SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4911

dna2vec: vektorska reprezentacija k-torki različite duljine

Mateo Kutnjak

Zagreb, svibanj 2017.

SADRŽAJ

1. Uvod

Analiza i manipulacija genima u novije vrijeme sve je raširenija tema. Sve se češće pojedini dio genoma povezuje sa bolestima kojima se nije znao uzrok. Kombinacija gena rezultirala je kvalitetnijim i dugotrajnijim prehrambenim proizvodima. Jednojajčani blizanci razdvojeni pri rođenju razvijaju slične karaktere. Genotip organizma određuje cjelokupna svojstva istog. Hartl i Jones (2005) tvrde kako su sve komponente fenotipa nužno određene genotipom i sukcesijom sredinskih uticaja sa kojim sudjeluju; nema živog bića bez genotipa niti on može egzistirati izvan prostorno–vremenskog kontinuuma vlastitog životnog okruženja. Razumijevanje genotipa nije samo želja nego potreba.

Uobičajan pristup u analizi dugačkih DNA fragmenata je podjela istih na kraće podnizove nazvanih k-torke. Pritom se k-torke prikazuju na različite načine. Reprezentacija k-torke kao bit-vektor je jednostavna i razumljiva. Bit-vektor sadrži jedinicu na dimenziji koja određuje k-torku, dok su ostale dimenzije jednake nuli.

Međutim, za izračunavanje sličnosti između dvije k-torke prikaz bit-vektorima je nepraktičan jer su udaljenosti između bilo koja dva bit-vektora iste, bez obzira što je npr. k-torki GCTGA sličnija GCTAA nego ATTGT. Potrebno je dobiti vektorski prikaz k-torki koji ne ovisi samo o k-torki koju opisuje, nego i o njenom kontekstu. Analogija se nalazi u kodiranju riječi korištenjem konteksta metodom *word2vec* i dobijanje vektorskog prostora za riječi vokabulara. Prilagođena izvedba može se iskoristiti za ispitivanje genoma.

Needleman i Wunsch (1970) razvili su algoritam koji mjeri razinu sličnosti DNA sekvenci i proteina. Vremenska složenost algoritma je kvadratna jer se koristi matrica za izračun sličnosti dinamičkim programiranjem.

U ovom radu ispituje se kodiranje k-torki varijabilne veličine izvedbom plitke neuronske mreže *word2vec* sa tri sloja. Ng (2017) tvrdi da postoji korelacija sličnosti k-torki algoritmom Needleman–Wunsch i kosinusne udaljenosti izlaznih vektora metode *dna2vec*. Cilj ovog rada je ispitati tu povezanost i ispitati složenost *dna2vec* pristupa vlastitom implementacijom neuronske mreže.

2. Pregled područja

Već početkom razvoja računarstva javila se ideja o razumijevanju prirodnog jezika od strane računala. Algoritamski se taj problem pokazao kao neriješiv zbog ogromnog broja riječi jezika te komplesnog odnosa između njih. Zbog komplicirane strukture nemoguće je doslovno prevoditi značenje svake riječi da bi se dobilo značenje izraza. Posljednjih godina pri strojnom učenju koriste se *word-embeddings*, tj. prikazivanje riječi prirodnog jezika vektorom realnih brojeva.

2.1. Word2vec

Kodirani prikaz riječi dobiva se *word2vec* metodom: neuronskom mrežom od tri sloja koja osim same riječi teksta u obzir uzima i kontekst u kojem se riječ nalazi. Prolaskom kroz dani tekst neuronska mreža gradi odnose između riječi vokabulara. Nakon procesa učenja uspoređivanjem izlaznih vektora za danu riječ dobiva se jedinstveni vektorski prikaz za pojedinu riječ kojim je jedinstveno određena u višedimenzijskom vektorskom prostoru.

Temelj za navedenu usporedbu riječi dao je Harris (1954) u tzv. distribucijskoj hipotezi kojom tvrdi da riječi sličnog konteksta imaju bliskije značenje.

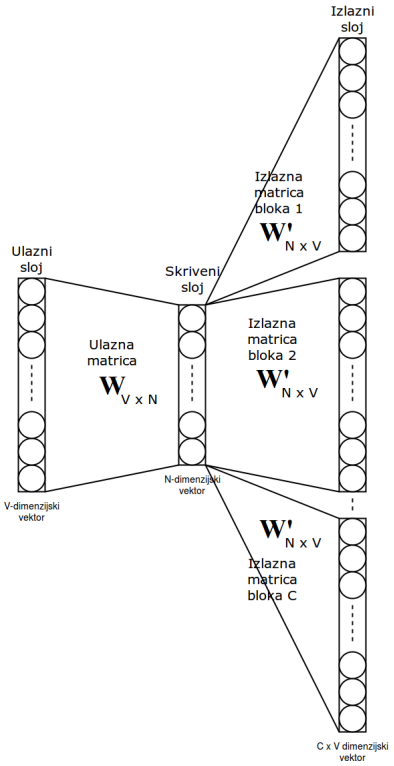
Arhitektura je određena nekom od izvedbi *word2vec* mreže koje uključuju *Continuous Bag-of-Word* (CBOW) te *Skip-Gram* model. Dimenzije ulaznih i izlaznih vektora ovise o modelu.

Obje izvedbe čine tri neuronska sloja. Težine između neurona prikazane su matricama čije dimenzije odgovaraju veličini slojeva neurona koji ih okružuju. Izračun izlaznog vektora zahtijeva matrična množenja te skaliranje vjerojatnosti izlaza *softmax* funkcijom. Postupak učenja čini se podešavanjem težina matrica mreže gradijentnim spustom s ciljem smanjivanje funkcije gubitka. Pritom stopa učenja definira brzinu prilagodbe težina. Da bi se ubrzao proces učenja nad velikim skupom podataka dodatno se implementira jedna od optimizacija: *negative sampling* ili *hierarchical softmax*.

Skip-Gram model će biti detaljnije opisan jer ga *dna2vec* koristi.

2.2. Skip-Gram model

Slika 2.1 prikazuje *Skip-Gram* model. Rong (2014) daje matematički prikaz metode *word2vec*. Za razliku od CBOW modela, *Skip-Gram* na ulaz prima bit-vektor samo ciljane riječi, a kao očekivani izlaz bit-vektore riječi konteksta koje prvotnu riječ okružuju. Dakle, jedan primjer za treniranje uključuje ulazni bit-vektor i skup izlaznih bit-vektora. Broj riječi konteksta koje se koriste određuje veličina tzv. "prozora". Na primjer, ako je veličina prozora 5, mreža koristi pet riječi teksta iza i pet riječi teksta ispred ulazne. Veličinu vokabulara označava se slovom V, a veličina skrivenog sloja



Slika 2.1: *Skip-Gram* model

slovom N. Ulazni bit-vektor je prema tome dimenzije V sa jedinicom na poziciji rednog broja riječi koju reprezentira u vokabularu. Broj riječi iz konteksta prikazuje se slovom C i jednak je dvostrukoj veličini prozora.

