkina tog‘ echkisi olib kelindi.	hayvonlar
3	Uch dona tuxumni idishda qaynatib oldi.	ovqat
4	Sumalak, ko‘k somsa, palov kabi bayram taomlari ulashildi.	ovqat
5	Parranda va quyon go‘shtidan taom tayyorlandi.	hayvonlar
6	Darvoza oldida somsa sotilar ekan.	ob-havo
7	Tuyaqush boqish ham barakali biznesga aylandi.	hayvonlar

Yuqoridagi sodda va kam sondagi gaplardan iborat namunaviy korpusimiz bir nechta toifalarga ('ob-havo‘,'hayvonlar','ovqat') ajratilgan. Biroq model natijalari samaradorligini va aniqligini oshirtish uchun katta hajmdagi til korpuslarida maqoladan foydalanish tavsiya etiladi. Til korpusidagi matnlarga mos sonli vektorlarni ishlab chiqishdan avval ularni boshlang‘ich qayta ishlash va normallashtirish kerak.

Matnni boshlang‘ich qayta ishlash

Matnli ma’lumotlarni tozalash va boshlang‘ich qayta ishlashning bir necha usullari mavjud. Tabiiy tilni qayta ishlashdagi *pipline konveyeri* muhim bosqich bo‘lib, ko‘plab NLP ilovalari uchun qo‘llaniladigan eng muhim boshlang‘ich amallar ketma-ketligidan iborat.

Til korpusidagi qayta ishlanmagan matn **boshlang‘ich qayta ishlanadi** va mashinali o‘rganish modeli uchun mos formatda ko‘rinishlariga keltiriladi. 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 (4-rasm). Ushbu jarayonda ma’lumotlardagi mavjud “shovqin”lar tozalanadi. Tozalangan ma’lumotlar NLP ilovasiga va mashinali o‘rganish modelining kiritish talablariga muvofiq turli shakl (shablon)larda taqdim etiladi.



4-rasm. *Til korpusi matnlarini boshlang‘ich qayta ishlash bosqichlari*

Yuqoridagi 4-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.

Ushbu maqolada asosiy e'tibor matndagi asosiy xususiyatlarni aniqlashga qaratilganligi sababli matndagi *maxsus belgilarni, qo‘shimcha bo‘shliqlarni, raqamlarni, nomuhim so‘zlarni* va *matn korpusining kichik harflarini olib tashlashga* qaratilgan dastlabki amallar bajariladi.

```
from string import punctuation
uz_stop_words = set([w.strip() for w in
open("d:\\nlp\\uz_stop_words.txt",encoding="utf8").read
lines()])
#print(uz_stop_words)
def filter_stop_words(train_sentences, stop_words):
    for i, sentence in enumerate(train_sentences):
        #print(i,sentence)
        #print(i,sentence.split())
        new_sent = [word for word in sentence.split() if
word not in stop_words]
        #print(new_sent)
        train_sentences[i] = ' '.join(new_sent)
    return train_sentences
```

Korpus matnlari ustida boshlang‘ich ishlov berishga qaratilgan funksiyalarni ishlab chiqqanimizdan so‘ng, ularni berilgan namunaviy korpusga qo‘llash mumkin:

```
corpus_text = open("d:\\nlp\\misol_text.txt",
encoding="utf8").readlines()
remove_terms = punctuation + '0123456789'
norm_corpus_text = [[word.lower() for word in sent if
word not in remove_terms] for sent in corpus_text]
norm_corpus_text = [''.join(tok_sent) for tok_sent in
norm_corpus_text]
norm_corpus_text = [tok_sent.strip() for tok_sent in
norm_corpus_text if len(tok_sent.split()) > 2]
norm_corpus_text =
filter_stop_words(norm_corpus_text, uz_stop_words)
print('Total lines:', len(corpus_text))
print('\nSample line:', corpus_text[4])
print('\nProcessed line:', norm_corpus_text[4])
```

```
Total lines: 8
Sample line: Sumalak, ko‘k somsa, palov kabi bayram taomlari ulashildi.
Processed line: sumalak ko‘k somsa palov bayram taomlari ulashildi
```

Keyingi qadamda, bugungi kundagi keng miqyosida foydalaniladigan so‘zlarni joylashtirish modellari va ularni o‘zbek tili korpusiga qo‘llash usullarini ko‘rib chiqamiz.

Word2Vec modeli

Ushbu model Google kompaniyasi tomonidan 2013-yilda yaratilgan bo‘lib, kontekstual va semantik o‘xshashlikni aks ettiruvchi so‘zlarning yuqori sifatli, taqsimlangan va uzluksiz zich vektor ko‘rinishlarini hisoblash va yaratish uchun chuqur o‘rganishga asoslangan bashoratli modeldir. Word2Vec nazoratsiz modellar bo‘lib, ular katta hajmdagi matn korpusini *qayta ishlashi*, korpusdagi unikal (takrorlanmaydigan) *so‘zlarning lug‘atini shakllantirilishi*, ushbu lug‘atni ifodalovchi vektor maydonidagi har bir so‘zga mos *sonli vektorlarni yaratishi* mumkin. Odatda, Word2Vec modelida so‘zni o‘rnatish vektorlarining hajmi belgilanadi va vektorlarning umumiy soni asosan lug‘at hajmiga teng bo‘ladi. Ushbu yondashuv asosida shakllantirilgan sonli vektorlar o‘lchovi an’anaviy Bag of Words (BoW) modellari yordamida qurilgan yuqori o‘lchamli siyrak vektor o‘lchovidan ancha kam.

Word2vec sayoz, ikki qatlamli neyron tarmog‘iga asoslangan bo‘lib, u matnlar korpusini qabul qiladi; natijada, korpusda ifodalangan har bir so‘z uchun sonli vektorni shakllantiradi. O‘rganilgan sayoz neyron tarmog‘ining asosiy xususiyati shundaki, korpusdagi o‘xshash kontekstli joylarda uchraydigan so‘zlar Word2Vec maydonida ham xuddi shunday joyga ega bo‘ladi. O‘xshash qo‘shni so‘zlarga ega so‘zlar semantik jihatdan o‘xshash bo‘lishi mumkinligi sababli bu Word2Vec yondashuvi semantik munosabatlarni saqlab qolishda juda yaxshi ekanligini anglatadi.

Bugungi kunda Word2Vec usulining ikki xil arxitekturasi mavjud bo‘lib, NLP vazifasiga ko‘ra so‘zlarni joylashtirish tasvirlarini yaratish uchun ishlatilishi mumkin:

- *CBOW modeli;*
- *Skip-gram modeli.*

Ushbu modellar dastlab Mikolov va boshqalar tomonidan amalga qo‘llanilgan bo‘lib, *“Distributed Representations of Words and Phrases and their Compositionality”*[14] va *“Efficient Estimation of Word Representations in Vector Space”*[15] kabi ilmiy maqolalarda modellar haqida boshlang‘ich tushunchalar berilgan.

[14] https://arxiv.org/pdf/1310.4546.pdf

[15] https://arxiv.org/pdf/1301.3781.pdf

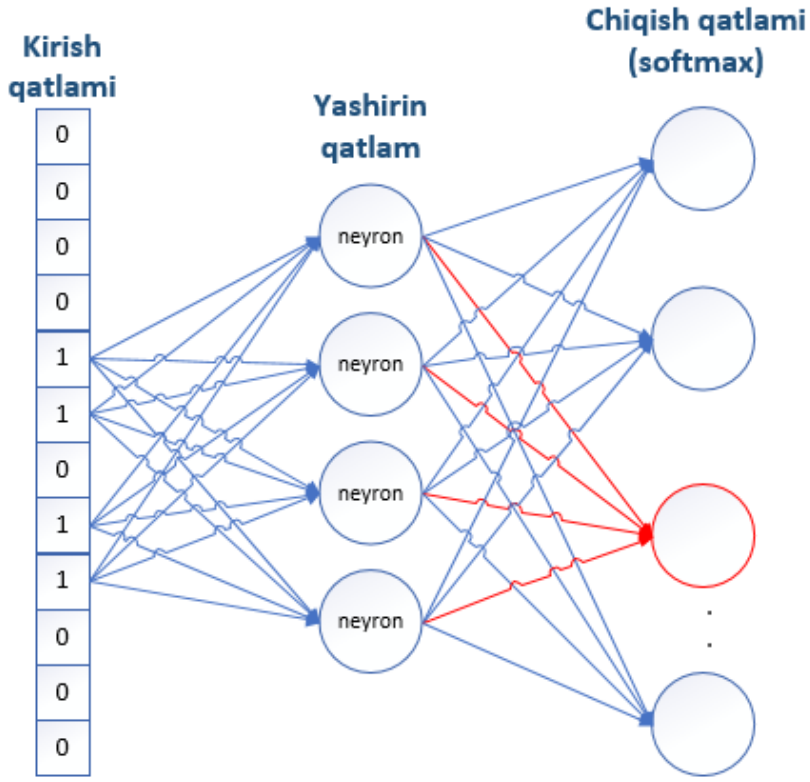
CBOW modeli

CBOW modeli arxitekturasi matndagi **kontekst so‘zlar (surrounding words)** asosida joriy **maqsadli so‘zni (the center word)** taxmin qilishga harakat qiladi. 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.

O‘zbek tilidagi *“men o‘zbek tilini yaxshi ko‘raman”* gapini CBOW metodida o‘qitish uchun **(context_window, target_word)** juftliklari hosil qilish lozim. Kontekst oynasi o‘lchami 2 ta teng bo‘lgan hol uchun yuqoridagi gapga mos quyidagi juftliklar shakllantiriladi: **([*men, tilini,*], *o‘zbek*)**, **([*o‘zbek, yaxshi*], *tilini*)**, **([*tilini, ko‘raman*], *yaxshi*)**. Shunday qilib, CBOW modeli kontekst so‘zlari asosida maqsadli so‘zni bashorat qilishga harakat qiladi.

CBOW nazoratsiz model bo‘lsa-da, modelga qo‘shimcha teglarsiz yoki ma’lumotsiz korpusni uzatish orqali korpusdagi zich so‘z birikmalarini shakllantirish mumkin. Biroq CBOW modeli orqali hosil qilingan vektorlarni tasniflash kerak bo‘ladi, ammo bu amal korpusning o‘zidan, hech qanday yordamchi ma’lumotsiz bajarilishi mumkin.

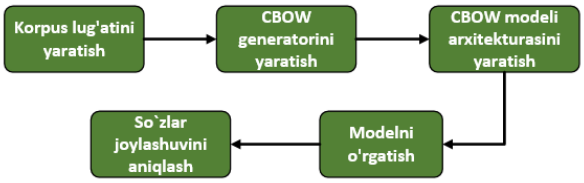
Endilikda CBOW modeli arxitekturasini chuqur o‘rganish tasniflash modeli sifatida modellashimiz mumkin. Shu sababli neyron tarmog‘ida kontekstdagi kirish so‘zlarini **X** sifatida belgilab, maqsadli so‘z **Y** ni bashorat qilishga harakat qilamiz. Aslida, CBOW modeli arxitekturasini yaratish skip-gram modelidan ko‘ra soddaroq hisonlanadi. Skip-gram metodida maqsadli so‘zdan bir qancha kontekstli so‘zlarni bashorat qilishga harakat qilinadi va bu jarayon ancha murakkab hisoblanadi.



5-rasm. *CBOW modeli arxitekturasi.*

CBOW modelini amalga oshirish

CBOW modeli orqali til korpusini mahsinali o‘rgatish jarayoni quyidagi beshta bosqich orqali amalga oshiriladi:



6-rasm. *CBOW modelini amalga oshirish bosqichlari*

1. Korpus lug‘atini yaratish

Birinchi bosqichda til korpusi lug‘atini shakllantirib, lug‘atdagi har bir unikal so‘zni ajratib olish va unga unikal raqamli identifikatorni mos qo‘yish kerak. Ushbu maqolada nisbatan kichikroq hajmdagi **10962** ta unikal so‘zdan iborat matn ustida amallar bajarilgan.

```
from tensorflow.keras.utils import to_categorical
import pickle
from tensorflow.keras.preprocessing import text
from tensorflow.keras.preprocessing import sequence
tokenizer = text.Tokenizer()
tokenizer.fit_on_texts(norm_corpus_text)
word2id = tokenizer.word_index

# unikal so‘zlarning lug‘atini shakllantirish
word2id['PAD'] = 0
id2word = {v:k for k, v in word2id.items()}
wids = [[word2id[w] for w in
text.text_to_word_sequence(doc)] for doc in
norm_corpus_text]
vocab_size = len(word2id)
print('Vocabulary Size:', vocab_size)
print('Vocabulary Sample:', list(word2id.items())[:50])
```

Vocabulary Size: 43
Vocabulary Sample: [('bulutli', 1), ('somsa’, 2), ('kun', 3), ('bo‘lsa’, 4), ('yomg‘ir', 5), ('yog‘ishi', 6), ('ertaga’, 7), ('havo‘, 8), ('bo‘ladi', 9), ('toshkent', 10), ('hayvonot', 11), ('bog‘iga’, 12), ('kichkina’, 13), ('tog‘', 14), ('echkisi', 15), ('olib', 16), ('kelindi', 17), ('uch', 18), ('dona’, 19), ('tuxumni', 20), ('idishda’, 21), ('qaynatib', 22), ('oldi', 23), ('sumalak', 24), ('ko‘k', 25), ('palov', 26), ('bayram', 27), ('taomlari', 28), ('ulashildi', 29), ('parranda’, 30), ('quyon', 31), ('go‘shtidan', 32), ('taom', 33), ('tayyorlandi', 34), ('darvoza’, 35), ('oldida’, 36), ('sotilar', 37), ('tuyaqush', 38), ('boqish', 39), ('barakali', 40), ('biznesga’, 41), ('aylandi', 42), ('PAD', 0)]

Ushbu koddagi **PAD** termini odatda kontekstli so‘zlarni kerak bo‘lganda belgilangan uzunlikka to‘ldirish uchun ishlatiladi.

2. CBOW (kontekst, maqsad) generatorini yaratish

Keyingi bosqichda maqsadli so‘z va uning atrofdagi kontekst so‘zlaridan iborat juftliklarni shakllantirish kerak. Ushbu maqolada CBOW modeli parametrlari sifatida maqsadli so‘z uzunligi **1** va uning atrofdagi kontekst uzunligi **window_size=2** bo‘lib, korpusdagi maqsadli so‘zdan oldin va keying 2 ta so‘zlar tahlil qilinadi.

```
def generate_context_word_pairs(corpus, window_size,
vocab_size):
    context_length = window_size*2
    for words in corpus:
        sentence_length = len(words)
        for index, word in enumerate(words):
            context_words = []
            label_word   = []
            start = index - window_size
            end = index + window_size + 1

            context_words.append([words[i]
                                  for i in range(start, end)
                                  if 0 <= i < sentence_length
                                  and i != index])
            label_word.append(word)

            x = sequence.pad_sequences(context_words,
maxlen=context_length)
            y = to_categorical(label_word, vocab_size)
            yield (x, y)

# Ba’zi namunalar uchun sinab ko‘rish
embed_size = 100
window_size = 2 # context window o‘lchovi
i = 0
for x, y in generate_context_word_pairs(corpus=wids,
window_size=window_size, vocab_size=vocab_size):
    if 0 not in x[0]:
        print('Context (X):', [id2word[w] for w in x[0]], '->
Target (Y):', id2word[np.argwhere(y[0])[0][0]])

        if i == 10:
            break
        i += 1
```

Context (X): ['kun', 'bulutli', 'yomg‘ir', 'yog‘ishi'] -> Target (Y): bo‘lsa
Context (X): ['toshkent', 'hayvonot', 'kichkina’, 'tog‘'] -> Target (Y): bog‘iga
Context (X): ['hayvonot', 'bog‘iga’, 'tog‘', 'echkisi'] -> Target (Y): kichkina
Context (X): ['bog‘iga’, 'kichkina’, 'echkisi', 'olib'] -> Target (Y): tog‘
Context (X): ['kichkina’, 'tog‘', 'olib', 'kelindi'] -> Target (Y): echkisi
Context (X): ['uch', 'dona’, 'idishda’, 'qaynatib'] -> Target (Y): tuxumni
Context (X): ['dona’, 'tuxumni', 'qaynatib', 'oldi'] -> Target (Y): idishda

Context (X): ['sumalak', 'ko‘k', 'palov', 'bayram'] -> Target (Y): somsa
Context (X): ['ko‘k', 'somsa’, 'bayram', 'taomlari'] -> Target (Y): palov
Context (X): ['somsa’, 'palov', 'taomlari', 'ulashildi'] -> Target (Y): bayram
Context (X): ['parranda’, 'quyon', 'taom', 'tayyorlandi'] -> Target (Y): go‘shtidan

Ushbu natija **X** ning kontekstdagi so‘zlarni qanday shakllantirishi haqida ma’lumot beradi va keying qadamda kontekstga asoslanib maqsadli **Y** so‘zini bashorat qilishga harakat qilamiz.