Neuroni susjednih slojeva povezani su matricama. Težine između ulaznog i skrivenog sloja dane su ulaznom matricom $\mathbf{W}$ dimenzija $V \times N$. Skriveni sloj iz matrice kopira taj redak množenjem translatirane ulazne matrice i ulaznog vektora:

$$\mathbf{h} = \mathbf{W}^T\mathbf{x} = \mathbf{v}_{w_I}, \tag{2.1}$$

gdje je $\mathbf{h}$ vektorska reprezentacija ulazne riječi od N dimenzija na izlazu skrivenog

sloja. Aktivacijska funkcija skrivenog sloja je linearna zbog jednostavnog kopiranja vrijednosti retka ulazne matrice na skriveni sloj.

Između skrivenog i izlaznog sloja nalazi se C izlaznih matrica $\mathbf{W}'$ veličina $N \times V$. Svaki blok matrica daje izlazni vektor za jednu riječ konteksta formulom

$$\mathbf{u}_c = \mathbf{W}'^T_c \mathbf{h}, \tag{2.2}$$

gdje je $\mathbf{u}_c$ prvotni rezultat c-tog bloka izlaznog sloja. Dimenzija vektora $\mathbf{u_c}$ iznosi V.

Vjerojatnosti odgovaranja pojedine riječi dijelu konteksta dobija se *softmax* funkcijom

$$p(w_{c,j} = w_{O,c}|w_I) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u_{c,j'})}, \tag{2.3}$$

koja skalira zbroj izlaznog sloja mreže na jedan. Ovdje $w_{c,j}$ predstavlja j-tu riječ c-tog bloka izlaznog sloja, koji je zapravo uvjetna vjerojatnost da za ulaznu riječ w_I odgovarajuća riječ konteksta u c-tom bloku bude $w_{O,j}$. $y_{c,j}$ je krajnja vjerojatnost j-te riječi da pripada u c-ti blok konteksta ulazne riječi nakon primjene *softmax* funkcije.

Koraci 2.1, 2.2 i 2.3 zovu se *forward propagation* i rezultiraju kodiranjem riječi s obzirom na kontekst koji ju okružuje. Promjena težina matrica obavlja se algoritmom *backpropagation*. Veličina koja prikazuje točnost mreže zove se "funkcija gubitka":

$$\begin{aligned} E &= -\log p(w_{O,1}, w_{O,2}, \ldots, w_{O,C}|w_I) \\ &= -\sum_{c=1}^{C} u_{j_c^*} + C \cdot \log \sum_{j'=1}^{V} \exp(u'_j) \end{aligned} \tag{2.4}$$

gdje je j_c^* indeks riječi u c-tom bloku koja je očekivana s obzirom na dani primjer za učenje. Cilj učenja je minimizirati funkciju gubitka, pa se derivacijom

$$\frac{\partial E}{\partial u_{c,j}} = y_{c,j} - t_{c,j} := e \tag{2.5}$$

funkcije gubitka s obzirom na rezultat jedinice izlaznog sloja $u_{c,j}$ dobija pretpostavka greške $e_{c,j}$ te jedinice. Dimenzija jednog bloka procijenjene pogreške $\mathbf{e}_c$ iznosi V. Gradijentni spust izlaznih matrica dobija se derivacijom funkcije gubitka u odnosu na svaki član istih matrica izrazom

$$\frac{\partial E}{\partial w'_{ij}} = \sum_{c=1}^{C} \frac{\partial E}{\partial u_{c,j}} \cdot \frac{\partial u_{c,j}}{\partial w'_{ij}} = \mathrm{EI}_j \cdot h_i, \tag{2.6}$$

gdje je h_i i-ti član izlaza skrivenog sloja, a EI suma svih vektora pogreške po blokovima konteksta. Kako su članovi desne strane skalari, gradijentni spust svakog člana

izlazne matrice je također skalar. Dodatni član

$$\mathrm{EI}_j = \sum_{c=1}^{C} e_{c,j} \tag{2.7}$$

stvoren je zbog jednostavnijeg zapisa, ali se u stvarnosti gradijentni spust pojedine izlazne matrice dobiva kao umnožak $\mathbf{h} \cdot \mathbf{e}_c^T$ gdje je $\mathbf{e}_c^T$ transponirana procijenjena pogreška c-tog bloka izlaznog sloja.

Dobiveni stupanj gradijentnog spusta koristi se kao korekcija trenutnih težina izlaznih matrica, pa općenito vrijedi

$$w'_{ij} \quad \leftarrow \quad w'_{ij} - \eta \cdot \mathrm{EI}_j \cdot h_i \tag{2.8}$$

gdje je η stopa učenja. Gornja korekcija radi se za sve članove svake od izlaznih matrica. Opet napominjem da EI postoji zboj jednostavnijeg zapisa te da se za korekciju matrice bloka c koristi pripadni vektor predikcija greške $\mathbf{e}_c$.

Gradijentni spust za korekciju ulazne matrice dobija se izrazom

$$\mathrm{EH} = \sum_{c=1}^{C} \mathbf{W}'_c \mathbf{e}_c \tag{2.9}$$

gdje je EH vektor gradijentnog spusta ulazne matrice. Kako je izlazna matrica dimenzija $N \times V$, a vektor predviđanja pogreške dimenzije V, rezultat je vektor EH dimenzije N.

Podešavanje ulazne matrice obavlja se sljedećim postupkom:

$$\mathbf{v}_{\mathrm{w_I}} \quad \leftarrow \quad \mathbf{v}_{\mathrm{w_I}} - \eta \cdot \mathrm{EH}^T \tag{2.10}$$

gdje je $\mathbf{v}_{\mathrm{w_I}}$ vektor redak matrice koji odgovara indeksu ulazne riječi.

Prethodno opisani postupak nije optimiziran, pa nije primjenjiv na velik skup podataka zbog predugog trajanja učenja.

Za razliku od CBOW modela, proces učenja *Skip-Gram* modela traje duže, ali daje bolje rezultate za riječi s manje pojavljivanja u tekstu.

2.3. Negative Sampling

Ova optimizacijska tehnika omogućuje rad s~~a~~ većim skupom podataka. Kako dimenzija vokabulara raste, raste i dimenzija izlaza mreže. Svaka dodatna riječ u ~~vokabularu~~ traži nekoliko operacija više za svaki primjer učenja.

Rješenje su korekcije manjeg broja riječi i pripadnih im redaka u težinskim matricama. U obzir se uzima "positive sample", tj. ulazna riječ te nekoliko odabranih "negative samples" temeljem vjerojatnosne distribucije $P_n(w)$. Distibucija je proizvoljna.

Rong (2014) naglašava da funkcija gubitka oblika

$$E = -\log \sigma({\mathbf{w}'_{w_O}}^T \mathbf{h}) - \sum_{w_j \in W_{neg}} \log \sigma(-{\mathbf{w}'_{w_j}}^T \mathbf{h}) \tag{2.11}$$

daje dobra kodiranja riječi. Ovdje je w_O *positive sample*, a $\mathbf{w}'_{w_O}$ pripadni vektor izlazne matrice, a W_{neg} skup negativnih uzoraka proizvoljne veličine odabranih temeljem distribucije $P_n(w)$.

Za razliku od klasičnog *Skip-Gram* modela, gradijentni spust izračunava se samo za pozitivan uzorak i negativne uzorke. Derivacijom funkcije gubitka po izlazu izlaznog sloja

$$\begin{aligned} \frac{\partial E}{\partial {\mathbf{v}'_{w_j}}^T \mathbf{h}} &= \begin{cases} \sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) - 1 & \text{akoje } w_j = w_O \\ \sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) & \text{akoje } w_j \in W_{neg} \end{cases} \\ &= \sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) - t_j \end{aligned} \tag{2.12}$$

dobivaju se skalari kojima se korigiraju posebno retci pozitivnog uzorka, a posebno retci negativnih uzoraka. Vrijedi $t = 1$ ako je w_j pozitivni uzorak, a inače $t = 0$. Dobivene vrijednosti koriste se pri izačunu gradijentnog spusta u izrazu

$$\frac{\partial E}{\partial \mathbf{v}'_{w_j}} = \Big(\sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) - t_j\Big)\mathbf{h} \tag{2.13}$$

gdje je rezultat vektor dimenzije N kojim se korigira pripadni stupac izlazne matrice formulom

$$\mathbf{v}'_{w_j} \quad \leftarrow \quad \mathbf{v}'_{w_j} - \eta \left(\sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) - t_j\Big)\mathbf{h}. \tag{2.14}$$

Što se tiče ulazne matrice, njezine težine se mijenjaju ovisno o derivaciji funkcije gubitka s obzirom na izlaz skrivenog sloja:

$$\frac{\partial E}{\partial \mathbf{h}} = \sum_{w_j \in \{w_O\} \cup W_{neg}} \Big(\sigma({\mathbf{v}'_{w_j}}^T \mathbf{h}) - t_j\Big)\mathbf{v}'_{w_j} := \mathrm{EH} \tag{2.15}$$

gdje je EH umožak koeficijenta skaliranja dobivenog izrazom 2.12 i pripadajućeg stupca izlazne matrice. Zbrajanjem umnožaka pozitivnog uzorka kao i svih negativnih dobija se vektor korekcije EH koji se koristi jednako kao u izrazu 2.10.

3. Metoda

U ovom radu opisan je i ispitan postupak *dna2vec*. Prethodno opisana metoda *word2vec* prilagođena je drugoj svrsi: umjesto riječi teksta koristi se vokabular genoma, odnosno k-torke promjenjive duljine za koje se neuronskom mrežom dobiva vektorska reprezentacija. Izračunatim izlaznim vektorom k-torka se stavlja u višedimenzijski vektorski prostor.

Dobivena vektorska reprezentacija ovisi o kontekstu u kojem se k-torka nalazi. Takav zapis omogućuje usporedbu sličnosti k-torki, odnosno njihovih vektorskih zapisa.

Uspoređivanje k-torki dinamičkim programiranjem je nepraktično zbog velike $O(n^2)$ vremenske složenosti. Algoritam Needleman–Wunsch primjer je takve usporedbe, gdje se dobiva najmanji broj operacija brisanja, dodavanja ili promjena znakova da se iz jedne k-torke dobije druga.

Namjera metode *dna2vec* je razvoj točnog i brzog načina izračunavanja sličnosti između k-torki koji je analogan algoritmu Needleman–Wunsch. Pretpostavka je da kosinusna udaljenost dvaju izlaznih vektora mreže odgovara njihovoj sličnosti dobivenoj algoritmom Needleman–Wunsch.

Izgradnja modela sastoji se od četiri dijela:

1. razdvajanje genoma na fragmente
2. podjela fragmenata na preklapajuće k-torke različitih duljina
3. treniranje modela korištenjem neuronske mreže s tri sloja
4. dekompozicija modela na manje s obzirom na duljinu k-torki

3.1. Razdvajanje genoma

Fragmentiranje cjelokupnog genoma radi se s obzirom na razdjelne znakove. Razdjelni znakovi ovise o izvoru s kojeg se genom preuzima. Kod genoma čovjeka sa stranice *http://hgdownload.cse.ucsc.edu/downloads.html#human* to je N a se poveća entropija, dodatno se može uzeti i komplement fragmenta

Genom *Escherichia coli* bakterije koji se koristi u ovom radu nema razdjelnih znakova i ne sadrži komplement.

3.2. Preklapajuće k-torke

Dijeljenje fragmenta na k-torke radi se s pomičnim prozorom varijabilne duljine. Veličina prozora mijenja pri svakom pomaku po uniformnoj razdiobi $\mathcal{U}\{3, 8\}$ zbog promijenjive veličine k-torki modela.

Na primjer, za fragment GTCGTTAGA i razdiobu $\mathcal{U}\{3, 5\}$ dobivene k-torke mogle bi biti $\{\mathsf{GTCG}, \mathsf{TCG}, \mathsf{CGTTA}, \mathsf{GTT}, \mathsf{TTAGA}\}$. Obuhvaćanjem zadnjeg znaka fragmenta prekida se rastavljanje na k-torke.

3.3. Neuronska mreža

Koristi se neuronska mreža s tri sloja temeljena na *word2vec* metodi za treniranje modela. Kako postoji više izvedbi *word2vec* mreže, odabran je *Skip-Gram* model jer, iako je sporiji od *CBOW* modela, daje točnije rezultate za k-torke s manje pojavljivanja.

Odabrana mreža predviđa kontekst oko dane riječi. Kod *dna2vec* kontekst je skup k-torki koje okružuju ulaznu. Tako za prethodni primjer, pod uvjetom da je veličina prozora kojeg mreža koristi 2, kontekst k-torke CGTTA bi bio $\{\mathsf{GTCG}, \mathsf{TCG}, \mathsf{GTT}, \mathsf{TTAGA}\}$.

Za ispitivanje rada modela koristit će se prozor veličine 5, što daje kontekst veličine 10 za pojedinu k-torku.

Zbog povelikog skupa podataka, implementirana je optimizacijska tehnika *Negative Sampling* zbog ubrzanja učenja.

3.4. Dekompozicija modela

Traženje najbližeg zajedničkog susjeda zahtijeva rastavljanje osnovnog modela na manje gdje su obuhvaćene k-torke jednake duljine. Broj krajnjih modela iznosi $k_{high} - klow + 1$, što u opisanom slučaju daje šest modela.