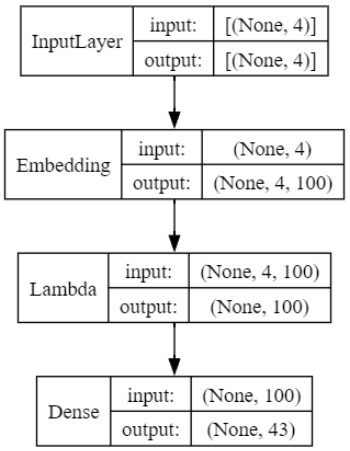
Misol uchun, agar asl matn 'Sumalak, ko‘k somsa, palov kabi bayram taomlari ulashildi.' bo‘lsa, u boshlang‘ich ishlov berish va nomuhim so‘zlarni olib tashlashdan so‘ng 'sumalak ko‘k somsa **palov** bayram taomlari ulashildi' ko‘rinishga keltiriladi. Kontekst sifatida ['sumalak','ko‘k','palov','bayram'] va maqsadli so‘z sifatida **"palov"** so‘zi tanlanadi.

3. CBOW modeli arxitekturasini yaratish

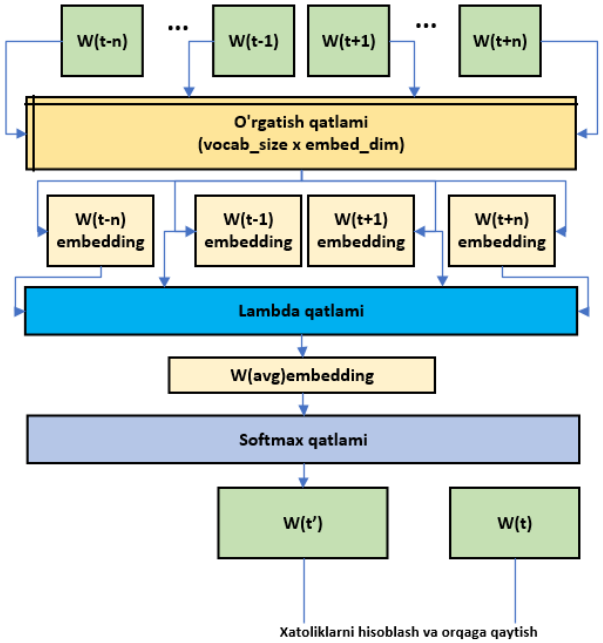
Ushbu bosqichda CBOW modeli uchun chuqur o‘rganish arxitekturasini yaratish uchun **keras** paketidan foydalanamiz. Bunda kiruvchi ma’lumotlar sifatida kontekst so‘zlari qabul qilinadi; o‘rgatish qatlamiga (tasodifiy og‘irliklar orqali) o‘tkaziladi. So‘zlar joylashuvi so‘zi lambda qatlamiga uzatiladi va so‘zlarning o‘rtacha vazni hisoblab chiqilib, softmax qatlami orqali maqsadli so‘z bashorat qilinadi. Softmax qatlami orqali bashorat qilingan maqsadli so‘z bilan haqiqiy maqsadli so‘z bilan taqqoslanadi va **categorical_crossentropy** yo‘qotish (xatolik) aniqlanadi hamda o‘rgatish qatlamini yangilash uchun har bir **davr (epoch)** bilan orqaga qaytish amalga oshiriladi. Quyidagi dasturiy kodda model arxitekturasi keltirilgan.

```
import keras.backend as K
from keras.models import Sequential
from keras.layers import Dense, Embedding, Lambda

# CBOW arxitecturasini shakllantirish
cbow = Sequential()
cbow.add(Embedding(input_dim=vocab_size,
output_dim=embed_size,
input_length=window_size*2))
cbow.add(Lambda(lambda x: K.mean(x, axis=1),
output_shape=(embed_size,)))
cbow.add(Dense(vocab_size, activation='softmax'))
cbow.compile(loss='categorical_crossentropy',
optimizer='rmsprop')
# model xulosasini ko‘rish
print(cbow.summary())
# model strukturasini vizuallash
from IPython.display import SVG
from keras.utils import model_to_dot
SVG(model_to_dot(cbow, show_shapes=True,
show_layer_names=False,
rankdir='TB',dpi=None).create(prog='dot',
format='svg‘))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 4, 100)            4300

lambda_3 (Lambda)            (None, 100)               0

dense_3 (Dense)              (None, 43)                4343

=================================================================
Total params: 8,643
Trainable params: 8,643
Non-trainable params: 0
_________________________________________________________________
None
```



Bizda **(2 x window_size)** o‘lchamdagi kontekstli kirish so‘zlari mavjud. Biz ularni **(vocab_size x embed_size)** o‘lchamdagi o‘rnatish qatlamiga uzatib, kontekstli so‘zlarning har biri uchun **(1 x embed_size)** o‘lchovli so‘zni aniqlash imkonini beradi.



7-rasm. *CBOW modeli orqali korpus matnlarini o'qitish jarayoni arxitekturasi*

Keyinchalik mazkur o'rnatishlarni o'rtacha hisoblash uchun lambda qatlamidan foydalaniladi va o'rtacha zich joylashtirishni **(1 x embed_size)** olish uchun softmax qatlamiga yuboriladi. Softmax qatlamida aniqlangan so'z va haqiqiy maqsadli so'z bilan *taqqoslanadi, yo'qotishni hisoblanadi, og'irliklar (o'rnatish qatlamida) sozlanadi, o'rgatish qatlamiga qaytiladi* va bu jarayon bir necha **davrlar (epoch)** bo'yicha (kontekst, maqsadli) so'z juftliklari uchun takrorlanadi. Yuqoridagi 7-rasmda CBOW modeli orqali korpus matnlarini o'qitish jarayoni arxitekturasi keltirilgan. Keyingi qadamda *(kontekst, maqsadli)* so'zlar juftliklaridan foydalanib CBOW modelni til korpusi asosida o'rgatish amalga oshiriladi.

4. Modelni o'rgatish

CBOW model katta hajmli korpusga qo'llash ko'proq vaqt talab etadi. Shu sababli quyida keltirilgan misolda o'qitish jarayoni **5** ta davrdan iborat.

```
from multiprocessing import cpu_count
epoch_count=cpu_count()-1
for epoch in range(1,epoch_count):
    loss = 0.
    i = 0
    for x, y in generate_context_word_pairs(corpus=wids,
window_size=window_size, vocab_size=vocab_size):
        i += 1
        loss += cbow.train_on_batch(x, y)
        if i % 100000 == 0:
            print('Processed {} (context, word)
pairs'.format(i))
    print('Epoch:', epoch, '\tLoss:', loss)
    print()
cbow.save("d:\\nlp\\cbowmodel03")
```

Epoch: 1 **Loss:** 166.0368070602417
Epoch: 2 **Loss:** 164.5994782447815
Epoch: 3 **Loss:** 163.4985933303833
Epoch: 4 **Loss:** 162.36359810829163
Epoch: 5 **Loss:** 161.17612719535828
Epoch: 6 **Loss:** 159.92315983772278
INFO:tensorflow:Assets written to: d:\nlp\cbowmodel03\assets

Ushbu model o'qitilgandan so'ng til korpusidagi o'xshash so'zlar bir xil og'irliklarga ega bo'lishi lozim.

5. So'zlar joylashuvini aniqlash

Korpus asosida shakllantirilgan lug'atdagi mavjud so'zlarga o'xshashlarini aniqlash uchun biz quyidagi koddan foydalanish lozim.

```
weights = cbow.get_weights()[0]
weights = weights[1:]
print(weights.shape)
pd.DataFrame(weights,
index=list(id2word.values())[1:]).head()
```

	0	1	2	3	4	5	6	7	8	9	...	90	91
somsa	-0.099008	0.001590	-0.033394	-0.034160	-0.036949	0.025167	-0.078199	0.062553	-0.038093	-0.008397	...	0.079543	-0.012132
kun	-0.039470	-0.005777	0.002456	0.069346	0.014947	0.113297	0.019107	0.047787	-0.025787	-0.034051	...	0.007428	-0.046224
bo'lsa	-0.030352	-0.009186	0.073287	-0.003744	-0.008187	0.061591	-0.034546	-0.059548	-0.029106	-0.005019	...	0.025981	-0.011866
yomg'ir	-0.056047	0.021026	0.012873	0.008626	-0.014609	0.008693	-0.097997	-0.042038	0.050018	0.019418	...	0.020456	-0.086164
yog'ishi	0.009126	0.005463	-0.011201	0.006949	-0.016802	0.028088	-0.062328	-0.063329	-0.007824	-0.014972	...	0.021245	-0.057241

5 rows × 100 columns

Shunday qilib, yuqorida keltirilgan jadvaldan lug'atdagi har bir so'zning o'lchami **(1x100)** zich joylashuvga ega ekanligini aniq ko'rishingiz mumkin. Quyida natija asosida ba'zi so'zlar uchun kontekstli o'xshash so'zlarni topishga harakat qilamiz. Buning uchun lug'atdagi barcha so'zlar orasida zich joylashtirish vektorlari asosida juft masofa matritsasi tuzamiz va so'ngra eng qisqa (evklid) masofaga qarab har bir so'zning n-yaqin qo'shnilarini topamiz.

```
from sklearn.metrics.pairwise import
euclidean_distances
# juftlik masofa matritsasini hisoblash
distance_matrix = euclidean_distances(weights)
print(distance_matrix.shape)
# kontekst jihatidan o'xshash so'zlarni aniqlash
```

```
similar_words = {search_term: [id2word[idx] for idx in
distance_matrix[word2id[search_term]-
1].argsort()[1:6]+1]
                for search_term in ['kun', 'bulutli', 'tuyaqush',
'somsa’]}
similar_words
```

```
{'kun': ['yomg‘ir', 'ertaga’, 'bo‘ladi', 'hayvonot',
'taomlari'],
 'bulutli': ['yog‘ishi', 'havo‘, 'taom', 'bo‘lsa’, 'parranda’],
 'tuyaqush': ['biznesga’, 'aylandi', 'barakali', 'sumalak',
'boqish'],
 'somsa’: ['taomlari', 'ulashildi', 'ko‘k', 'sotilar',
'hayvonot']}
```

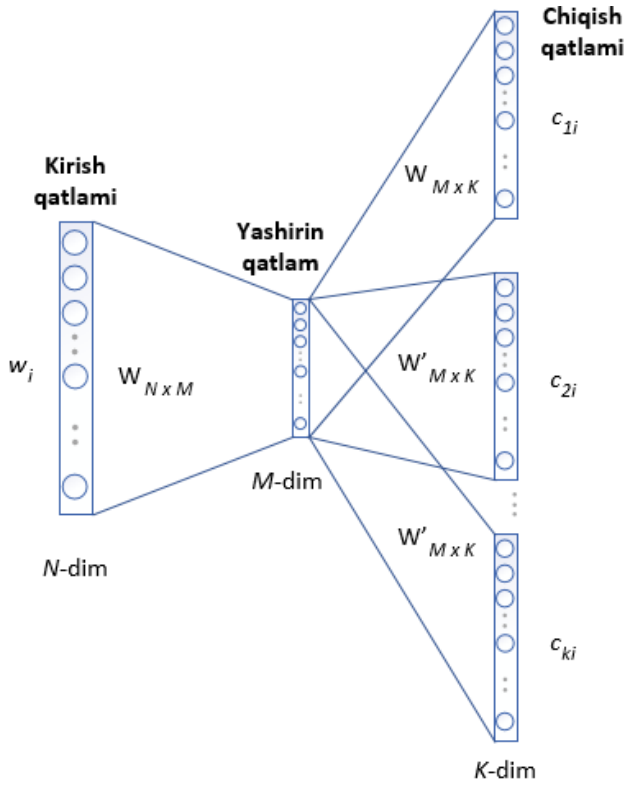
Hosil qilingan ba’zi ro‘yxatlar kontekstual ma’noga ega ekanligini qayd etish mumkin. Korpus matnlari uchun ko‘proq davrlar orqali o‘quv mashg‘ulotlarini amalga oshirish odatda yaxshi natijalar beradi.

Skip-gram modeli

Skip-gram modeli arxitekturasi odatda CBOW modelining teskarisiga erishishga harakat qiladi. Model **maqsadli so‘z (center_word)** asosida **kontekstli so‘zlarni (surrounding words)** taxmin qilishga harakat qiladi (8-rasm).

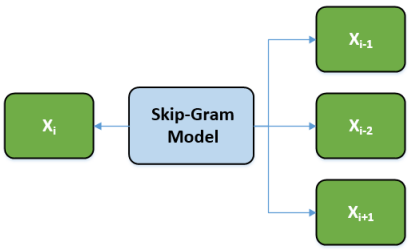
CBOW modelida namuna sifatida keltirilgan *«men o‘zbek tilini yaxshi ko‘raman»* gapini tahlil qiliamiz. CBOW modelidan foydalanib, kontekst oynasi o‘lchami 2 ta teng bo‘lgan hol uchun yuqoridagi gapga mos quyidagi **(context_window,target_word)** juftliklar shakllantirilgan:

([*men, tilini,*], *o‘zbek*), ([*o‘zbek, yaxshi*], *tilini*), ([*tilini, ko‘raman*], *yaxshi*).



8-rasm. *Skip-gram modeli arxitekturasi.*

Skip-gram modeli maqsadli so‘z asosida kontekstni bashorat qilish ekanligini hisobga olsak, model, odatda, har bir kontekst so‘zini maqsadli so‘zdan bashorat qilishga harakat qiladi. Demak, vazifa berilgan maqsadli “*o‘zbek*” so‘zidan [*men, tilini*] kontekstni yoki berilgan maqsadli “*yaxshi*” so‘ziga mos [*tilini, ko‘raman*] kabi juftliklarni bashorat qilishdan iborat bo‘ladi. Shunday qilib, model *target_word* asosida *context_window* so‘zlarini bashorat qilishga harakat qiladi (8-rasm).



9-rasm. *Skip-gram modeli*

Xuddi CBOW modelida muhokama qilganimizdek, ushbu Skip-gram arxitekturasini chuqur o‘rganish tasniflash modeli sifatida modellashtirishimiz kerak. Skip-gram modelida maqsadli so‘zni kirish sifatida qabul qilinib, kontekstdagi so‘zlar bashorat qilishga harakat qilinadi. Kontekstda bir nechta talabgor so‘zlar mavjudligi sababli bu jarayon biroz murakkab.

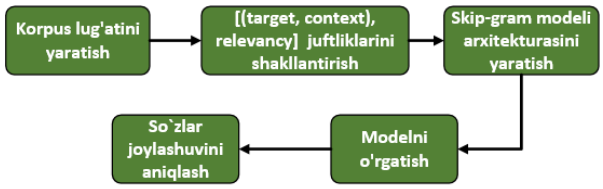
Qo‘yilgan vazifani soddalashtirish uchun har bir *(target, context_words)* juftligini *(target, context)* juftliklariga ajratish lozim. Shunda har bir kontekst faqat bitta so‘zdan iborat bo‘ladi. Shunday qilib, bizning oldingi ma’lumotlar to‘plamimiz (*tilini*, *o‘zbek*), (*tilini*, *yaxshi*), (*o‘zbek*, *men*), (*o‘zbek*, *tilini*) va boshqalar kabi juftlarga aylanadi. Ammo kontekstni tushunish uchun modelni qanday nazorat qilish yoki o‘rgatish kerak?

Buning uchun Skip-gram modelidagi (X,Y) juftliklar ustida ba’zi amallarni bajarishimiz lozim. Bu erda **X** – kiritish vektori, **Y** – yorliq (nishon). Ushbu holda **ijobiy kiritish namunalari** sifatida [*(target, context),1*] juftliklaridan foydalanib, bizni qiziqtirgan so‘z va kontekstdagi maqsadli so‘zga yaqin joylashgan kontekstli so‘zlarni aniylaymiz. Yuqoridagi juftlikdagi **1** qiymat, joriy juftlikning kontekstga mos ekanligini bildiradi.

Shuningdek, **salbiy kiritish namunalari** sifatida [[*(target, context), 0*] juftliklarni ajratib olamiz. Bunda maqsad yana bizni qiziqtiradigan so‘zdan iborat bo‘lib, tasodifiy so‘z bizning lug‘atimizdan tasodifiy tanlanadi va maqsadli so‘z bilan kontekst yoki aloqasi yo‘qligi aniqlanadi. Demak, salbiy yorliq **0** bu kontekstga aloqador bo‘lmagan juftlik ekanligini ko‘rsatadi. Ushbu mulohaza va belgilash asosida model so‘zlarning qaysi juftlari kontekstga mos kelishi, qaysi biri mos kelmasligini bilib olishi, semantik jihatdan o‘xshash so‘zlar uchun o‘xshash qiymatlarni mos qo‘yishi uchun asos bo‘ladi.