4. Implementacija

Programsko rješenje napravljeno je u programskom jeziku *Python*. Projekt se sastoji od *.py* datoteka s programskim kodom, serijaliziranih datoteka te tekstualnog zapisa genoma *Escherichia coli* bakterije.

Programskog kod broji devet datoteka podijeljenih po odgovornostima koje će detaljnije biti opisane u sljedećim dijelovima poglavlja. Datoteke koje nisu izvršive sadrže po jedan razred.

Tablica 4.1: Datoteke programskog koda

Datoteka	Odgovornost	Izvršivost
DNA_data	Statički dostupne strukture podataka	-
DNA_iterator	Dohvat riječi i konteksta	-
DNA_kmerinfo	Potpora ispitivanju sličnosti k-torki	-
DNA_network	Neuronska mreža sa optimizacijom	-
DNA_test	Ispitivanje naučenog modela	+
DNA_tokenizer	Podjela genoma na k-torke	-
DNA_train	Treniranje mreže	+
DNA_unigram	Unigram tablica za optimizaciju mreže	-
DNA_util	Pomoćne funkcije	-

Serijalizirani objekti koriste se za spremanje trenutne neuronske mreže u procesu učenja te objekata sa kojima ona radi. Nakon učenja finalna mreža sprema se na isti način.

Genom pod imenom "E. coli K-12 MG1655 U00096.1" u tekstualnom formatu preuzet je sa stranice imena *http://www.ecogene.org/old/SequenceDownload.php* i pune je duljine bez komplementa.

Moduli koji su korišteni u implementaciji su:

Tablica 4.2: Funkcije korištenih modula

Modul	Funkcija
numpy	linearna algebra, matematičke funkcije
pickle	serijalizacija i deserijalizacija objekata
os.path	ispitivanje postojanja datoteka
random	generiranje slučajnih brojeva
Bio	Needleman-Wunsch algoritam

4.1. DNA_data

Datoteka *DNA_data* sadrži jedan razred **Data** sa statičkim podacima. Zbog potrebne brzine pristupa podacima k-torki ovaj razred koristi četiri *dictionary*-ja.

Tablica 4.3: Statičke varijable razreda **Data**

Tip	Naziv	Ključ	Vrijednost
`dict`	`index_of_kmer`	string k-torke	indeks k-torke
`dict`	`kmer_of_index`	indeks k-torke	string k-torke
`dict`	`kmer_count`	indeks k-torke	broj pojavljivanja k-torke
`dict`	`kmer_distribution`	index k-torke	razdioba $P_n(w)$ k-torke

Kod treniranja mreže koristi se "index_of_kmer" jer se rastavljanjem tekstualnog zapisa genoma na k-torke dobija znakovni prikaz, a mreža koristi indekse za izgradnju bit-vektora i indeksiranja vektora matrica. Kod ispitivanja rezultata dobijaju se vjerojatnosti po indeksima k-torki koji se rječnikom "kmer_of_index" pretvaraju u znakovni zapis.

Rječnik "kmer_count" broji pojavljivanje k-torki prilikom rastavljanja fragmenta. Jedini razlog brojanja k-torki je izgradnja unigram tablice za koju je potrebno izračunati vjerojatnosnu razdiobu $P_n(w)$ svake k-torke. Zadnji rječnik sadrži vrijednost $P_n(w)$ za svaku k-torku.

4.2. DNA_iterator

Primjena razreda **Iterator** kojeg datoteka *DNA_iterator* sadrži je slijedni prolaz po k-torkama te dohvaćanje njih i njihovog konteksta.

Tablica 4.4: Članske varijable razreda **Iterator**

Tip	Naziv	Opis
`int`	`window_size`	veličina prozora kojeg mreža koristi
`int`	`counter`	redni broj sljedećeg k-mera i pripadnog konteksta
`list`	`kmers`	lista svih k-torki fragmenta

Sve članske varijable se dobijaju kao parametri u konstruktoru razreda. Konstruktor prima veličinu prozora te listu svih k-torki fragmenta. Inicijalni brojač postavlja se na velikinu prozora: prva k-torka s kontekstom koja se dohvaća je ona s indeksom jednakim veličine prozora.

Metode koje razred sadrži su sljedeće:

- `has_next()`
  - vraća bool vrijednost `True` ako postoji još k-torki s kontekstom u listi
- `next_kmer()`
  - iz liste svih k-torki uzima sljedeću k-torku i pripadni kontekst te ih vraća kao par vrijednosti

Obavezno je prije inicijalizacije objekta tipa **Iterator** imati listu k-torki koja se dobije inicijalizacijom objekta tipa **Tokenizer**.

4.3. DNA_kmerinfo

Razred **Kmer_info** kojeg datoteka *DNA_kmerinfo* sadrži, koristi se za ispitivanje *dna2vec* modela nakon procesa učenja neuronske mreže.

Podjela na manje modele diskretno se odvija u metodama ovog razreda. Odabirom željene duljine traženog najbližeg susjeda određuje se interval indeksa k-torki kojem odgovara određena duljina.

Tablica 4.5: Članske varijable razreda **Kmer_info**

Tip	Naziv	Opis
`Network`	`network`	referenca na neuronsku mrežu

Članska varijabla se dobija preko parametra konstruktora. Referenca na neuronsku mrežu je potrebna jer metode za traženje najbližih susjeda moraju ispitivati rezultate mreže za k-torke.

Razred sadrži dvije metode za pronalazak najbližih susjeda:

- `nearest_neighbor(v, neighbor_len)`
    - vraća najbliži susjed duljine `neighbor_len` po kosinusnoj udaljenosti izlaznom vektoru ili znakovnom zapisu k-torke `v`
- `n_nearest_neighbors(v, neighbor_len, n)`
    - vraća listu od `n` najbližih susjeda duljine `neighbor_len` po kosinusnoj udaljenosti izlaznom vektoru ili znakovnom zapisu k-torke `v`

Svaka od metoda kao parametar `v` može primiti ili znakovni zapis k-torke ili izlazni vektor neuronske mreže dane k-torke.

Valja napomenuti da se metode najbližih susjeda ne koriste kod ispitivanja korelacije sličnosti s Needleman-Wunsch algoritmom, već kod ostalih mogućnosti *dna2vec* modela, kao što je konkatenacija k-torki pomoću aritmetike izlaznih vektora.

4.4. DNA_network

Glavni i najkompleksniji razred ovog modela nalazi se u datoteci *DNA_network*. Rezred **Network** izgrađuje neuronsku mrežu *Skip-Gram* sa optimizacijom *Negative Sampling*.