Skip-gram modelini amalga oshirish

Skip-gram modelidagi amallar ketma-ketligini mavjud til korpusiga qo‘llash orqali sinovdan o‘tkazamiz. Shuningdek, Skip-gram modeli asosida olingan natijalarni CBOW modeli natijalari bilan solishtiramiz. Skip-gram modeli quyidahi besh qadam orqali amalga oshiriladi:



10-rasm. *Skip-gram modelini amalga oshirish bosqichlari.*

1. Korpus lug‘atini yaratish

Xuddi CBOW modeliga o‘xshash tarzda birinchi bosqichda til korpusi lug‘ati va undagi har bir unikal so‘zni ajratib olamiz hamda unga unikal raqamli identifikatorni mos qo‘yamiz.

```
from tensorflow.keras.utils import to_categorical
import pickle
from tensorflow.keras.preprocessing import text
from tensorflow.keras.preprocessing import sequence
tokenizer = text.Tokenizer()
tokenizer.fit_on_texts(norm_corpus_text)
word2id = tokenizer.word_index
# korpusga mos unikal so‘zlar lug‘atini shakllantirish
word2id['PAD'] = 0
id2word = {v:k for k, v in word2id.items()}
wids = [[word2id[w] for w in
text.text_to_word_sequence(doc)] for doc in
norm_corpus_text]
vocab_size = len(word2id)
print('Vocabulary Size:', vocab_size)
print('Vocabulary Sample:', list(word2id.items())[:10])
```

Vocabulary Size: 43
Vocabulary Sample: [('bulutli', 1), ('somsa’, 2), ('kun', 3), ('bo‘lsa’, 4), ('yomg‘ir', 5), ('yog‘ishi', 6), ('ertaga’, 7), ('havo‘, 8), ('bo‘ladi', 9), ('toshkent', 10)]

Ushbu bosdiqda korpusdagi har bir unikal so‘z va raqamli identifikatorga ega bo‘lgan lug‘at shakllantirildi.

2. Skip-gram ([(target, context), relevancy]) generatorini yaratish

Yuqorida aytib o‘tilganidek, bizga so‘zlar juftligi va ularning ahamiyatini beradigan skip-gram generatorini ishlab chiqish lozim. Python tilidagi **keras** paketida **Skipgrams** yordamchi funksiyasi mavjud bo‘lib, CBOW metodida bo‘lgani kabi bu generatorni qo‘lda amalga oshirishimiz shart emas. Ushbu **Skipgrams** funktsiyasi so‘z indekslari (butun sonlar ro‘yxati) ketma-ketligini quyidagi shakldagi so‘z birikmalariga aylantiradi:

- *(so‘z, bir xil oynadagi so‘z), 1 yorlig‘i bilan (ijobiy namunalar);*
- *(so‘z, lug‘atdagi tasodifiy so‘z), 0 yorlig‘i bilan (salbiy namunalar).*

```
from keras.preprocessing.sequence import skipgrams
embed_size = 100
window_size = 2 # context_window o‘lchovi
# skip-gramlarni generatsiya qilish
```

```
skip_grams = [skipgrams(wid,
vocabulary_size=vocab_size, window_size=10) for wid
in wids]

# skip-gram namunalarini ko‘rish
pairs, labels = skip_grams[0][0], skip_grams[0][1]
for i in range(10):
   print("({:s} ({:d}), {:s} ({:d})) -> {:d}".format(
        id2word[pairs[i][0]], pairs[i][0],
        id2word[pairs[i][1]], pairs[i][1],
        labels[i]))
```

```
(kun (3), yog‘ishi (6)) -> 1
(yog‘ishi (6), bulutli (1)) -> 1
(yomg‘ir (5), quyon (31)) -> 0
(bulutli (1), kun (3)) -> 1
(kun (3), bulutli (1)) -> 1
(kun (3), yomg‘ir (5)) -> 1
(bo‘lsa (4), havo (8)) -> 0
```

Shunday qilib, yuqoridagi dastur kodi orqali berilgan korpus asosida kerakli skip-grammlar muvaffaqiyatli shakllantirildi; oldingi chiqishdagi namunaviy skip-grammalarga asoslanib, **0** yoki **1** yorliq asosida nimaning muhim, nimaning ahamiyatsiz ekanligini namoyish qilindi.

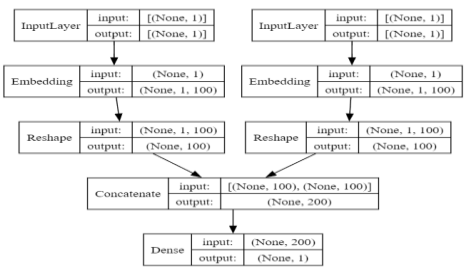
3. Skip-gram modeli arxitekturasini yaratish

Keyingi bosqichda Skip-gram modeli uchun chuqur o‘rganish arxitekturasini yaratish maqsadida **tensorflow** paketidagi **keras** modulidan foydalaniladi. Buning uchun bizning neyron tarmog‘iga kirish ma’lumotlari sifatida *maqsadli so‘z* va *kontekst* yoki *tasodifiy* so‘z juftligi uzatiladi. Ularning har biri o‘ziga xos **joylashtirish qatlami (embedding layer)ga** *tasodifiy og‘irliklar (random weights)* bilan uzatiladi. Asliyat va kontekst so‘zi uchun so‘zlar joylashuvi aniqlanganidan so‘ng ular **birlashma qatlami (merge layer)ga** uzatiladi, ikki vektorning skalyar ko‘paytmasi hisoblanadi. Keyingi qadamda ushbu skalyar ko‘paytma qiymati **zich sigmasimon qatlam(dense sigmoid layer)**ga uzatiladi. Ushbu qatlam so‘zlar juftligi kontekstga mos kelishiga yoki shunchaki tasodifiy so‘zlarga **(Y')** qarab **1** yoki **0** ni bashorat qiladi. Aniqlangan qiymat va haqiqiy tegishlilik yorlig‘i (**Y**)ni taqqoslash natijasida *mean_squared_error orqali* yo‘qotishni hisoblaymiz va o‘rnatish qatlami (embedding layer)ni yangilash uchun har bir **davr (epoch)** bilan **orqaga qaytish (backpropagation)** jarayonini amalga oshiramiz.

Quyidagi dasturiy kodda Skip-gram arxitekturasini tadbiq qilingan:

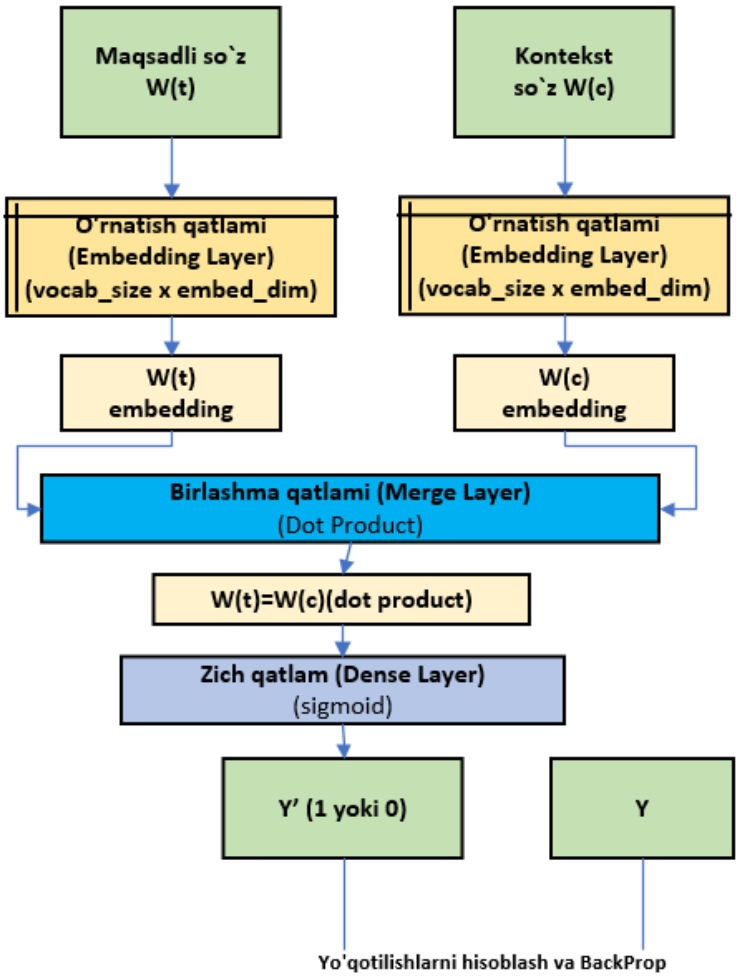
```
from tensorflow.keras.layers import concatenate, Dense,
Reshape,Embedding,Flatten,Input
from tensorflow.keras.models import Sequential
from keras.models import Model
inputs = Input(shape=(vocab_size,embed_size),
dtype='float32')
# skip-gram arxitekturasini shakllantirish
word_model = Sequential()
word_model.add(Embedding(vocab_size, embed_size,
                    embeddings_initializer="glorot_uniform"
,
                    input_length=1))
word_model.add(Reshape((embed_size, )))
context_model = Sequential()
context_model.add(Embedding(vocab_size, embed_size,
                embeddings_initializer="glorot_uniform",
                input_length=1))
context_model.add(Reshape((embed_size,)))
model_concat = concatenate([word_model.output,
context_model.output], axis=-1)
model_concat = Dense(1,
kernel_initializer="glorot_uniform",
activation="sigmoid")(model_concat)
model = Model(inputs=[word_model.input,
context_model.input], outputs=model_concat)
model.compile(loss="mean_squared_error",
optimizer="rmsprop")
# model xulosasini ko‘rish
print(model.summary())
# model tuzilishini tasvirlash
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
SVG(model_to_dot(model, show_shapes=True,
show_layer_names=False,
                rankdir='TB',dpi=None).create(prog='dot',
format='svg‘))
```

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
embedding_46_input (InputLayer  [(None, 1)]          0         []
)

embedding_47_input (InputLayer  [(None, 1)]          0         []
)

embedding_46 (Embedding)     (None, 1, 100)        4300      ['embedding_46_input[0][0]']

embedding_47 (Embedding)     (None, 1, 100)        4300      ['embedding_47_input[0][0]']

reshape_46 (Reshape)         (None, 100)           0         ['embedding_46[0][0]']

reshape_47 (Reshape)         (None, 100)           0         ['embedding_47[0][0]']

concatenate_21 (Concatenate)  (None, 200)          0         ['reshape_46[0][0]',
                                                               'reshape_47[0][0]']

dense_14 (Dense)             (None, 1)             201       ['concatenate_21[0][0]']

==================================================================================================
Total params: 8,801
Trainable params: 8,801
Non-trainable params: 0
```



Yuqorida keltirilgan chuqur o‘rganish modelini juda oddiy tushunish mumkin. Bizda har bir o‘quv misoli uchun bir juft kirish so‘zlari mavjud bo‘lib, ular yagona raqamli identifikatorga ega bo‘lgan bitta kirish maqsadli so‘zdan va yagona raqamli identifikatorga ega bo‘lgan bitta kontekst so‘zidan iborat. Agar ushbu juftliklar ijobiy namuna bo‘lsa, so‘z kontekst ma’nosiga ega, kontekstli so‘z va bizning yorlig‘imiz **Y=1**, aks holda u salbiy namuna bo‘lsa, so‘z kontekstual ma’noga ega emas, shunchaki tasodifiy so‘z va bizning yorlig‘imiz **Y=0**.

Biz ularning har birini o‘z o‘lchamiga (**vocab_size x embed_size**) ega bo‘lgan *o‘rnatish qatlamiga (embedding layer)* uzatamiz. Ushbu qatlam bizga ikkita so‘zning har biri uchun zich so‘zlarni joylashtirish imkonini beradi.



11-rasm. *Skip-gram modeli arxitekturasi*

Keyingi bosqichda *birlashma qatlami (merge layer)*dan foydalangan holda ushbu ikkita joylashtirishning skalyar ko‘paytmasi hisoblanadi. So‘ngra ushbu qiymat *zich sigmasimon qatlam (dense sigmoid layer)ga* uzatiladi va **1** yoki **0** natija olinadi. Olingan natija haqiqiy **Y** (**1** yoki **0**) yorlig‘i bilan taqqoslanadi va yo‘qotish qiymati hisoblanadi. Og‘irliklar sozlash uchun xatolar o‘rnatish qatlamida qayta taqsimlanadi va bu jarayon barcha *(target, context)* juftliklar uchun bir nechta davrlar orqali takrorlanadi. Yuqoridagi 11-rasmda Skip-gram modeli arxitekturasi keltirilgan. Keyingi qadam Skip-gram modelni o‘rgatish bosqichi amalga oshiriladi.

4. Modelni o‘rgatish

O‘zbek tili korpusi matnlariga modelni to‘liq qo‘llash CBOW modeliga qaraganda kamroq vaqt talab etadi. Shu sababli quyida keltirilgan misolda berilgan til korpusi **5** davr davomida sinovdan o‘tkazildi. Quyida keltirilgan dastur kodida ba’zi parametrlarni o‘zgartirish orqali sinov natijalarini yaxshilash mumkin.

```
for epoch in range(1, 6):
    loss = 0
    for i, elem in enumerate(skip_grams):
        pair_first_elem = np.array(list(zip(*elem[0]))[0], dtype='int32')
        pair_second_elem = np.array(list(zip(*elem[0]))[1], dtype='int32')
        labels = np.array(elem[1], dtype='int32')
        X = [pair_first_elem, pair_second_elem]
        Y = labels
        if i % 10000 == 0:
            print('Processed {} (skip_first, skip_second, relevance) pairs'.format(i))
        loss += model.train_on_batch(X,Y)

    print('Epoch:', epoch, 'Loss:', loss)
model.save("d:\\nlp\\skipgrammodel03")
```

```
Processed 0 (skip_first, skip_second, relevance) pairs
Epoch: 1 Loss: 2.0351533591747284
Processed 0 (skip_first, skip_second, relevance) pairs
Epoch: 2 Loss: 2.009832590818405
Processed 0 (skip_first, skip_second, relevance) pairs
Epoch: 3 Loss: 1.9986148029565811
Processed 0 (skip_first, skip_second, relevance) pairs
Epoch: 4 Loss: 1.9898146241903305
Processed 0 (skip_first, skip_second, relevance) pairs
Epoch: 5 Loss: 1.9821369647979736
INFO:tensorflow:Assets written to: d:\nlp\skipgrammodel03\assets
```

Skip-gram modeli o‘qitilgandan so‘ng o‘xshash so‘zlar o‘rnatish qatlamiga asoslangan o‘xshash og‘irliklarga ega bo‘lishi talab etiladi; biz buni quyidagi bosqichda keltirilgan dastur kodi orqali sinab ko‘ramiz.

5. So‘zlar joylashuvini aniqlash

Til korpusidagi so‘zlar asosida hosil qilimgan lug‘atdagi so‘zlarga mos o‘xshash so‘zlarni aniqlash uchun quyida keltirilghan koddan foydalanish mumkin. E’tibor bering, bizni faqat maqsadli so‘zni o‘rnatish qatlami qiziqtiradi, shuning uchun biz o‘rnatishlarni **word_model** o‘rnatish qatlamidan hosil qilamiz.

```
print(model.layers)
word_model = model.layers[0]
word_embed_layer = model.layers[2]
weights = word_embed_layer.get_weights()[0][0:]
print(weights.shape)
pd.DataFrame(weights, index=id2word.values()).head()
```

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96	97	98	99
bulutli	-0.139875	-0.185174	-0.026181	0.191785	-0.146155	-0.013704	-0.008815	0.129678	-0.133712	-0.149228	...	0.141146	0.025068	-0.137051	-0.079421	-0.007101	-0.088469	-0.204443	0.094920	-0.062644	0.120715
somsa	-0.065589	-0.034056	0.002109	-0.049096	-0.050671	-0.186366	-0.170520	-0.120491	0.047598	-0.051284	...	-0.046324	-0.069038	0.143228	-0.099867	-0.202370	0.011071	-0.196643	0.166952	-0.176937	-0.142399
kun	-0.042496	-0.124861	-0.143368	-0.074580	0.176426	0.125792	0.067645	0.114272	0.107859	-0.096484	...	0.050309	-0.057864	-0.104248	0.116224	-0.116168	-0.154794	-0.021090	-0.007712	0.043084	0.180451
bo'lsa	-0.104660	-0.188902	-0.148654	-0.130386	0.000261	0.128668	0.190925	-0.115356	0.091946	0.014238	...	0.172624	0.139696	-0.034644	0.053437	-0.003889	0.006338	-0.034796	-0.030430	0.199156	-0.055640
yomg'ir	0.195015	-0.109596	-0.030353	-0.079414	-0.101665	0.124057	-0.121494	0.194748	0.197448	0.186360	...	0.173882	-0.198572	-0.023881	0.095972	0.031008	-0.122185	0.135128	-0.186325	-0.197245	0.108158

5 rows × 100 columns

Shunday qilib, lug‘atdagi har bir so‘zning CBOW modelida tasvirlanganidek, (1x100) o‘lchamdagi zich joylashuvga ega ekanligini ko‘rish mumkin. Lug‘atdagi har bir so‘z uchun juftlik masofa ko‘rsatkichini hisoblash uchun ushbu zich joylashtirish vektorlariga Evklid masofasi ko‘rsatkichini qo‘llaymiz. So‘ngra eng qisqa (evklid) masofa asosida har bir so‘zning **n** ta eng yaqin qo‘shnilarini aniqlaymiz.

```
from sklearn.metrics.pairwise import euclidean_distances
distance_matrix = euclidean_distances(weights)
print(distance_matrix.shape)
similar_words = {search_term: [id2word[idx] for idx in distance_matrix[word2id[search_term]-1].argsort()[1:6]+1]
                 for search_term in ['kun', 'bulutli', 'tuyaqush', 'somsa']}
similar_words
```

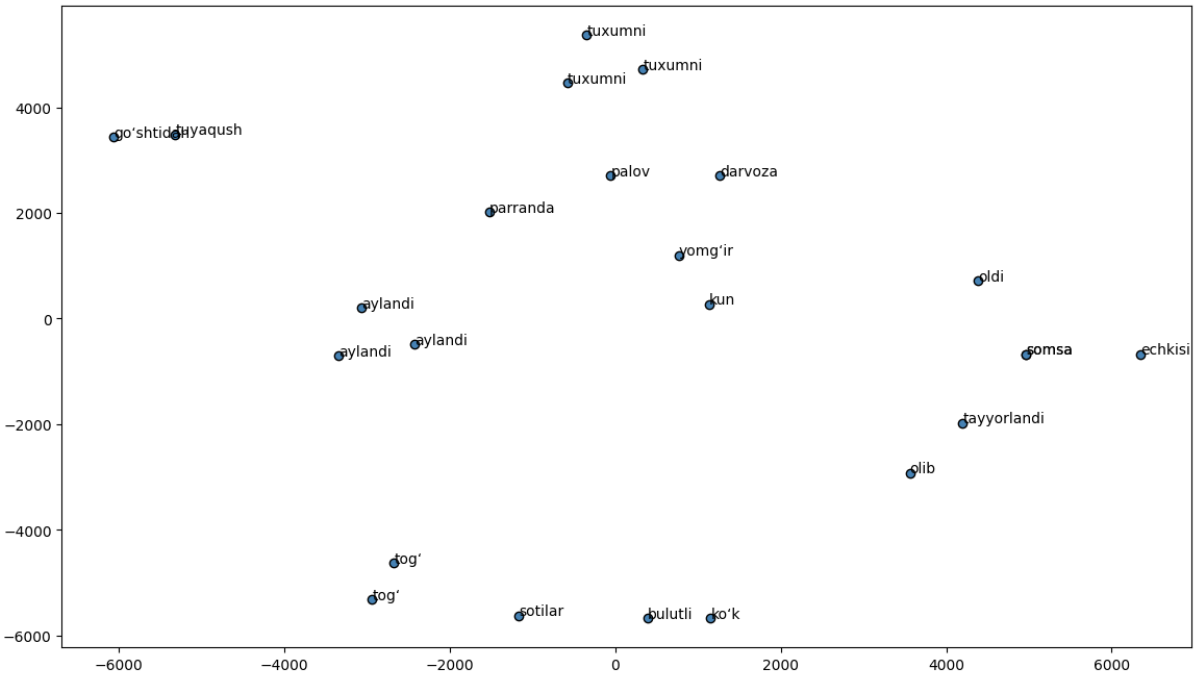
```
{'kun': ['tayyorlandi', 'olib', 'tuxumni', 'aylandi', 'tog‘'],
 'bulutli': ['darvoza’, 'aylandi', 'go‘shtidan', 'sotilar', 'tog‘'],
 'tuyaqush': ['tuxumni', 'somsa’, 'echkisi', 'yomg‘ir', 'ko‘k'],
 'somsa’: ['palov', 'tuxumni', 'parranda’, 'aylandi', 'oldi']}
```

Natijalardan yaqqol ko‘rinib turibdiki, qiziqtirgan so‘zlarning har biri uchun juda ko‘p o‘xshash so‘zlar ma’noga ega; bu natija CBOW modelimiz bilan solishtirganda yaxshiroq natijalarni qaytardi. Keyingi qadamda 2-D o‘lchamli fazoda **t-stoxastik qo‘shnini** (t-distributed stochastic neighbor[16]) ifodalovchi **t-SNE** yordamida ushbu so‘zlarni joylashtirishni shakllantiramiz.

```
from sklearn.manifold import TSNE
words = sum([[k] + v for k, v in similar_words.items()], [])
words_ids = [word2id[w] for w in words]
```

[16] https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding

```
word_vectors = np.array([weights[idx] for idx in
words_ids])
print('Total words:', len(words), '\tWord Embedding
shapes:', word_vectors.shape)
tsne = TSNE(n_components=2, random_state=0,
n_iter=10000, perplexity=3)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(word_vectors)
labels = words
plt.figure(figsize=(14, 8))
plt.scatter(T[:, 0], T[:, 1], c='steelblue', edgecolors='k')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
   plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0),
textcoords='offset points')
```



12-rasm. t-SNE yordamida skip-gram so‘zlarni o‘rnatishni vizualizatsiya qilish

Gensim paketi vositasida Word2Vec modellari ishlab chiqish

Yuqorida keltirilgan dasturiy kodlar vositasida katta bo‘lmagan til korpuslarini o‘qtish va natijalarni olish mumkin. Biroq katta hajmdagi til korpuslarida ushbu kodlardan foydalanib bo‘lmaydi. R.Radim tomonidan yaratilgan **gensim**[17] paketi vositasida Word2Vec modelini *mustahkam, samarali* va *kengaytiriladigan* tartibda tatbiq etish mumkin. Mualliflar tomonidan ishlab chiqilgan o‘zbek tili korpusi matnlari ustida gensim paketidan foydalanib, Word2Vec modeli sonli vektorlarini hosil qilish masalasini ko‘rib chiqamiz. Keyingi qadamlarda Word2Vec modelini hosil qilish uchun quyidagi to‘rtta parametrdan foydalaniladi:

– **size:** *so‘zlarni joylashtirish o‘lchovi;*
– **window:** *kontekst oynasining o‘lchovi;*
– **min_count:** *minimal so‘zlar soni;*
– **sample:** *so‘zlarning statistik qiymatining minimal qiymati.*

Word2Vec modeli shakllantirilganidan so‘ng korpus asosida shakllantirilgan lug‘atdagi har biri so‘z uchun eng yaqin o‘xshash so‘zlar ro‘yxarini aniqlash mumkin.

```
from gensim.models import word2vec
tokenized_corpus = [document.split(' ') for document in
norm_corpus_text]
# Turli parametrlar uchun qiymatlarni o‘rnatish
vector_size=100
window_context = 30          # Kontekst oynasining
o‘lchovi
min_word_count = 1           # Minimal so‘zlar
soni
sample = 1e-3              # So‘zlar uchun minmal qiymat
```

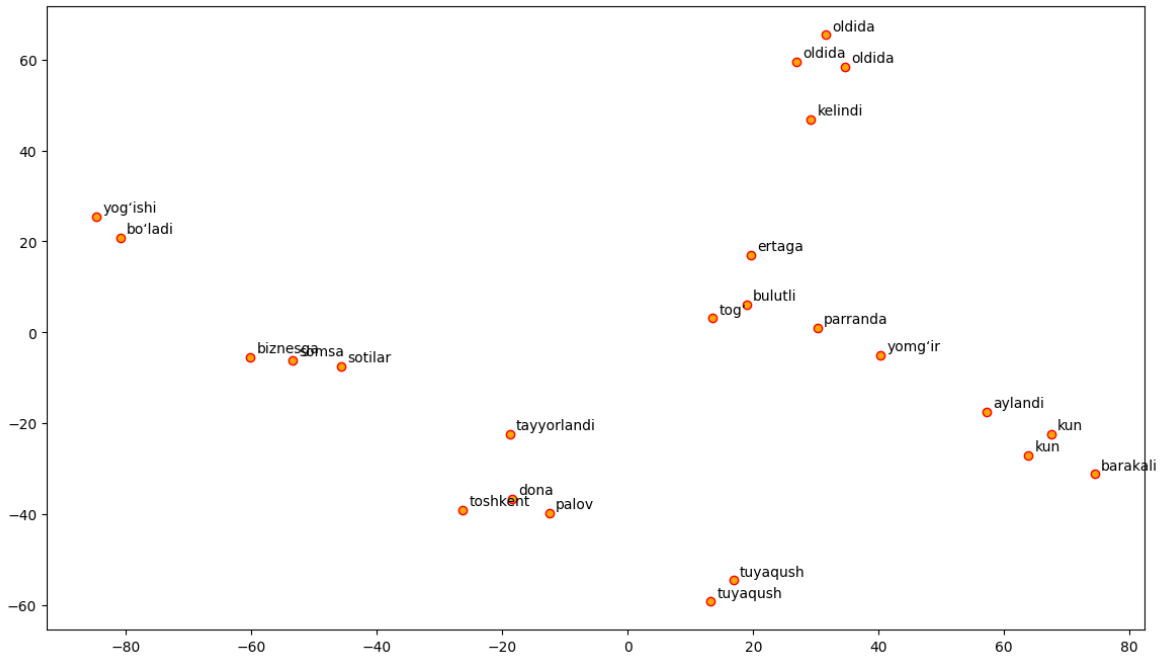
[17] https://radimrehurek.com/gensim/

```
w2v_model = word2vec.Word2Vec(tokenized_corpus, vector_size=vector_size,
                              window=window_context, min_count=min_word_count,
                              sample=sample)
# gensim modeliga asoslangan modeldagi o‘xshash so‘zlarni ko‘rish
similar_words = {search_term: [item[0] for item in w2v_model.wv.most_similar([search_term], topn=5)]
                 for search_term in ['kun', 'bulutli', 'tuyaqush', 'somsa’]}
similar_words
```

```
{'kun': ['aylandi', 'barakali', 'oldida’, 'tuyaqush', 'yog‘ishi'],
 'bulutli': ['ertaga’, 'tog‘', 'oldida’, 'tayyorlandi', 'parranda’],
 'tuyaqush': ['dona’, 'kelindi', 'oldida’, 'kun', 'palov'],
 'somsa’: ['sotilar', 'biznesga’, 'bo‘ladi', 'yomg‘ir', 'toshkent']}
```

Gensim paketi vositasida hosil qilingan o‘xshash so‘zlar bizni qiziqtirgan so‘zlar bilan ko‘proq bog‘liq; ushbu modeldagi davrlar soni kattaroq bo‘lgani sababli kontekstli joylashtirish imkonini beradi. Hosil qilingan model asosida namunaviy so‘zlar va ularga mos bo‘lgan o‘xshash so‘zlarni t-SNE bilan 2 o‘lchamli fazoda shakllantiramiz.

```
from sklearn.manifold import TSNE
words = sum([[k] + v for k, v in similar_words.items()], [])
wvs = w2v_model.wv[words]
tsne = TSNE(n_components=2, random_state=0, n_iter=10000, perplexity=2)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(wvs)
labels = words
plt.figure(figsize=(14, 8))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points')
```



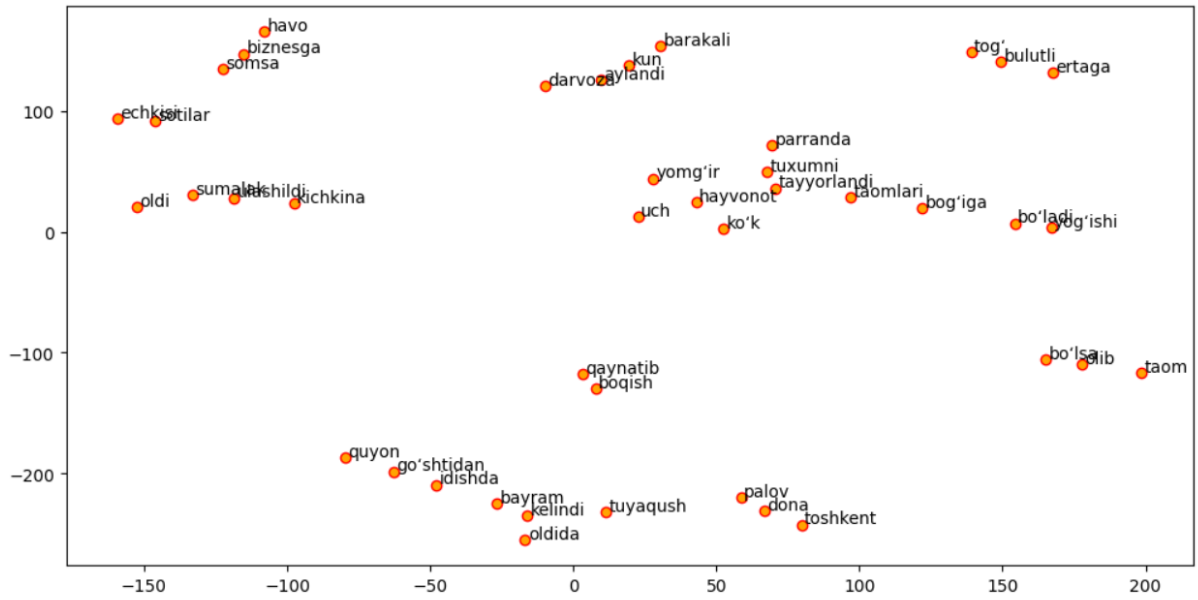
13-rasm. *t-SNE yordamida skip-gram so‘zlarni o‘rnatishni vizualizatsiya qilish*

Ushbu 13-rasmdagi qizil doiralar orqali Word2Vec modelidagi qiziqarli assotsiatsiyalar namoyish qilingan. Yuqorida tasvirlangan 13-rasmdan bizning modelimizdagi so‘z birikmalariga asoslanib bir-biriga yaqin ekanligini aniq ko‘rishimiz mumkin.

Word2Vec funksiyalarini ML vazifalarida qo‘llash

Gensim paketi orqali hosil qilingan Word2Vec modelidan NLPdagi klasterlash vazifasini hal qilishda foydalanish mumkin. Buning uchun o‘zbek tili korpusi matnlaridan foydalanib, Word2Vec modelini shakllantiramiz va turli tahlillarni amalga oshiramiz:

```
# word2vec modelni shakllantirish
tokenized_corpus = [document.split(' ') for document in
norm_corpus_text]
# Turli parametrlar uchun qiymatlarni o‘rnatish
window_context = 10          # Kontekst oynasining
o‘lchovi
min_word_count = 1           # Minimal so‘zlar
soni
sample = 1e-3                # So‘zlar uchun minmal qiymat
w2v_model = word2vec.Word2Vec(tokenized_corpus,
vector_size=vector_size,
                              window=window_context, min_count
= min_word_count,
                              sample=sample)
# so‘zlar o‘xshashligini namoyish qilish
from sklearn.manifold import TSNE
words = w2v_model.wv.index_to_key
wvs = w2v_model.wv[words]
tsne = TSNE(n_components=2, random_state=0,
n_iter=5000, perplexity=2)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(wvs)
labels = words
plt.figure(figsize=(12, 6))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0),
textcoords='offset points')
```



14-rasm. *Kichik hajmdagi korpus asosida shakllantirilgan word2vec modelining so‘z birikmalarini vizualizatsiya qilish*

Namunaviy korpus sifatida kichik hajmga ega korpus asosida shakllantirilgan so‘zlar o‘xshashligining aniqlik va samaradorlik darajasi ancha past. Shuning uchun ma’noli so‘zlarni kiritish, modelga ko‘proq kontekst va semantika olish uchun katta hajmga ega korpusdan foydalanish kerak. Korpus asosida shakllantirilgan Word2Vec modelida **“kun”** so‘zi uchun quyidagi zich vector hosil qilingan:

```
w2v_model.wv['kun']
```

```
array([-0.00652111,  0.00892327, -0.00158468, -
0.00828349,  0.00801596,
       -0.00296089, -0.00133201, -0.00067581,
0.00282682, -0.00471678,
       -0.00025782, -0.00284552,  0.00877594,  0.0068789
,  0.00109954,
       -0.00606629,  0.00313261,  0.00113649, -
0.00612438, -0.00575232,
       -0.00309658,  0.0023865 , -0.00811502, -
0.00499184,  0.00809853,
       -0.00369231,  0.00795793,  0.00053581, -
0.00553437, -0.0049641 ,
       -0.00021518,  0.00459268, -0.00608348, -
0.0055082 , -0.00681099,
        0.00674646, -0.00240377,  0.00401605, -
0.00162639, -0.00741547,
       -0.00126834,  0.0004031 ,  0.00358639, -
0.00867601, -0.00680536,
```

```
       -0.00655691,  0.00299229, -0.00374216,
0.00101714,  0.00415705,
        0.00451883,  0.00236562, -0.00287308,
0.00064065, -0.00539167,
       -0.00635273,  0.00871573, -0.00405068, -0.009318
,  0.00373121,
        0.00845417, -0.00018828,  0.00226474, -0.0035472
, -0.0041924 ,
       -0.00861984, -0.007853  , -0.00907923, -0.0059706
,  0.0084879 ,
        0.00943666, -0.00167923,  0.00001528,
0.00785286,  0.00126256,
       -0.00514527,  0.00262805,  0.00180579, -
0.00372638, -0.00489015,
        0.00964372,  0.00764874, -0.00857014, -
0.00552514,  0.00142141,
        0.0009151 , -0.008351  ,  0.00932817, -0.0053818 ,
0.00460131,
        0.00923111,  0.00109029,  0.00422058, -0.0027155
, -0.00968235,
        0.00059987, -0.00108389, -0.00992071,
0.00700562, -0.00738599],
      dtype=float32)
```

Keyingi qadamda namunaviy korpusdagi sakkizta hujjatni turli guruharga ajratish masalasini ko‘rib chiqamiz. Buning uchun korpusdagi har bir hujjatda mavjud bo‘lgan har bir so‘zdan hujjat darajasidagi joylashuvlarni aniqlash talab etiladi. Hujjatdagi har bir so‘z uchun so‘zlarni joylashtirishning o‘rtacha qiymatini hisoblash orqali masalasini hal qilish mumkin. Har bir hujjatning xususiyatlarini aniqlash uchun namunaviy korpusda quyidagi dasturiy kodni qo‘llaymiz.