Parametri koji određuju arhitekturu mreže primaju se preko konstruktora razreda **Network**. Osim veličine vokabulara s kojim model radi i veličine skrivenog sloja, u konstruktoru se prima referenca na objekt razreda **Unigram** pomoću kojeg se određuju negativni uzorci za trenutni primjer učenja.

Dodatno se mogu kao parametri `learning_rate` i `ns_size` zadati stopa učenja odnosno broj negativnih uzoraka koji se koriste pri trenutnom primjeru učenja. Pretpostavljeni parametri su `learning_rate` $= 0.05$ i `ns_size` $= 5$.

Tablica 4.6: Članske varijable razreda **Network**

Tip	Naziv	Opis	Dimenzija
`float`	`learning_rate`	stopa učenja	-
`int`	`V`	veličina ulaznog sloja	-
`int`	`N`	veličina skrivenog sloja	-
`int`	`C`	broj blokova	-
`numpy.matrix`	`theta1`	ulazna matrica	$V \times N$
`list`	`theta2`	lista izlazna matrica	$C \cdot (N \times V)$
`numpy.matrix`	`h`	vektor rezultat skrivenog sloja	$N \times 1$
`list`	`u`	lista izlaznih vektora blokova	$C \cdot (V \times 1)$
`numpy.matrix`	`y`	izlaz nakon *softmax* funkcije	$C \cdot (V \times 1)$
`list`	`loss`	funkcija gubitka za svaki blok	C

Kod klasičnog modela bez *Negative Sampling* optimizacije, *forward propagation* izrađuje skup podataka koji su potrebni za *backpropagation* algoritam. Ova implementacija mreže nema klasičnu podjelu, već je čini metoda kojom se mreža trenira odnosno metoda kojom se izračunavaju rezultati za već naučenu mrežu.

4.4.1. Metoda result

`result(input_index)`

– vraća člansku varijablu `y`: vektor dimenzija $(C \cdot V)$ koji je izlaz neuronske mreže za dani indeks k-torke `input_index` nakon primjene *softmax* funkcije

Listing 4.1: Programski kod metode result

```
import numpy as np
...
def result(input_index):
        self.h = np.matrix(self.theta1[input_index, :]).
                copy().transpose()
        self.u = [num.transpose() * self.h for num in self.
            theta2]
        self.y = DNA_util.softmax_list(self.u)
        return np.concatenate(self.y)
```

4.4.2. Metoda train

`train(input_index, output_indexes)`

- za svaki blok matrica podešava težine prema optimizaciji *Negative Sampling*
- izračunava vrijednost funkcije gubitka za svaki od blokova podataka kao člansku varijablu `loss`

Algorithm 1 Negative Sampling

function TRAIN(w_I, $[w_O]$)
  `loss` $\leftarrow [\,]$
  for all w_O **do**
    $W_{neg} \leftarrow unigram.choose\_n(P_n)$
    $\mathbf{g}_{w_O} \leftarrow (\sigma(\mathbf{v'}_{w_O}^T \mathbf{v}_{w_I}) - 1)\mathbf{v}_{w_I}$
    $\mathbf{g}_{w_I} \leftarrow (\sigma(\mathbf{v'}_{w_O}^T \mathbf{v}_{w_I}) - 1)\mathbf{v}_{w_O}$
    $\mathbf{g}_{w_j} \leftarrow [\,]$
    $\texttt{loss}_{w_O} \leftarrow \texttt{loss}_{w_O} - \log(\sigma(\mathbf{v'}_{w_O}^T \mathbf{v}_{w_I}))$
    for all $w_j \in W_{neg}$ **do**
      $\mathbf{g}_{w_j} \leftarrow \sigma(\mathbf{v'}_{w_j}^T \mathbf{v}_{w_I})\mathbf{v}_{w_I}$
      $\mathbf{g}_{w_I} \leftarrow \mathbf{g}_{w_I} + \sigma(\mathbf{v'}_{w_j}^T \mathbf{v}_{w_I})\mathbf{v}_{w_j}$
      $\texttt{loss}_{w_O} \leftarrow \texttt{loss}_{w_O} - \log(-\sigma(\mathbf{v'}_{w_j}^T \mathbf{v}_{w_I}))$
    $\mathbf{v'}_{w_O} \leftarrow \mathbf{v'}_{w_O} - \eta \cdot \mathbf{g}_{w_O}$
    $\mathbf{v}_{w_I} \leftarrow \mathbf{v}_{w_I} - \eta \cdot \mathbf{g}_{w_I}^T$
    for all $w_j \in W_{neg}$ **do**
      $\mathbf{v'}_{w_j} \leftarrow \mathbf{v'}_{w_j} - \eta \cdot \mathbf{g}_{w_j}$

Metoda `train` prikazana je pseudokodom zbog preglednosti. Pseudokod je izrađen po uzoru na matematički model opisan u poglavlju 2.2. Izračun funkcije gubitka dodan je u svaku iteraciju učenja jer koristi međurezultate algoritma.

4.5. DNA_test

Izvršna datoteka *DNA_test* provjerava rezultate već naučene mreže.

Mreža se deserijalizira iz datoteke imena "final_network" te se deserijalizira objekt tipa **Network**. Izrađuju se dva izlazna vektora za slučajno odabrane k-torke iste duljine. Na 1000 primjera uspoređuje se sličnost algoritmom Needleman-Wunsch i kosinusna sličnost dvaju izlaznih vektora.

Struktura ove datoteke ugrubo je opisana jer je korisnik naučene mreže u mogućnosti sam prilagoditi kod za dobijanje rezultata.

4.6. DNA_tokenizer

Svrha razreda **Tokenizer** iz datoteke *DNA_tokenizer* je rastav genoma iz tekstualne datoteke na k-torke različitih duljina po uniformnoj razdiobi. Ovaj razred inicijalizira se prvi kod učenja.

Konstruktor prima ime tekstualne datoteke u kojoj se nalazi genom i veličinu prozora koju model zahtijeva kao parametre. Dodatno se može zadati raspon duljina k-torki koji se odabiru uniformnom razdiobom. U ovom programskom rješenju pretpostavljenje vrijednosti su `lower=3` odnosno `upper=8`.

Tablica 4.7: Članske varijable razreda **Tokenizer**

Tip	Naziv	Opis
`int`	`lower`	donja granica duljine k-torki
`int`	`upper`	gornja granica duljine k-torki
`int`	`window_size`	veličina prozora konteksta
`int`	`counter`	indeks trenutne k-torke
`file`	`f`	tekstualna datoteka genoma

Razred se sastoji od dvije metode. Metoda `string_to_kmers` je privatna i poziva ju `generate_kmers` nakon što izgradi znakovni zapis genoma.

Metode su implementirane na način da čitaju cijeli genom bez potrebe za traženjem fragmenata i prekidnih znakova zbog rada s genomom koji tih znakova nema.