```
def average_word_vectors(words, model, vocabulary,
num_features):
    feature_vector =
np.zeros((num_features,),dtype="float64")
    nwords = 0.
    for word in words:
        if word in vocabulary:
            nwords = nwords + 1.
            feature_vector = np.add(feature_vector,
model.wv.get_index(word))
    if nwords:
        feature_vector = np.divide(feature_vector,
nwords)
    return feature_vector

def averaged_word_vectorizer(corpus, model,
num_features):
    vocabulary = set(model.wv.index_to_key)
    features =
[average_word_vectors(tokenized_sentence, model,
vocabulary, num_features)
                for tokenized_sentence in corpus]
    return np.array(features)

# hujjat darajasidagi o‘rnatishlarni aniqlash
w2v_feature_array =
averaged_word_vectorizer(corpus=tokenized_corpus,
model=w2v_model, num_features=vector_size)
pd.DataFrame(w2v_feature_array)
```

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96	97	98	99
0	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000	...	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000	19.000000
1	11.250000	11.250000	11.250000	11.250000	11.250000	11.250000	11.250000	11.250000	11.250000	11.250000	...	11.250000	11.250000	11.250000	11.250000	11.250000	11.250000	11.250000	11.250000	11.250000	11.250000
2	8.375000	8.375000	8.375000	8.375000	8.375000	8.375000	8.375000	8.375000	8.375000	8.375000	...	8.375000	8.375000	8.375000	8.375000	8.375000	8.375000	8.375000	8.375000	8.375000	8.375000
3	14.166667	14.166667	14.166667	14.166667	14.166667	14.166667	14.166667	14.166667	14.166667	14.166667	...	14.166667	14.166667	14.166667	14.166667	14.166667	14.166667	14.166667	14.166667	14.166667	14.166667
4	32.285714	32.285714	32.285714	32.285714	32.285714	32.285714	32.285714	32.285714	32.285714	32.285714	...	32.285714	32.285714	32.285714	32.285714	32.285714	32.285714	32.285714	32.285714	32.285714	32.285714
5	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	...	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000
6	21.250000	21.250000	21.250000	21.250000	21.250000	21.250000	21.250000	21.250000	21.250000	21.250000	...	21.250000	21.250000	21.250000	21.250000	21.250000	21.250000	21.250000	21.250000	21.250000	21.250000
7	19.800000	19.800000	19.800000	19.800000	19.800000	19.800000	19.800000	19.800000	19.800000	19.800000	...	19.800000	19.800000	19.800000	19.800000	19.800000	19.800000	19.800000	19.800000	19.800000	19.800000

8 rows × 100 columns

Har bir hujjatga xos xususiyatlar aniqlanganidan so‘ng ushbu hujjatlarni **Affinity Propagation**[18] algoritmidan foydalangan holda klasterlash mumkin. Affinity Propagation – ma’lumotlar nuqtalari o‘rtasida *“xabar uzatish” (message passing)* konsepsiyasiga asoslangan klasterlash algoritmi hisoblanadi.

```
corpus = ['Kun bulutli bo‘lsa, yomg‘ir yog‘ishi mumkin',
'Ertaga havo bulutli bo‘ladi hamda yomg‘ir yog‘adi',
'Toshkent hayvonot bog‘iga kichkina tog‘ echkisi olib
kelindi',
'Uch dona tuxumni idishda qaynatib oldi',
'Sumalak, ko‘k somsa, palov kabi bayram taomlari
ulashildi',
'Parranda va quyon go‘shtidan taom tayyorlandi',
'Darvoza oldida somsa sotilar ekan',
'Tuyaqush boqish ham barakali biznesga aylandi'
]
labels = ['ob-havo‘, 'ob-havo‘, 'hayvonot', 'oziq-ovqat',
'oziq-ovqat', 'oziq-ovqat', 'oziq-ovqat', 'hayvonot']
corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus,
                          'Category': labels})
corpus_df = corpus_df[['Document', 'Category']]
corpus_df
```

№	Document	Category
0	Kun bulutli bo‘lsa, yomg‘ir yog‘ishi mumkin	ob-havo

[18] https://en.wikipedia.org/wiki/Affinity_propagation

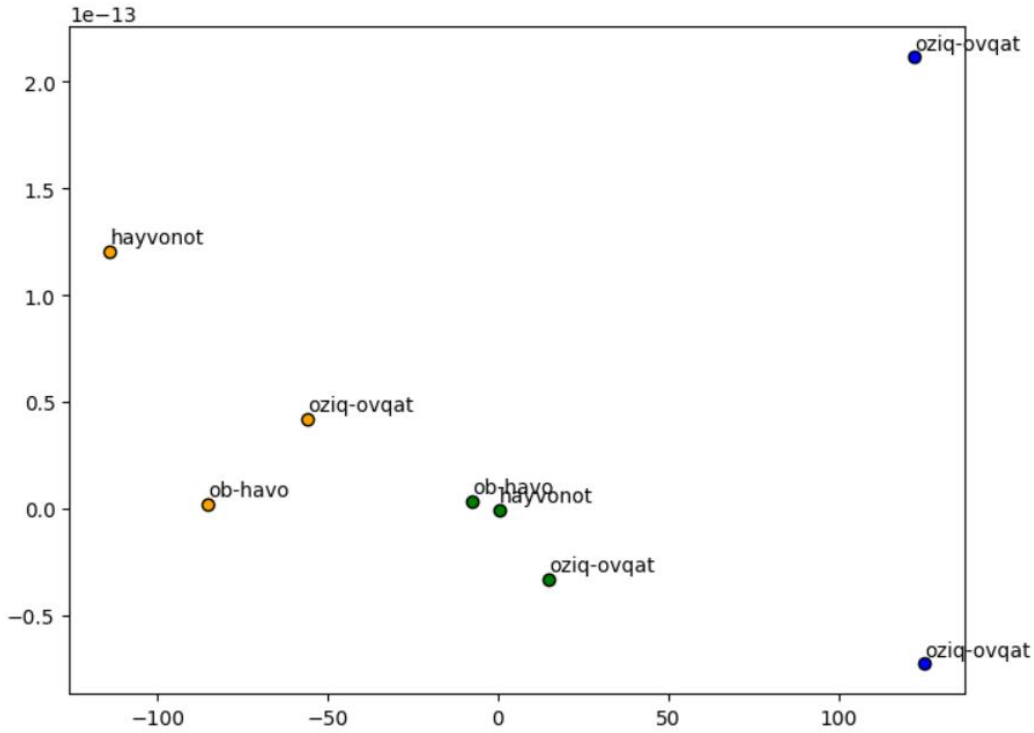
1	Ertaga havo bulutli bo‘ladi hamda yomg‘ir yog‘adi	ob-havo
2	Toshkent hayvonot bog‘iga kichkina tog‘ echkisi olib kelindi	hayvonot
3	Uch dona tuxumni idishda qaynatib oldi	oziq-ovqat
4	Sumalak, ko‘k somsa, palov kabi bayram taomlari ulashildi	oziq-ovqat
5	Parranda va quyon go‘shtidan taom tayyorlandi	oziq-ovqat
6	Darvoza oldida somsa sotilar ekan	oziq-ovqat
7	Tuyaqush boqish ham barakali biznesga aylandi	hayvonot

```
from sklearn.cluster import AffinityPropagation
ap = AffinityPropagation()
ap.fit(w2v_feature_array)
cluster_labels = ap.labels_
cluster_labels = pd.DataFrame(cluster_labels,
columns=['ClusterLabel'])
pd.concat([corpus_df, cluster_labels], axis=1)
```

№	Document	Category	Cluster-Label
0	Kun bulutli bo‘lsa, yomg‘ir yog‘ishi mumkin	ob-havo	2
1	Ertaga havo bulutli bo‘ladi hamda yomg‘ir yog‘adi	ob-havo	0
2	Toshkent hayvonot bog‘iga kichkina tog‘ echkisi olib kelindi	hayvonot	0
3	Uch dona tuxumni idishda qaynatib oldi	oziq-ovqat	0
4	Sumalak, ko‘k somsa, palov kabi bayram taomlari ulashildi	oziq-ovqat	1
5	Parranda va quyon go‘shtidan taom tayyorlandi	oziq-ovqat	1
6	Darvoza oldida somsa sotilar ekan	oziq-ovqat	2
7	Tuyaqush boqish ham barakali biznesga aylandi	hayvonot	2

Bizning algoritmimiz Word2Vec xususiyatlaridan kelib chiqqan holda har bir hujjatni to‘g‘ri guruhga ajratganini ko‘rish mumkin. Shuningdek, biz har bir hujjatning har bir klasterda qanday joylashishini asosiy komponentlar tahlili (Principal Component Analysis, PCA)dan foydalanib, xususiyat o‘lchamlarini 2 o‘lchamligacha qisqartirishimiz va keyin xuddi shunday vizualizatsiya qilishimiz mumkin.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2, random_state=0)
pcs = pca.fit_transform(w2v_feature_array)
labels = ap.labels_
categories = list(corpus_df['Category'])
plt.figure(figsize=(8, 6))
#print(pcs)
for i in range(len(labels)):
    label = labels[i]
    color = 'orange' if label == 0 else 'blue' if label == 1
else 'green'
    annotation_label = str(categories[i])
    x, y = pcs[i]
    #print('x',x,x+1e-4)
    #print('y',y,y+1e-3)
    plt.scatter(x, y, c=color, edgecolors='k')
    plt.annotate(annotation_label, xy=(x, y), xytext=(0, 4),
textcoords='offset points')
    #plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0),
textcoords='offset points')
```



15-rasm. *Korpus hujjatlarini klasterlashni vizualizatsiyalash*

Yuqoridagi jadval va 15-rasmdan har bir klasterdagi hujjatlar bir-biriga yaqinroq; boshqa klasterlardan uzoqroq ekanligini aniqlash mumkin.

Aksariyat hollarda Word2Vec metodi vositasida so‘zlar joylashuvini shakllantirishda xususiyat vektorlarining uzunligini sozlash imkonini beradi va *(target,context)* so‘z juftliklari yordamida tokenlar o‘rtasidagi munosabatlarni saqlaydi. Biroq Word2Vec metodida ba’zi cheklovlar mavjud. Ularni quyida qisqacha tavsiflaymiz:

– *Word2Vec metodi so‘zlar haqidagi lokal ma’lumotlarga tayanadi, ya’ni so‘zning konteksti faqat qo‘shnilariga tayanadi.*

– *Metod vositasida shakllantirilgan so‘zlar joylashtirish neyron tarmoqni o‘qitishning qo‘shimcha mahsulotidir. Shuning uchun xususiyat vektorlari orasidagi chiziqli munosabatlar no‘malum.*

– *Word2Vec metodi lug‘atdan tashqari (out-of-vocabulary, OOV) so‘zlarni, ya’ni o‘quv ma’lumotlari(korpus)da mavjud bo‘lmagan so‘zlarni tushuna olmaydi. OOV so‘zlar uchun ishlatiladigan UNK tokenini belgilash yoki OOV so‘zlariga mustahkam bo‘lgan boshqa modellardan foydalanish mumkin.*

– *Har bir so‘zga alohida vektor belgilash orqali Word2Vec metodi so‘zlarning morfologiyasini e’tiborsiz qoldiradi. Misol uchun,* ***yeyish, yegan*** *va* ***yeyildi*** *Word2Vec metodi tomonidan mustaqil ravishda har xil (turli) so‘zlar deb hisoblanadi. Ammo ular bir asos* ***“yemoq”*** *(o‘zagi)dan kelib chiqqan.*

Ushbu cheklagichlarni bartaraf qila oladigan o‘rnatish modellarini taqdim etamiz: **GloVe** va **FastText**. Keyingi qadamlarda korpus matnlari yaxlit (bitta) hujjat sifatida qayta ishlanadi va hujjatlar to‘plami korpus bo‘yicha yuritiladi.

NLP vazifalarini hal qilishda oldindan tayyorlangan Word2Vec modellaridan ham foydalanish mumkin. Ushbu turdagi Word2Vec modeliga uch million so‘z va so‘z birikmalari uchun **300** o‘lchovli sonli vektorli model Google News ma’lumotlar to‘plamining bir qismida (taxminan 100 milliard so‘z) o‘qitilgan bo‘lib, uni quyidagi manzildan yuklab olish mumkin: https://code.google.com/archive/p/word2vec/

GloVe modeli

Word2Vec metodidan farqli o‘laroq, GloVe (Global Vectors) metodi to‘g‘ridan to‘g‘ri xususiyat vektorlarining chiziqli tuzilmalarni topish uchun yaratilgan. Bunda GloVe metodi to‘g‘ridan to‘g‘ri global korpus statistikasini oladi va bu ma’lumotlardagi takrorlanishning katta miqdoridan foydalanadi. Natijada yuqoridagi Word2Vec metodida ko‘rsatib o‘tilgan 1- va 2-cheklovlari hal qilinadi.

GloVe modeli global vektorlarni anglatib, nazoratsiz o‘rganish modeli hisoblanadi. GloVe modeli til korpusidan Word2Vec modeliga o‘xshash zich so‘z vektorlarini olish uchun ishlatilishi mumkin. Biroq GloVe modelidagi mashg‘ulot jarayonida so‘zlarning birgalikda paydo bo‘lish matritsasi global miqyosida amalga oshiriladi. Bu esa bizga mazmunli kichik tuzilmalarga ega vektor maydonini beradi. GloVe modeli Stenfordda Pennington va boshqalar tomonidan ixtiro qilingan[19] bo‘lib, "GloVe: Global Vectors for Word Representation" nomli maqolada ushbu usulning ishlashi haqida batafsil ma’lumot berilgan [47].

GloVe modeli *"o‘xshash kontekstlarda uchraydigan so‘zlar o‘xshash ma’noga ega bo‘lishi mumkin"* gipotezasiga asoslanadi. GloVe korpus matnlarini Word2Vec metodiga qaraganda biroz boshqacha o‘rganadi va so‘zlarning vektorlarini ularning birgalikda paydo bo‘lish statistikasidan foydalangan holda shakllantiradi.

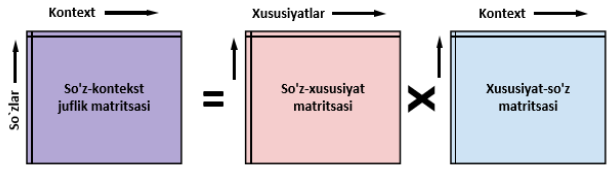
Word2Vec va GloVe o‘rtasidagi asosiy farqlardan biri shundaki, Word2Vec bashoratli xususiyatga ega. Masalan, Skip-gram usuli so‘z vektor ko‘rinishlari asosida kontekstdagi so‘zlardan to‘g‘ri maqsadli so‘zni "bashorat" qilishga harakat qiladi. Biroq GloVe metodi hisob-kitoblarga asoslangan tarzda sonli vektorlarni shakllantiradi. Ushbu maqoldada GloVe modelining asosiy tamoyillari keltiriladi.

Maqolaning boshlang‘ich qismida LSA kabi matritsalarni faktorizatsiyalash usullari va Word2Vec kabi bashoratli usullar haqida ma’lumot berilgan edi. Soha mutaxassislarining fikriga ko‘ra, ikkala turdagi metodda ham jiddiy kamchiliklar mavjud. LSA kabi usullar statistik ma’lumotlardan samarali foydalanadi, ammo ular so‘z o‘xshashligi vazifasini nisbatan yomon bajaradi. Masalan, semantik jihatdan o‘xshash so‘zlarni qanday topiladi? Skip-gram kabi usullar o‘xshashlik vazifasini yaxshiroq bajarishi mumkin, ammo ular global darajada korpus statistikasidan yaxshi foydalanmaydi.

GloVe modelining ish prınsıpını tushunish uchun GloVe asosida yaratilgan ikki asosiy usul – *global matritsalarni faktorizatsiyalash (global matrix factorization, GMF)* va *lokal kontekst oynasi (local context window)ni* tushunishimiz kerak.

NLPda **global matritsa faktorizatsiyasi** – katta hajmga ega chastotali matritsalar o‘lchovini kamaytirish uchun chiziqli algebradan matritsalarni faktorizatsiya qilish usullaridan foydalanish jarayonidir. Ushbu matritsalar odatda hujjatda so‘zlarning paydo bo‘lishi yoki mavjud emasligini ifodalaydi. GMFlarni terminlarning chastotali matritsalariga qo‘llash NLPda yashirin semantik tahlil (Latent Semantic Analysis, LSA) deb ataladi. **Lokal kontekst oynasi** – CBOW va Skip-Gram usullari bo‘lib, Skip-gram kichik hajmdagi o‘quv ma’lumotlari bilan yaxshi ishlaydi va hatto kamdan kam uchraydigan so‘zlarni ham ifodalaydi. CBOW usuli esa bir necha baravar tezroq ishlab, korpusdagi baland chastotali so‘zlarni yaxshiroq aniqlaydi.

GloVe modelining asosiy metodologiyasi birinchi navbatda (so‘z, kontekst) juftliklaridan tashkil topgan ulkan so‘z-kontekstning birgalikdagi matritsasini yaratishdan iborat. Bu matritsadagi har bir element so‘zning kontekstda qanchalik ko‘p uchrashini ifodalaydi. GloVe modelining umumiy tavsifi quyidagi 16-rasmda ko‘rsatilganidek, ushbu matritsani faktorizatsiya qilishdan iborat.



16-rasm. *GloVe modelini amalga oshirishning kontseptual modeli*

So‘z-kontekst (Word-Context, WC) juflik matritsasi, **so‘z-xususiyat (Word-Feature, WF)** matritsasi va **xususiyat-so‘z (Feature-Context,**

[19] https://nlp.stanford.edu/pubs/glove.pdf

FC) matritsalarini hisobga olgan holda, **WC = WF x FC** matritsasini faktorlarga ajratishga harakat qilamiz. Asosiy maqsad **WF** va **FC** matritsalarni ko'paytirish orqali **WC** matritsani qayta tiklashdan iborat. Buning uchun **WF** va **FC** matritsalarni ba'zi tasodifiy og'irliklar bilan initsalizatsiyalanadi, ularni ko'paytirib **WC`** matritsa aniqlanadi hamda uning **WC** matritsaga qanchalik yaqinligi o'lchanadi. Xatolikni kamaytirish uchun **Stochastic Gradient Descent (SGD)** yordamida bir necha marta amalga oshiramiz. Va nihoyat, **WF** matritsasi bizga har bir so'z uchun so'zlarni joylashtirishni imkonini taqdim etadi. Bu yerda **F** miqdor oldindan o'rnatilishi mumkin. Word2Vec va GloVe modellarining ishlash tamoyili juda o'xshash bo'lib, ularning ikkalasi ham maqsadi vektor fazosini qurishdan iborat. Vector fazosidagi har bir so'zning o'rni kontekst va semantikaga asoslangan qo'shni so'zlar orqali shakllantiriladi. Word2Vec metodi so'z birikmalarining lokal konteksti asosida, GloVe mteodi korpusdagi barcha so'zlar bo'yicha global kontekstda paydo bo'lish statistikasi orqali sonli vektorlarni hosil qiladi.

Mashinali o'rganish vazifalari uchun GloVe xususiyatlarini qo'llash

Hujjatlarni klasterlash NLP vazifasi uchun GloVe metodiga asoslangan o'rnatishlardan foydalanishga harakat qilamiz.

Python tilidagi **Space** paketidagi funksiyalar vositasida turli til modellari asosida GloVe sonli vektorlarni shakllantirish mumkin. Shuningdek, usbu paket orqali oldindan o'rgatilgan so'z vektorlarini[20] olish va kerak bo'lganda ularni **gensim** yoki **spacy** yordamida yuklash mumkin.

Space modellarini o'rnatishning[21] turli avtomatlashtirilgan usullari mavjud. Birinchi navbatda paketdagi zarur til va unga mos vektorlarni yuklab olish lozim.

```
config_cooccurrence_dir="E:\\nlp\\glove\data"
config_vocab_size=23700
config_embedding_size=100
config_x_max=100
config_alpha=0.75
config_model_filepath="E:\\nlp\\glove\\data\model.pt"
with open(os.path.join(config_cooccurrence_dir,
"vocab.pkl"), "rb") as f:
   vocab = pickle.load(f)

model = GloVe(
      vocab_size=config_vocab_size,
      embedding_size=config_embedding_size,
      x_max=config_x_max,
      alpha=config_alpha
   )
model.load_state_dict(torch.load(config_model_filepat
h))

wordList=[vocab.get_token(index) for index in
range(config_vocab_size)]
vectorList=(model.weight.weight.detach() +
model.weight_tilde.weight.detach()).numpy()

keyed_vectors =
KeyedVectors(vector_size=config_embedding_size)
keyed_vectors.add_vectors(
      keys=wordList,
      weights=vectorList
   )
total_vectors = len(vectorList)
print('Total word vectors:', total_vectors)
```

Total word vectors: 23700

Yuqoridagi dastur kodi orqali ingliz tilidagi 23700 ta sonli vektor xotiraga yuklab olinadi. Keyingi qadamda, namunaviy korpusning har bir so'zi uchun GloVe o'xshashliklar shakllantiriladi.

```
unique_words = list(set([word for sublist in [doc.split()
for doc in norm_corpus_text] for word in sublist]))
word_glove_vectors = np.array([keyed_vectors[word]
for word in unique_words])
pd.DataFrame(word_glove_vectors,
index=unique_words)
```

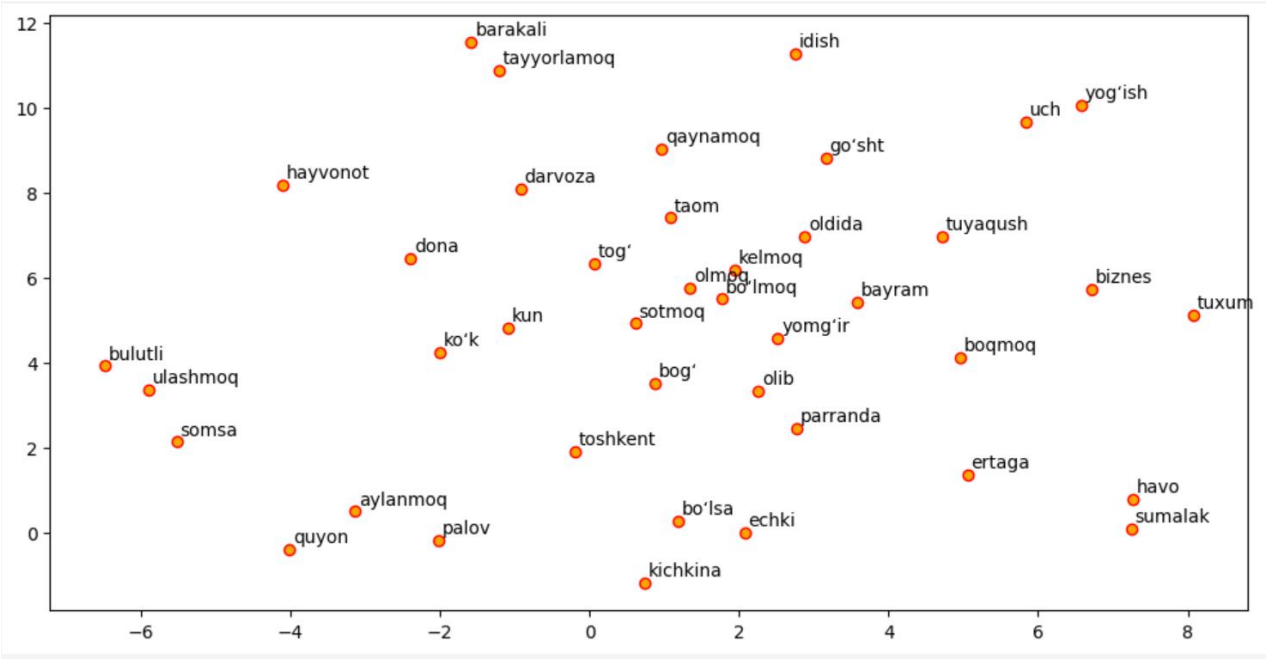
[20] https://nlp.stanford.edu/projects/glove/

[21] https://spacy.io/usage/models

	0	1	2	3	4	5	6	7	8	9	...	90	91	92
toshkent	0.090396	-0.686545	0.073527	0.496121	-0.213236	0.433611	-0.048264	0.534554	-0.144608	0.143409	...	0.609385	0.009055	-0.428743
tuyaqush	1.525145	-1.143674	-1.195047	-0.397236	-0.613176	-0.941239	0.009479	0.242290	0.019543	-0.779340	...	-0.604940	0.848494	-0.873229
idish	-0.881208	-0.767945	0.944807	1.363350	-1.102733	0.727654	-0.628996	0.945548	0.536121	0.033776	...	0.781085	-0.604311	0.115275
barakali	-0.063739	0.024553	-0.746210	0.245452	0.464121	-0.267621	0.477328	-1.022480	-0.595336	-1.582388	...	-1.465868	0.389009	-0.406575
bo‘lmoq	0.332895	-0.206622	0.031110	0.678350	-0.270524	0.464874	-0.041595	0.090912	0.604319	-0.244968	...	0.138015	-0.260565	0.086660
tuxum	-0.483026	-0.565688	-0.231970	-0.702243	0.629288	-0.535467	-0.404881	0.592932	-0.521123	0.147060	...	1.736139	-0.680600	-1.005477
uch	0.730118	-0.955107	0.669986	0.857212	0.018876	0.825921	0.074078	0.722744	0.173598	-0.075602	...	0.021261	-0.374457	-0.683637
havo	0.821705	-0.203711	0.137931	0.552784	-0.982719	0.523452	0.574903	0.102734	-0.101222	-0.352757	...	0.556759	-0.173185	1.227091
somsa	-0.029290	-0.425786	-0.103042	-0.645338	0.758055	0.059107	0.479908	0.082295	-0.875270	1.215963	...	-0.545617	-0.203530	1.875015
palov	-0.328134	0.133118	-0.879259	0.760600	-1.279392	0.424542	0.977280	0.488457	-1.040472	0.476391	...	1.227506	1.128089	-0.475977
qaynamoq	1.647881	1.050931	0.569563	-0.336356	0.052042	-0.061109	2.240242	1.384194	0.261380	-0.647138	...	-0.038296	-0.308016	-0.394205
quyon	-0.524620	1.087090	0.020797	-0.208130	-1.418160	-1.986593	0.550177	-0.278797	1.667666	0.446119	...	-0.224875	-0.751312	-0.310744
tog‘	0.520344	-0.661772	-0.691606	1.030620	0.511221	1.772027	0.072208	-0.545186	-0.128469	0.070731	...	-0.645817	0.148724	0.504427
ko‘k	1.023090	0.114466	0.759049	-0.606611	-0.468410	1.339189	1.247785	-1.551438	0.645205	1.434168	...	-0.118165	-0.400158	0.652149
bulutli	-0.239736	1.723938	0.276993	-0.406639	-0.821774	-1.424534	1.347438	1.872456	1.300686	1.097851	...	-0.857515	-0.122131	-1.542512
darvoza	-0.486058	-0.649757	-0.151608	-0.071623	-0.450085	-0.813769	0.496410	0.385309	0.092261	-0.154452	...	-1.490163	-0.731220	0.704747
tayyorlamoq	-0.583116	0.328385	-0.055053	0.809958	-0.714259	-0.019316	0.169172	0.101261	0.436170	-0.649349	...	0.004393	-0.489689	0.625224
go‘sht	0.266041	-0.368711	0.318641	0.378727	0.425442	-0.390369	0.336482	-0.262518	0.384608	0.883596	...	0.131382	-0.406357	-0.341179
kun	0.431981	-0.627036	0.371524	-0.006661	0.016069	0.510295	-0.496849	-0.205241	0.014473	-0.536997	...	0.006822	-0.288580	-0.300162

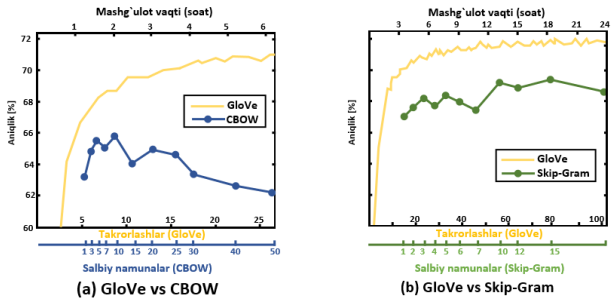
Yuqoridagi dastur kodi orqali berilgan til korpusidagi vektorlarni t-SNE texnologiyasi yordamida vizualizatsiya qilish mumkin

```
from sklearn.manifold import TSNE
import numpy as np
import matplotlib.pyplot as plt
tsne = TSNE(n_components=2, random_state=0,
n_iter=5000, perplexity=3)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(word_glove_vectors)
labels = unique_words

plt.figure(figsize=(12, 6))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x, y), xytext=(2, 4),
textcoords='offset points')
```



Quyidagi 17-rasmda keltirilganidek, GloVe modelining ko‘pgina ssenariylarda Word2Vec modelidan yaxshiroq natija qaytarishi va samaradorligi yuqori ekanligini qayd etish o‘rinli[22].

[22] https://nlp.stanford.edu/pubs/glove.pdf



17-rasm. *GloVe va Word2Vec metodlar natijasini solishirish*

Yuqoridagi tajribalar **300** o'lchovli sonli vektorlarni **400000** so'zli lug'atdan iborat **6B**[23] korpusida **10** o'lchamdagi oynaga ega simmetrik kontekst oynasi bilan o'qitish orqali amalga oshirildi.

FastText modeli

FastText modeli[24] birinchi marta Facebook kompaniyasi tomonidan 2016 yilda Word2Vec modelining kengaytirilgan, takomillashtirilgan varianti sifatida taqdim etilgan. Mikolov va boshqalarning *"Enriching Word Vectors with Subword Information"* nomli maqolasida[25] FastText modelining ishlash tamoyili keltirilgan [48].

Umuman olganda, FastText so'z ko'rinishlarini o'rganish, shuningdek, matn tasnifini *samarali, tez* va *aniq* amalga oshirish platformasi hisoblanadi. Facebook kompaniyasining GitHub platformasidagi profilida[26] FastText modeli haqidagi quyidagi ochiq ma'lumotlar joylashtirilgan:

- *eng so'nggi zamonaviy inglizcha so'z vektorlari*[27]*;*
- *vikipediya va Crawl tizimlaridagi 157 ta til uchun o'rgatilgan so'z vektorlari*[28]*;*
- *til identifikatsiyasi*[29] *va turli nazorat qilinadigan vazifalar*[30] *uchun modellar.*

Ushbu maqolada FastText modelining to'liq tavsifi keltirilmaydi. Biroq mavjud tadqiqotlarga asoslanib, modelning qanday ishlashi, uning o'zbek tili korpusi matnlariga qo'llanishi ko'rib chiqiladi. Umuman olganda, Word2Vec modeli kabi bashoratli modellar, odatda, har bir so'zni alohida obyekt sifatida ko'rib chiqadi va so'z uchun zich joylashuv vektorlarini shakllantiradi. Biroq ushbu modellarda *katta lug'atga ega bo'lgan tillar* va *turli korpuslarda ko'p uchramaydigan ko'plab noyob so'zlar* bilan bog'liq jiddiy cheklovlar mavjud. Word2Vec modeli har bir so'zning morfologik tuzilishini e'tiborsiz qoldiradi va *so'zni yagona element sifatida* ko'rib chiqadi. FastText modeli har bir so'zni **n-gramm belgilar ketma-ketligi (Bag of Character n-grams)** sifatida ko'rib chiqadi. Ba'zi ilmiy tadqiqotlarda ushbu usul **so'zosti modeli** deb ataladi.