- `generate_kmers()`
    - vraća listu indeksi k-torki za cijeli genom iz datoteke
- `string_to_kmers(sequence)`
    - za dobiveni string `sequence` gradi listu indeksi k-torki na temelju uniformne distribucije
    - bilježi broj pojavljivanja k-torki

4.7. DNA_train

Datoteka sa programskim kodom za pokretanje procesa učenja je *DNA_train*. Prvi dio datoteke čini inicijalizacija ili deserijalizacija objekata potrebnih da bi učenje moglo krenuti. Drugi dio čini sam proces učenja i serijalizacija završne mreže.

Hoće li se objekti stvoriti ili deserijalizirati ovisi postoje li serijalizirane datoteke u istom direktoriju gdje je programski kod, odnosno je li proces učenja već počeo i serijalizirao svoje stanje.

Parametri s kojima programski kod radi zadaju se kao globalne varijable. Nazivi globalnih varijabli i njihovo značenje je sljedeće:

Tablica 4.8: Globalne varijable u datoteci *DNA_train*

Tip	Naziv	Opis
`int`	`WINDOW_SIZE`	veličina prozora
`int`	`HIDDEN_LAYER_SIZE`	veličina skrivenog sloja
`int`	`LAST_LOADED_COUNTER`	redni broj k-torke zadnje deserijalizacije
`int`	`SAVE_SWITCH`	oznaka zadnje spremljene promjene
`string`	`SWITCH_FILENAME`	ime datoteke za spremanje oznake
`string`	`ITERATOR_FILENAME`	ime datoteke za spremanje iteratora k-torki
`string`	`NETWORK_FILENAME`	ime datoteke za spremanje mreže
`string`	`FINAL_NETWORK`	ime datoteke za spremanje naučene mreže

4.7.1. Periodičko spremanje stanja

Zbog dugotrajnog procesa učenja, ostvarena je potpora za periodično spremanje stanja. Ostvareno rješenje ne gubi podatke ako se prekine u bilo kojem trenutku procesa, čak ni u trenutku spremanja stanja. Ovakva mogućnost dobivena je izmijenjivom serijalizacijom stanja. Objekti razreda **Network** i **Iterator** serijaliziraju se u da-

toteke imena NETWORK_FILENAME s dodanim sufiksom SAVE_SWITCH odnosno ITERATOR_FILENAME s dodanim sufiksom SAVE_SWITCH. Pri svakom spremanju vrijednost SAVE_SWITCH varijable se promijeni na 0 ili 1. Serijalizira se i vrijednost same varijable SAVE_SWITCH u datoteku SWITCH_FILENAME da se pri sljedećem pokretanju programa objekti deserijaliziraju iz zadnje promijenjenih datoteka NETWORK_FILENAME i SAVE_SWITCH.

Kod ponovnog pokretanja programa, traži se postoji li serijalizirane datoteke. Ukoliko postoje, prvo se deserijalizira objekt datoteke imena SWITCH_FILENAME u varijablu SAVE_SWITCH, a zatim objekti tipa iz datoteka imena NETWORK_FILENAME s dodanim sufiksom SAVE_SWITCH odnosno ITERATOR_FILENAME s dodanim sufiksom SAVE_SWITCH.

4.7.2. Proces učenja

Nakon što se potrebni objekti stvore ili deserijaliziraju, započinje proces učenja. Pomoću objekta razreda **Iterator** dohvaća se indeks k-torke i njezin kontekst kojim mreža uči metodom train.

Listing 4.2: Programski kod procesa učenja

```
while iterator.has_next():
        input_index, output_indexes = iterator.next_kmer()
        network.train(input_index, output_indexes)

        if iterator.counter % 100 == 0:
            print "iterations:", iterator.counter, "/", len(
                iterator.kmers)
            print "loss:", network.loss
        if iterator.counter % 100000 == 0 and
        iterator.counter != LAST_LOADED_COUNTER:
            save_data(iterator, network)
```

Serijalizacija datoteka, odnosno spremanje stanja učenja odvija se svakih 100000 iteracija.

Nakon što se iteriraju sve k-torke fragmenta, neke članske varijable, kao što je unigram polje, postavljaju se na None zbog smanjivanja veličine serijalizirane datoteke naučene mreže. Naučena mreža serijalizira se u datoteku imena koje određuje varijabla FINAL_NETWORK.

Nakon završetka treninga mreže može se koristiti programski kod datoteke *DNA_test*.

4.8. DNA_unigram

Odgovornost razreda **Unigram** iz datoteke *DNA_unigram* je stvaranje i popunjavanje polja koje se koristi za odabir negativnih uzoraka kod optimizacije *Negative Sampling*.

McCormick (17.1.2017) savjetuje implementirati polje za slučajan pristup elementima vokabulara *word2vec* metode. Unigram polje sadrži približno 10 000 000 indeksa k-torki. Polje se popunjava indeksima k-torki s obzirom na njihovu distribuciju P_n. U ovoj implementaciji modela koristi se distribucija

$$P_n(w_i) = \frac{f(w_i)^3}{\sum_{j=0}^{n} \left(f(w_i)^{3/4}\right)}. \tag{4.1}$$

Količina popunjenosti indeksima pojedine k-torke u tablici dobiva se umnoškom distribucije te k-torke po izrazu 4.1 i veličine unigram polja. Neuronska mreža tijekom učenja iz unigram tablice uzima određeni broj negativnih uzoraka, a opisani način odabire uzorke u skladu sa željenom vjerojatnosti. Kako indeksi k-torki s većom vrijednosti $P_n(w)$ više popunjavaju tablicu, imaju veću vjerojatnost biti izabrani kao negativni uzorci. Članska varijabla `size` manja je od željene veličine tablice. Zaokruživa-

Tablica 4.9: Članske varijable razreda **Unigram**

Tip	Naziv	Opis
`list`	`unigram_table`	unigram tablica
`int`	`size`	realna veličina tablice

njem umnoška veličine tablice i realnog broja $P_n(w)$ na cjelobrojni tip gubi se dio informacije koji određuje stupanj popunjenosti polja. Za veliki broj `size` gubitak je zanemariv.

Metode koje razred sadrži su sljedeće:

- `count_to_probability()`
  - popunjava rječnik `kmer_distribution` razreda **Data** informacijama o disribuciji $P_n(w)$ za sve k-torke
- `create_unigram_table()`
  - gradi unigram tablicu približne veličine 10 000 000 i popunjava ju količinom indeksa k-torki po izrazu $P_n(w_i) * \texttt{size}$
- `choose_n(n, forbidden_value)`
  - iz `unigram_table` odabire skup od `n` negativnih uzoraka bez vrijednosti `forbidden_value` i vraća ga kao listu

Metode `count_to_probability` i `create_unigram_table` pozivaju se u konstruktoru a `choose_n` tijekom učenja kod svakog bloka neuronske mreže.