FastText usulida so'zning boshi va oxiriga **"<"** va **">"** kabi maxsus chegara belgilari qo'shiladi. Bu amal bizga so'zdagi boshqa belgilar ketma-ketligidan *prefiks* va *suffikslarni* ajratish imkonini beradi. Shuningdek, har bir so'zni sonli tarzda ifodalash uchun (uning n-gramm belgilaridan tashqari) **w** so'zining o'zini ham **n-grammlar** to'plamiga kiritamiz.

<qayerda> so'zi va **n=3** (trigramm) bo'lgan hol uchun, quyidagi n-grammlar to'plami hosil qilinadi: **<qa**, **qay**, **aye**, **yer**, **erd, rda, da>** va butun so'zni ifodalovchi **<qayerda>** maxsus ketma-ketligi. E'tibor bering, **<yer>** so'ziga mos keladigan ketma-ketlik **<qayerda>** so'zining trigrammasidan farq qiladi.

Amalda, $\mathbf{n \geq 3}$ va $\mathbf{n \leq 6}$ bo'lgan hol uchun barcha n-grammlarni ajratib olish tavsiya qilinadi. Bu juda oddiy yondashuv hisoblanib, barcha prefiks va suffikslarni olish orqali n-grammlarning turli to'plamlarini ko'rib chiqish mumkin. So'zning vektor tasvirini (o'rnatish) uchun uning n-grammlari ishlab chiqiladi va berilgan so'z bilan mos bog'lanadi. Shunday qilib, so'zni uning n-grammlarining vektor ko'rinishlarining yig'indisi (birlashmasi) yoki ushbu n-grammlarning o'rtacha kiritilishi bilan ifodalash mumkin. N-grammlarni o'z belgilariga qarab so'zlardan foydalanishning

[23] Vikipediya 2022 + Gigaword 5
[24] https://fasttext.cc/
[25] https://arxiv.org/pdf/1607.04606.pdf
[26] https://github.com/facebookresearch/fastText
[27] https://fasttext.cc/docs/en/english-vectors.html
[28] https://github.com/facebookresearch/fastText/blob/master/docs/crawl-vectors.md
[29] https://fasttext.cc/docs/en/language-identification.html#content
[30] https://fasttext.cc/docs/en/supervised-models.html#content

bunday yondashuvi tufayli, noyob (unikal) so‘zlarning yaxshilangan ko‘rinishga ega bo‘lish ehtimoli yuqori. Chunki so‘zlarning ba’zi n-grammlari korpusning boshqa so‘zlarida mavjud bo‘lishi mumkin.

Mashinali o‘rganish vazifalarida FastText xususiyatlarini qo‘llash

Gensim paketidagi **gensim.models.fasttext** moduli orqali FastText modelidan foydalanish masalasini ko‘rib chiqamiz. Buning uchun ushbu modilni o‘zbek tili korpusi matnlariga qo‘llaymiz va zarur so‘zlarimizga o‘xshash so‘zlari to‘plamini shakllantiramiz.

```
uz_stop_words = set([w.strip() for w in
open("d:\\nlp\\uz_stop_words.txt",encoding="utf8").read
lines()])
#print(uz_stop_words)
def filter_stop_words(train_sentences, stop_words):
    for i, sentence in enumerate(train_sentences):
        #print(i,sentence)
        #print(i,sentence.split())
        new_sent = [word for word in sentence.split() if
word not in stop_words]
        #print(new_sent)
        train_sentences[i] = ' '.join(new_sent)
    return train_sentences
from nltk.corpus import gutenberg
from string import punctuation
corpus_text = open("d:\\nlp\\misol_text.txt",
encoding="utf8").readlines()
remove_terms = punctuation + '0123456789'
norm_corpus_text = [[word.lower() for word in sent if
word not in remove_terms] for sent in corpus_text]
norm_corpus_text = [''.join(tok_sent) for tok_sent in
norm_corpus_text]
norm_corpus_text = [tok_sent.strip() for tok_sent in
norm_corpus_text if len(tok_sent.split()) > 2]
norm_corpus_text =
filter_stop_words(norm_corpus_text, uz_stop_words)
print('Total lines:', len(corpus_text))
print('\nSample line:', corpus_text[1])
print('\nProcessed line:', norm_corpus_text[1])
tokenized_corpus = [document.split(' ') for document in
norm_corpus_text]
print(tokenized_corpus)
```

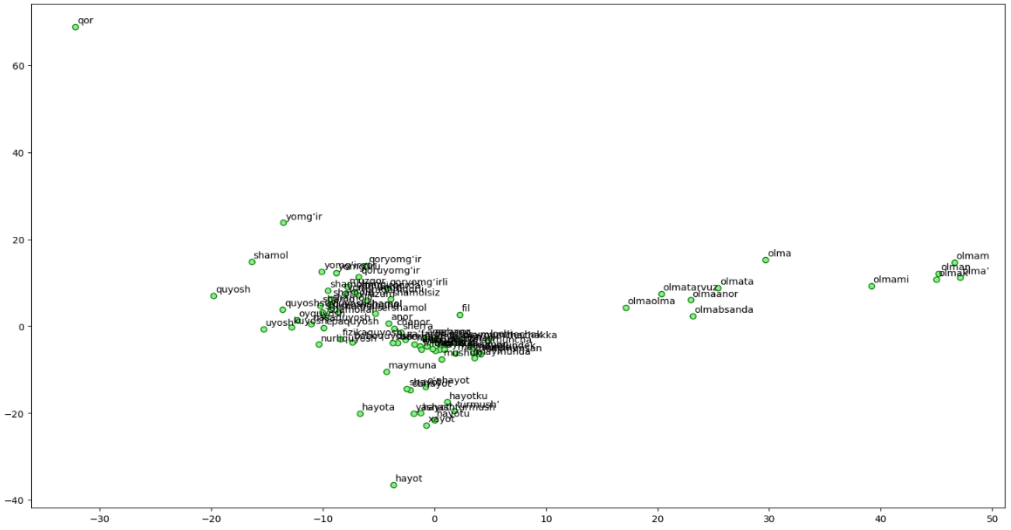
```
[['kun', 'bulutli', 'bo‘lsa’, 'yomg‘ir', 'yog‘ishi'], ['ertaga’,
'havo‘, 'bulutli', 'bo‘ladi'], ['toshkent', 'hayvonot', 'bog‘iga’,
'kichkina’, 'tog‘', 'echkisi', 'olib', 'kelindi'], ['uch', 'dona’,
'tuxumni', 'idishda’, 'qaynatib', 'oldi'], ['sumalak', 'ko‘k',
'somsa’, 'palov', 'bayram', 'taomlari', 'ulashildi'], ['parranda’,
'quyon', 'go‘shtidan', 'taom', 'tayyorlandi'], ['darvoza’,
'oldida’, 'somsa’, 'sotilar'], ['tuyaqush', 'boqish', 'barakali',
'biznesga’, 'aylandi']]
from gensim.models.fasttext import FastText
# Turli parametrlar uchun qiymatlarni o‘rnatish
feature_size = 100    # So‘z vektor o‘lchovliligi
window_context = 50   # Kontekst oynasining
o‘lchami
min_word_count = 1    # So‘zlarning minimal
soni
ft_model = FastText(vector_size=feature_size,
window=window_context,
min_count=min_word_count)  # o‘rnatish
ft_model.build_vocab(corpus_iterable=tokenized_corpus
)
ft_model.train(corpus_iterable=tokenized_corpus,
total_examples=len(tokenized_corpus), epochs=20)  #
mashg‘ulot
from gensim.test.utils import get_tmpfile
fname =
get_tmpfile("d:\\nlp\\fasttextmodel03\\fasttext.model")
ft_model.save(fname)
similar_words = {search_term: [item[0] for item in
ft_model.wv.most_similar([search_term], topn=10)]
                 for search_term in ['anor', 'maymun', 'olma',
'qor', 'quyosh', 'sher', 'shamol','hayot']}
similar_words
```

```
'anor': ['anoranor', 'anorli', 'hanor', 'uzum', 'anoru',
'chanor', 'anorday', 'anori', 'anorchi', 'za’faron'],
'maymun': ['maymunchechak', 'maymuncha',
'maymunsan', 'maymuna', 'maymundek', 'maymunda',
'maymunchechakka', 'mushuk', 'maymunday',
'maymunsimon'], 'olma': ['olmaolma', 'olmatarvuz',
'olmaanor', 'olman', 'olmak',
...
'hayot', 'o‘nhayot', 'yashashturmush', 'turmush’', 'shayot']
```

FastText metodi orqali shakllantirilgan yuqoridagi jadvaldagi har bir so‘zga mos o‘xshash so‘zlar bilan Word2Vec modeli orqali hosil qilingan natijalarda juda ko‘p o‘xshashlikni ko‘rish mumkin. Principal Component Analysis (PCA) usulidan foydalanib, 2-D formatdagi so‘zlar o‘xshashligining vizual shaklini hosil qilamiz.

```
from sklearn.decomposition import PCA
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
pd.options.display.max_colwidth = 200
%matplotlib inline
words = sum([[k] + v for k, v in similar_words.items()],
[])
wvs = ft_model.wv[words]

pca = PCA(n_components=2)
np.set_printoptions(suppress=True)
P = pca.fit_transform(wvs)
```

```
labels = words

plt.figure(figsize=(18, 10))
plt.scatter(P[:, 0], P[:, 1], c='lightgreen', edgecolors='g')
for label, x, y in zip(labels, P[:, 0], P[:, 1]):
    plt.annotate(label, xy=(x, y), xytext=(2, 4),
textcoords='offset points')
```



FastText metodi orqali hosil qilingan har qanday so‘z quyidagi shakldagi sonli qiymatlardan iborat vektor tarzda indekslashtirilgan.

```
ft_model.wv['taom']
```

```
array([ -4.8098307 ,  0.08534396, -7.785515  ,  8.860114  ,
 -1.548883  , 13.094814  ,  4.470709  , -1.2608742 ,
 -0.58699304, -2.675268  ,  3.530713  , -3.09222   ,
  7.9175706 ,  2.7406228 ,  0.81931686, -1.1441586 ,
 -1.9119034 ,  9.282662  , -4.7862544 ,  4.456146  ,
 -9.911456  ,  1.4293267 ,  1.9909474 , -4.923098  ,
  1.2552458 , -3.9267457 ,  6.712175  , -1.4241142 ,
 -5.589383  , 10.719287  , -4.8469186 ,  0.2690369 ,
 -1.9205508 , -3.8617089 ,  3.8061    , -0.6445294 ,
 -0.86160207,  3.1802042 , -4.3851337 ,  2.7969096 ,
 -1.161522  , -5.093157  ,  2.2180312 , -10.999288  ,
  3.261592  ,  0.91937965, -3.9755828 ,  1.959505  ,
 -13.40441   , -2.0512755 , -15.968756 , -0.99200344,
  1.5856717 ,  5.293834  , -7.0113754 ,  0.17416582,
  1.7392821 ,  4.9609838 , -1.1166772 , -8.603888  ,
  5.7432756 ,  3.7928197 ,  9.740997  ,  6.377664  ,
 -10.503758  , -10.319839  , -4.1690006 ,  1.6302071 ,
 -4.7865286 , -0.97159237, -1.1480995 , -8.546399  ,
 -1.6673336 , 10.684597  , -6.6626863 , -1.5147434 ,
 -0.5121564 , 11.031017  , -0.34850731,  3.6526282 ,
 -2.9895196 , -1.2809095 ,  3.0267022 ,  5.3742733 ,
  4.719764  , -8.857332  , -13.29697  ,  4.2683344 ,
  3.3305006 ,  3.171509  ,  4.580689  ,  3.2604735 ,
 -15.704077 ,  6.747136  , 10.841292  ,  1.2590234 ,
 -4.364266  ,  9.069968  ,  1.9469426 ,  5.8769536 ],
 dtype=float32)
```

Ushbu o‘rnatishlarga ega bo‘lgan holda, tabiiy tilga ishlov berishning ba’zi qiziqarli vazifalarini bajarish mumkin. Ulardan biri turli so‘zlar (obyektlar) o‘rtasidagi o‘xshashlikni aniqlashdir.

```
print(ft_model.wv.similarity(w1='somsa',
w2='bayram'))
print(ft_model.wv.similarity(w1='idish',
w2='maymun'))
```

```
0.24808136
-0.05988481
```

So‘zlarning joylashuvi mavjudligini hisobga olsak, berilgan so‘zlar qatoridan ortiqcha so‘zlarni quyidagicha topishimiz mumkin:

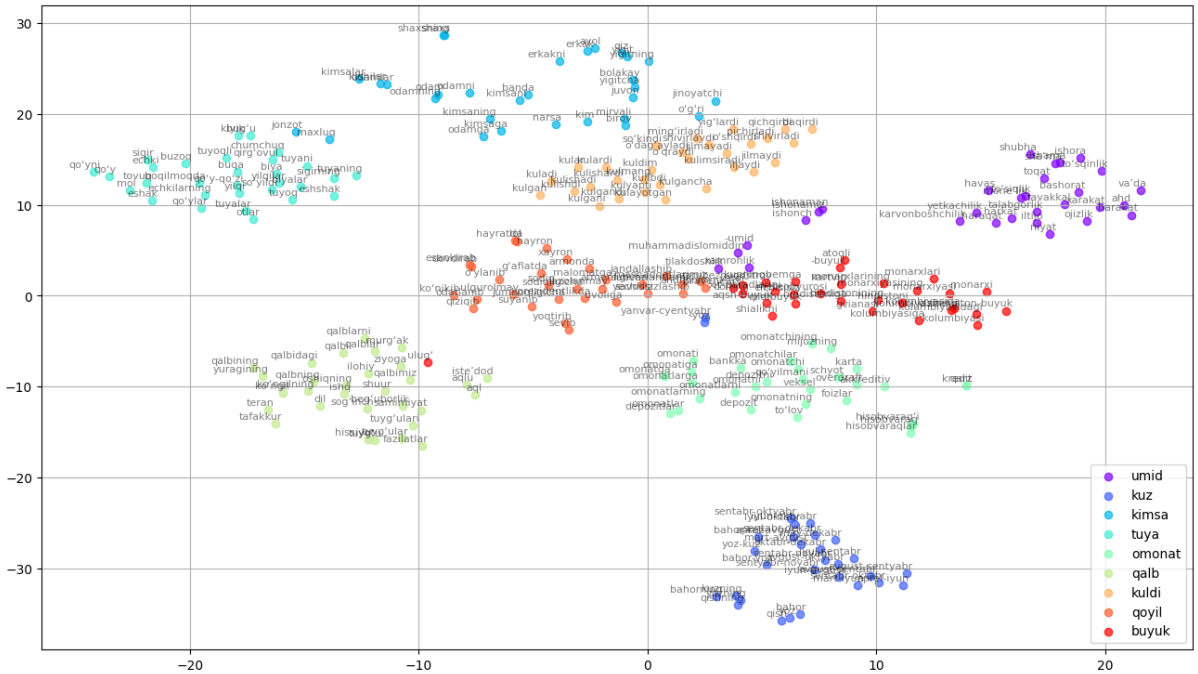
```
st1 = "tuyaqush go‘shti bayram ulashildi"
print('Odd one out for [',st1, ']:',
    ft_model.wv.doesnt_match(st1.split()))
```

———

Odd one out for [tuyaqush go‘shti bayram ulashildi]: ulashildi

Hosil qilingan natijadan so‘zlar to‘plamiga semantik jihatdan o‘xhash bo‘lmagan so‘zlar aniqlangani qayd etiladi.

Shuningdek, chuqur neyron tarmoqlar vositasida hosil qilingan modellarni doimiy xotira (disk)ga saqlab, keyinchalik ularni yuklab olib, NLPning turli vazifalarini hal qilishda foydalanish mumkin. Quyida 5Gb hajmga ega o‘zbek tili korpusi asosida ishlab chiqilgan CBOW modelidagi ba’zi so‘zlarning o‘xshash variantlarini aniqlaymiz:



18-rasm. *O`zbek tili korpusi matnlari asosida shakllantirilgan CBOW modelidagi ba’zi so‘zlarning o‘xshash variantlari*

Xulosa

Ushbu maqolaning asosiy maqsadi matn tahlili vazifalarini bajarish uchun chuqur o‘rganish muhitida so‘zlarni joylashtirish samaradorligini o‘rganish; natijalar asosida undan foydalanishni tavsiya qilishdan iborat. Ushbu maqolada NLP sohasidagi so‘nggi tadqiqot tendentsiyalarini qamrab olish va matn tahlili vazifalarida samarali natijalarga erishish uchun so‘zlarni joylashtirish va chuqur o‘rganish modellaridan qanday foydalanish batafsil tushuntirildi. Muayyan NLP vazifasini hal qilishga qaratilgan so‘zlarni joylashtirish modellari va chuqur o‘rganish yondashuvlari bo‘yicha olib borilgan tadqiqotlar tizimli tarzda sharhlandi. Bugungi kunda o‘zbek tili matnlari tahlili vazifalari uchun so‘zlarni joylashtirish usullari va chuqur o‘rganish modellarini tanlash, ularning kuchli va zaif tomonlarini tavsiflovchi ilmiy izlanishlar deyarli olib borilmagan. Ushbu maqolada o‘zbek tili matn tahlilidagi asosiy yangiliklardan biri – *so‘zlarni joylashtirish (word embeddings)* yoki *so‘z vektorlari (word vectors)* o‘rganildi. Bu nafaqat hujjat va so‘zlarni ifodalash, balki so‘zlarga yangicha qarashga ham yordam beradi. Word2Vec metodining muvaffaqiyati korpus asosida hosil qilingan model vositasida so‘zlarni joylashtirish NLP masalasida katta shov-shuvga sabab bo‘ldi, ushbu turdagi yangi usullarni ishlab chiqishda asos bo‘lib xizmat qildi. Ushbu maqolada Word2Vec, GloVe va FastText kabi mashinali o‘qitishning chuqur o‘rganish usullari haqida batafsil ma’lumot berildi hamda ularning afzallik va kamchiliklari keltirildi. Maqolada keltirilgan misollar orqali matn ma’lumotlaridan xususiyatlarni ajratib olish uchun chuqur o‘rganish til modellaridan

foydalanish bo'yicha yangi va samarali strategiyalar haqida ma'lumot berildi. Shuningdek, maqolada so'z semantikasi, kontekst va ma'lumotlarning kamligi kabi muammolarni hal qilish usullari keltirildi. O'zbek tilidagi matnli ma'lumotlarni Word2Vec, GloVe va FastText chuqur o'rganish usullaridan foydalanib, sonli ko'rinishlarini hosil qilish; ulardan NLPning turli vazifalarini hal qilishda foydalanish usullari keltirildi va namunaviy til korpusiga qo'llandi. Maqolada amalga oshirilgan ishlarni umumlashtirib quyidagi xulosalarni keltirish mumkin:

1. Tadqiqotda NLP tadqiqotchilarning so'zlarni joylashtirish modellari va ularning turli NLP ilovalarini umumiy rivojlanishiga qo'shgan hissalari o'rganildi.
2. So'zlarni joylashtirish va chuqur o'rganish modellari arxitekturasi keltirildi va amaliy misollar orqali ko'rib chiqildi.
3. Matn tahlili bo'yicha amalga oshirilgan tadqiqotlar sharhi va so'zlarni joylashtirish usullaridan foydalanishni ko'rib chiqish mezonlarga ko'ra tasniflandi.
4. Matn tahlili vazifasini bajarish uchun chuqur o'rganish metodlari orqali so'zlarni joylashtirish usullari samaradorligi o'rganildi; olingan natijalar muhokama qilindi.
5. Word2Vec, GloVe va FastText chuqur o'rganish usullari vositasida qayta ishlash namunaviy o'zbek tili korpusi matnlariga qo'llandi.
6. Tadqiqotda so'zlarni sonli ifodalashning turli yondashuv va chuqur o'rganish modellarining asoslari, afzallik va kamchiliklari, shuningdek, kelajakdagi tadqiqotlar istiqbollari haqida qisqacha ma'lumot berildi.

Foydalanilgan adabiyotlar

1. Asudani, D. S., Nagwani, N. K., & Singh, P. (2023). Impact of word embedding models on text analytics in deep learning environment: a review. *Artificial Intelligence Review*, *56*(9).
2. https://doi.org/10.1007/s10462-023-10419-1
3. Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing [Review Article]. In *IEEE Computational Intelligence Magazine* (Vol. 13, Issue 3). https://doi.org/10.1109/MCI.2018.2840738
4. Lavanya, P. M., & Sasikala, E. (2021). Deep learning techniques on text classification using Natural language processing (NLP) in social healthcare network: A comprehensive survey. *2021 3rd International Conference on Signal Processing and Communication, ICPSC 2021*. https://doi.org/10.1109/ICSPC51351.2021.9451752
5. Moreo, A., Esuli, A., & Sebastiani, F. (2021). Word-class embeddings for multiclass text classification. *Data Mining and Knowledge Discovery*, *35*(3). https://doi.org/10.1007/s10618-020-00735-3
6. B.Elov, Z.Xusainova, N.Xudayberganov. (2022). Tabiiy tilni qayta ishlashda Bag of Words algoritmidan foydalanish. *O'zbekiston: til va madaniyat (Amaliy filologiya), 2022, 5(4).* 31-45
7. Elov B., Hamroyeva Sh., Matyakubova N., Yodgorov U. One-hot encoding and Bag-of-Words methods in processing the uzbek language corpus texts // Труды XI Международной конференции «Компьютерная обработка тюркских языков» «TURKLANG 2023». Бухара, 20-22 октября 2023 г.
8. B.Elov, Z.Xusainova, N.Xudayberganov. O'zbek tili korpusi matnlari uchun TF-IDF statistik ko'rsatkichni hisoblash. *SCIENCE AND INNOVATION INTERNATIONAL SCIENTIFIC JOURNAL VOLUME 1 ISSUE 8 UIF-2022: 8.2 | ISSN: 2181-3337*
9. Erk, K. (2012). Vector Space Models of Word Meaning and Phrase Meaning: A Survey. *Linguistics and Language Compass*, *6*(10). https://doi.org/10.1002/lnco.362
10. Salton, G., Wong, A., & Yang, C. S. (1975). A Vector Space Model for Automatic Indexing. *Communications of the ACM*, *18*(11). https://doi.org/10.1145/361219.361220
11. Liu, G., Lu, Y., Shi, K., Chang, J., & Wei, X. (2019). Mapping Bug Reports to Relevant Source Code Files Based on the Vector Space Model and Word Embedding. *IEEE Access*, *7*. https://doi.org/10.1109/ACCESS.2019.2922686

12. Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, *41*(6). https://doi.org/10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9
13. Campbell, J. C., Hindle, A., & Stroulia, E. (2015). Latent Dirichlet Allocation: Extracting Topics from Software Engineering Data. In *The Art and Science of Analyzing Software Data*.
14. https://doi.org/10.1016/B978-0-12-411519-4.00006-9
15. Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*(6). https://doi.org/10.1162/153244303322533223
16. Wang, B., Wang, A., Chen, F., Wang, Y., & Kuo, C. C. J. (2019). Evaluating word embedding models: Methods and experimental results. In *APSIPA Transactions on Signal and Information Processing* (Vol. 8). https://doi.org/10.1017/ATSIP.2019.12
17. Shahzad, K., Kanwal, S., Malik, K., Aslam, F., & Ali, M. (2019). A word-embedding-based approach for accurate identification of corresponding activities. *Computers and Electrical Engineering*, *78*. https://doi.org/10.1016/j.compeleceng.2019.07.011
18. El-Demerdash, K., El-Khoribi, R. A., Ismail Shoman, M. A., & Abdou, S. (2022). Deep learning based fusion strategies for personality prediction. In *Egyptian Informatics Journal* (Vol. 23, Issue 1). https://doi.org/10.1016/j.eij.2021.05.004
19. Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. *OpenAI.Com*.
20. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, *1*.
21. Bakarov A. (2018) Asurvey of word embeddings evaluation methods. arXiv preprint https://doi.org/10.48550/arXiv.1801.09536
22. 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
23. 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*.
24. Schnabel, T., Labutov, I., Mimno, D., & Joachims, T. (2015). Evaluation methods for unsupervised word embeddings. *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.18653/v1/d15-1036
25. Glavaš, G., Litschko, R., Ruder, S., & Vulic, I. (2020). How to (properly) evaluate cross-lingual word embeddings: On strong baselines, comparative analyses, and some misconceptions. *ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*. https://doi.org/10.18653/v1/p19-1070
26. Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. *Proceedings of the 25th International Conference on Machine Learning*.
27. Mikolov, T., Yih, W. T., & Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. *Proceedings of the 2nd Workshop on Computational Linguistics for Literature, CLfL 2013 at the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2013*.
28. Wang, Y., Liu, S., Afzal, N., Rastegar-Mojarad, M., Wang, L., Shen, F., Kingsbury, P., & Liu, H. (2018). A comparison of word

embeddings for the biomedical natural language processing. *Journal of Biomedical Informatics*, *87*. https://doi.org/10.1016/j.jbi.2018.09.008

29. Levy, O., & Goldberg, Y. (2014). Dependency-based word embeddings. *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, *2*. https://doi.org/10.3115/v1/p14-2050
30. Yu, L. C., Wang, J., Robert Lai, K., & Zhang, X. (2018). Refining Word Embeddings Using Intensity Scores for Sentiment Analysis. *IEEE/ACM Transactions on Audio Speech and Language Processing*, *26*(3). https://doi.org/10.1109/TASLP.2017.2788182
31. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. In *Introduction to Information Retrieval*. https://doi.org/10.1017/cbo9780511809071
32. Ganguly, D., Roy, D., Mitra, M., & Jones, G. J. F. (2015). A word embedding based generalized language model for information retrieval. *SIGIR 2015 - Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. https://doi.org/10.1145/2766462.2767780
33. Wang, Y., Wang, L., Rastegar-Mojarad, M., Moon, S., Shen, F., Afzal, N., Liu, S., Zeng, Y., Mehrabi, S., Sohn, S., & Liu, H. (2018). Clinical information extraction applications: A literature review. In *Journal of Biomedical Informatics* (Vol. 77). https://doi.org/10.1016/j.jbi.2017.11.011
34. Zeng, D., Liu, K., Lai, S., Zhou, G., & Zhao, J. (2014). Relation classification via convolutional deep neural network. *COLING 2014 - 25th International Conference on Computational Linguistics, Proceedings of COLING 2014: Technical Papers*.
35. Nguyen, T. H., & Grishman, R. (2014). Employing word representations and regularization for domain adaptation of relation extraction. *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, *2*. https://doi.org/10.3115/v1/p14-2012
36. Chen, W., Zhang, M., & Zhang, Y. (2015). Distributed feature representations for dependency parsing. *IEEE Transactions on Audio, Speech and Language Processing*, *23*(3). https://doi.org/10.1109/TASLP.2014.2365359
37. Ouchi, H., Duh, K., Shindo, H., & Matsumoto, Y. (2016). Transition-based dependency parsing exploiting supertags. *IEEE/ACM Transactions on Audio Speech and Language Processing*, *24*(11). https://doi.org/10.1109/TASLP.2016.2598310
38. Shen, M., Kawahara, D., & Kurohashi, S. (2014). Dependency parse reranking with rich subtree features. *IEEE Transactions on Audio, Speech and Language Processing*, *22*(7). https://doi.org/10.1109/TASLP.2014.2327295
39. Tang, D., Wei, F., Yang, N., Zhou, M., Liu, T., & Qin, B. (2014). Learning sentiment-specific word embedding for twitter sentiment classification. *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, *1*. https://doi.org/10.3115/v1/p14-1146
40. Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. *ACL-HLT 2011 - Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, *1*.
41. Zhou, G., Xie, Z., He, T., Zhao, J., & Hu, X. T. (2016). Learning the Multilingual Translation Representations for Question Retrieval in Community Question Answering via Non-Negative Matrix Factorization. *IEEE/ACM Transactions on Audio Speech and Language Processing*, *24*(7). https://doi.org/10.1109/TASLP.2016.2544661
42. Hao, Y., Zhang, Y., Liu, K., He, S., Liu, Z., Wu, H., & Zhao, J. (2017). An end-to-end model for question answering over knowledge base with cross-attention

combining global knowledge. *ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)*, *1*. https://doi.org/10.18653/v1/P17-1021
43. Ren, M., Kiros, R., & Zemel, R. S. (2015). Exploring models and data for image question answering. *Advances in Neural Information Processing Systems*, *2015-January*.
44. Dong, L., Wei, F., Zhou, M., & Xu, K. (2015). Question answering over freebase with multi-column convolutional neural networks. *ACL-IJCNLP 2015 - 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Proceedings of the Conference*, *1*. https://doi.org/10.3115/v1/p15-1026
45. Yogatama, D., Liu, F., & Smith, N. A. (2015). Extractive summarization by maximizing semantic volume. *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*.
46. https://doi.org/10.18653/v1/d15-1228
47. Rush, A. M., Chopra, S., & Weston, J. (2015). A Neural Attention Model for Abstractive Sentence Summarization. *Proceedings of EMNLP 2015*, *1509*(685).
48. Zhang, B., Xiong, D., Su, J., & Duan, H. (2017). A Context-Aware Recurrent Encoder for Neural Machine Translation. *IEEE/ACM Transactions on Audio Speech and Language Processing*, *25*(12). https://doi.org/10.1109/TASLP.2017.2751420
49. Chen, K., Zhao, T., Yang, M., Liu, L., Tamura, A., Wang, R., Utiyama, M., & Sumita, E. (2018). A Neural Approach to Source Dependence Based Context Model for Statistical Machine Translation. *IEEE/ACM Transactions on Audio Speech and Language Processing*, *26*(2). https://doi.org/10.1109/TASLP.2017.2772846
50. Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*. https://doi.org/10.3115/v1/d14-1162
51. Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, *5*.
52. https://doi.org/10.1162/tacl_a_00051
53. Glavaš, G., Litschko, R., Ruder, S., & Vulic, I. (2020). How to (properly) evaluate cross-lingual word embeddings: On strong baselines, comparative analyses, and some misconceptions. *ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*. https://doi.org/10.18653/v1/p19-1070
54. Qiu, Y., Li, H., Li, S., Jiang, Y., Hu, R., & Yang, L. (2018). Revisiting correlations between intrinsic and extrinsic evaluations of word embeddings. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11221 LNAI*. https://doi.org/10.1007/978-3-030-01716-3_18
55. Megan Leszczynski, Avner May, Jian Zhang, Sen Wu, Christopher R. Aberger, and Christopher R´ e. (2020). *Understanding the downstream instability of word embeddings. In Proceedings of MLSys, pages 262–290.* https://doi.org/10.48550/arXiv.2003.04983
56. Baroni, M., Dinu, G., & Kruszewski, G. (2014). Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, *1*. https://doi.org/10.3115/v1/p14-1023
57. Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. *Proceedings of the 25th International Conference on Machine Learning*.
58. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from

scratch. *Journal of Machine Learning Research*, *12*.
59. Tjong Kim Sang, E. F., & de Meulder, F. (2003). Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. *Proceedings of the 7th Conference on Natural Language Learning, CoNLL 2003 at HLT-NAACL 2003*.
60. Søgaard, A. (2016). Evaluating word embeddings with fmri and eye-tracking. *Proceedings of the Annual Meeting of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/w16-2521
61. Schwenk, H. (2013). CSLM - A modular open-source continuous space language modeling toolkit. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*.
62. https://doi.org/10.21437/interspeech.2013-326
63. Li, Z., Zhang, M., Che, W., Liu, T., & Chen, W. (2014). Joint optimization for Chinese POS tagging and dependency parsing. *IEEE Transactions on Audio, Speech and Language Processing*, *22* (1). https://doi.org/10.1109/TASLP.2013.2288081
64. Xu, J., He, H., Sun, X., Ren, X., & Li, S. (2018). Cross-Domain and Semisupervised Named Entity Recognition in Chinese Social Media: A Unified Model. *IEEE/ACM Transactions on Audio Speech and Language Processing*, *26*(11).
65. https://doi.org/10.1109/TASLP.2018.2856625
66. Ravi, K., & Ravi, V. (2015). A survey on opinion mining and sentiment analysis: Tasks, approaches and applications. *Knowledge-Based Systems*, *89*. https://doi.org/10.1016/j.knosys.2015.06.015
67. Bahdanau, D., Cho, K. H., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.
68. (x1) Wang, Y., Huang, G., Li, J., Li, H., Zhou, Y., & Jiang, H. (2021). Refined Global Word Embeddings Based on Sentiment Concept for Sentiment Analysis. *IEEE Access*, *9*.
69. https://doi.org/10.1109/ACCESS.2021.3062654