4.9. DNA_util

Datoteku *DNA_util* čini skup metoda koje ne ovise o prethodnim razredima, pa su odijeljene zbog preglednosti.

- `create_vocabulary()`
  - vraća dva unakrsna rječnika koji povezuju znakovne zapise k-torki s njihovim indeksima u modelu *dna2vec*
- `int_to_dna()`
  - pretvara indeks k-torke u njezin znakovni zapis
- `get_dimension()`
  - izračunava ulaznu dimenziju neuronske mreže po formuli $V = \sum_{k=3}^{8} 4^k$
- `softmax(vector)`
  - obavlja *softmax* funkciju nad vektorom `vector` po izrazu 2.3
- `softmax_list(l)`
  - mijenja vektore liste `l` s vektorima po izrazu 2.3
- `sigmoid(x)`
  - vraća vrijednost po izrazu $\sigma(x) = \frac{1}{1+\exp(-x)}$
- `random_matrix(rows, columns)`
  - vraća matricu dimenzija $rows \times columns$ popunjenu slučajnim realnih brojeva u rasponu $[0, 1]$
- `range_for_kmer_length(length)`
  - vraća granice intervala indeksa k-torki koji su duljine `length`
- `sim(v, w)`
  - vraća kosinusnu udaljenost vektora `v` i `w` po formuli $cos\_sim = \frac{v \cdot w}{|v||w|}$
- `Needleman_Wunsch_score(kmer1, kmer2)`
  - vraća mjeru sličnosti k-torki `kmer1` i `kmer2` algoritmom Needleman-Wunsch

5. Rezultati

Provjera uspješnosti rada naučenog modela zahtjeva usporedbu kosinusne udaljenosti k-torki modela *dna2vec* sa sličnostima k-torki algoritmom Needleman-Wunsch.

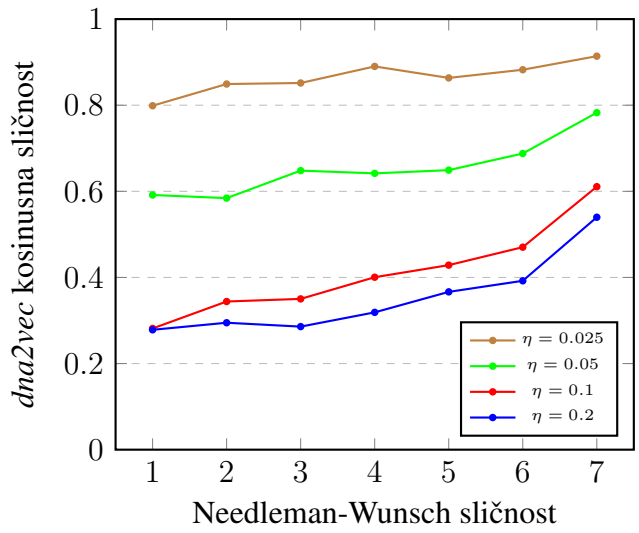
Algoritam koji se koristi nalazi se u pythonovom modulu `Bio.pairwise2` u metodi `align.globalxx`. Metoda koristi parametre `match=1`, `mismatch=0` i `gap=0`.

Rezultat algoritma Needleman-Wunsch za dvije k-torke iste duljine iznosi k ako su k-torke identične. Ako je rezultat jednak nuli, ni jedan znak se ne podudara odnosno potrebno ih je sve zamijeniti da se iz jedne k-torke dobije druga.

Ispitane su mreže raznih stopa učenja sa 8-torkama. Za mrežu s najboljom korelacijom dviju sličnosti dodatno je provjerena korelacija za sve duljine k-torki.

5.1. Rezultati po stopama učenja

Napravljene su četiri mreže učene na genomu *Escherichia coli* bakterije. Stope učenja mreža su redom: $0.025, 0.05, 0.1$ i 0.2.



Slika 5.1: Usporedba sličnosti za k-torke veličine 8 znakova

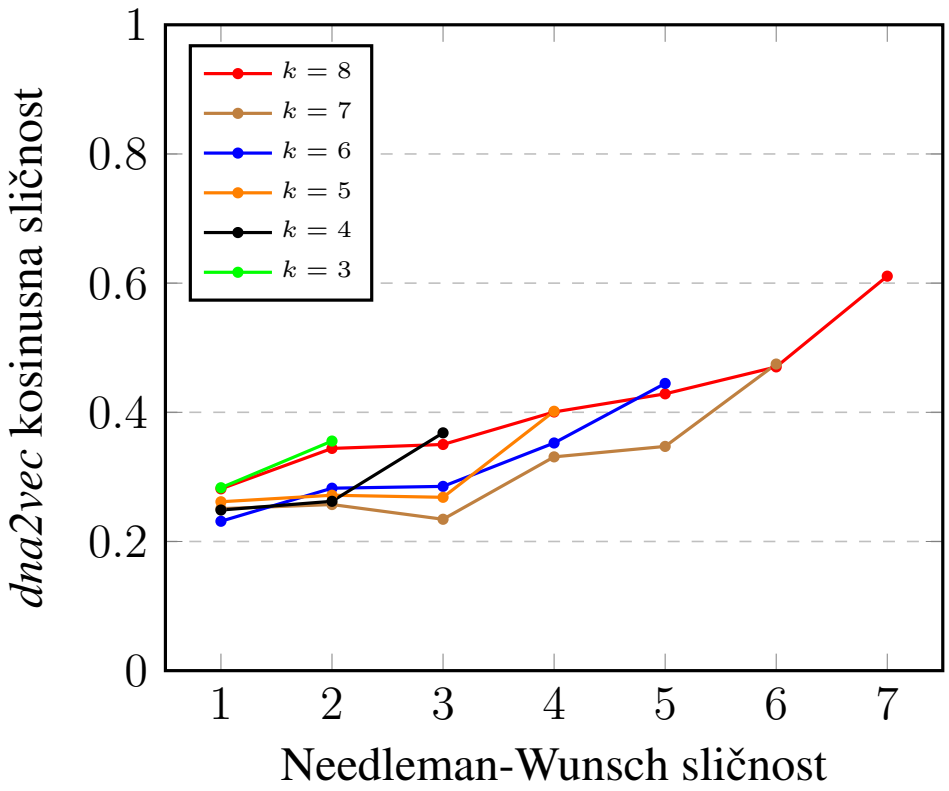
Rezultati uspješnosti za svaku mrežu izrađeni su na slučajno odabranih 50 parova k-torki veličine 8. Odabrano je 100 uzoraka za svaku vrijednost algoritma Needleman-Wunsch iz intervala $[1, 7]$. Izračunava se prosječna kosinusna sličnost za svaku vrijednost Needleman-Wunsch algoritma za par k-torki. Graf 5.1 prikazuje razinu korelacije između tih dviju sličnosti.

Mreže sa stopama učenja 0.1 i 0.2 daje najveću promjenu kosinusne sličnosi po rastu rezultatom Needleman-Wunsch algoritma. Detaljnije će biti ispitana mreža sa stopom učenja 0.1 za različite duljine k-torke.

Rezultati sa stopama učenja 0.05 i 0.025 dani su kao primjeri presporog učenja mreže. Stopa učenja premala je čak za veliki skup podataka k-torki zbog relativno rijetke pojave pojedine k-torke. U ta dva primjera kosinusna sličnost dviju k-torki je približno ista bez obzira na sličnost Needleman-Wunsch algoritmom.

5.2. Rezultati po duljini k-torki

Odabrana je mreža s stopom učenja 0.1 za ispitivanje korelacije kosinusne sličnosti sa sličnosti algoritmom Needleman-Wunsch.



Slika 5.2: Usporedba sličnosti različitih duljina k-torki za mrežu sa stopom učenja 0.1

Iz grafičkog prikaza vidljivo je da za kosinusna sličnost raste što su sličnije k-torke po referentnom algoritmu. Kako algoritam Needleman-Wunsch daje vrijednost k za pojedine k-torke ako su one identične, taj slučaj nije ispitivan. Naime, kako je izlaz neuronske mreže za identične ulaze isti, kosinusna sličnost dva ista vektora dala bi rezultat 1.0 odnosno potpuno podudaranje.

Rezultati na grafu 5.2 prikazuju nagli rast kosinusne sličnosti za vrijednost referentnog algoritma od $k - 1$ gdje je k duljina k-torke, odnosno nagli porast kosinusne udaljenost dviju k-torki ako su one sličnije po Needleman-Wunsch algoritmu.

6. Diskusija

Ogroman broj k-torki genoma *Escherichia coli* od približno 4.5 milijuna odužio je pojedino treniranje neuronske mreže na 8 sati. Neophodno je bilo implementirati programsku potporu za spremanje trenutnih vrijednosti matrica neuronskih mreža u datoteke. Osim toga, konstantno zauzeće svih 8 jezgri procesora onemogućilo je paralelno korištenje računala za druge svrhe.

Unatoč tome, improvizacijom i eksperimentiranjem s raznim stopama učenja te korištenjem modula *numpy* koji implementira višedretvenost, učenje mreže dalo je zadovoljavajuće rezultate.

Glavni nedostatak brzini izvođenja je korištenje vlastite implementacije neuronske mreže *Skip-Gram* u objektnom jeziku. Prema autoru metode *dna2vec* trajanje treninga na cjelokupnom genomu čovjeka u profesionalnom alatu *gensim* traje 72 sata.

Profesionalni alati imaju promijenjivu stopu učenja koja prilagođavanjem vrijednosti poboljšava učenje. Obično se kao jednostavna funkcija promijene stope učenja koristi hiperbola oblika $\eta(t) = \eta(0)/(1 + t/T)$. Osim toga, optimizacijske metode koje profesionalni alati koriste su brže te uključuju višedretvenost i implementirani su u nižim programskim jezicima, što ih dodatno ubrzava.

Algoritam Needleman-Wunsch ima vremensku složenost $O(n^2)$ jer koristi matricu kao spremnik međurezultata.

Kako je ulazna dimenzija neuronske mreže $V = \sum_{k=3}^{8} 4^k = 87360$ jedinica, a izlazna višestruko veća, dobijanje izlaznog vektora metodom *dna2vec* vremenske je složenosti $O(VN)$ zbog samog matričnog množenja unutar metode `result` razreda **Network**. Ostale operacije zanemarive su zbog povećeg broja dimenzija matrica u neuronskoj mreži.

Za dobijanje kosinusne sličnosti dvije k-torke potrebno je dobiti dva izlazna vektora iz neuronske mreže čija složenost je $O(VN)$ te dodatno izračunati kosinusnu sličnost dobijenih vektora sa složenošću $O(V)$. Za k-~~tokre~~ do veličina 8 znakova algoritam Needleman-Wunsch je brži od *dna2vec* metode.

Međutim, izlazni vektori za svaku k-torku mogu se pohraniti. S obzirom na veličinu konteksta od 10 okolnih k-torki koji se koriste u ovom radu, količina memorije potrebna za spremanje izlaznih vektora svih k-torki veličina $[3, 8]$ iznosi :

$$n \cdot V \cdot \texttt{context\_size} * 4B = 305270784000B \approx 284GB$$

gdje je n broj k-torki za koje se spremaju izlazni vektori.

U originalnom radu *dna2vec* korišteno je 20 okolnih k-torki za treniranje mreže. Zbog toga svaki izlazni vektor neuronske mreže sadrži više podataka od onih dobijenih u ovome radu. Osim toga, veličina skrivenog sloja postavljena je na 5 jedinica, što je mnogostruko manje od veličine ulaznog vektora mreže.

Ako se za svaki izlazni vektor dodatno spremi norma vektora koja zauzima samo 4 bajta, izračunavanje kosinusne sličnosti dvije k-torke metodom `dna2vec` dobilo bi vremensku složenost $O(V)$.

7. Zaključak

Temeljem dobivene značajne povezanosti između sličnosti znakovnog zapisa k-torki algoritmom Needleman-Wunsch i kosinusne udaljenosti vektorske reprezentacije k-torki metodom *dna2vec*, zaključuje se da testirana metoda efikasno gradi višedimenzionalni vektorski prostor s mogućnošću usporedbe sličnosti k-torki.

Korelacija s dobro poznatim algoritmom daje alternativni način usporedbe k-torki. Postoji i niz drugih mogućnosti rada s modelom *dna2vec* kao što su konkatenacija k-torki pomoću linearne arimetike s izlaznim vektorima. Te operacije nisu ispitane u ovom radu, ali svejedno imaju bogatu mogućnost ~~bogate~~ primjene.

Implementirana izvedba je sporija od algoritma Needleman-Wunsch, međutim, dovoljnom količinom računalnih resursa može se višestruko ubrzati. Vremenska složenost velika je zbog potrebe da se izlazni vektor izračunava tzv. *forward propagation* metodom koja množi matrice velikih dimenzija. Spremanjem izlaznih vektora za sve k-torke nema potrebe za njihovim ponovnim izračunavanje, već treba dobiti samo kosinusnu udaljenost vektora učitanih iz datoteka.

LITERATURA

Z. Harris. Distributional structure. *Word*, 10:146–162, 1954.

D. Hartl i E. Jones. *Genetics: Analysis of genes and genomes*. Jones & Bartlett, New York, 6 izdanju, 2005.

C. McCormick. Word2vec tutorial part 2 - negative sampling, 17.1.2017. URL `http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/`. Pristup: 15.3.2017.

B. Needleman i D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 1970.

P Ng. dna2vec: Consistent vector representations of variable-length k-mers. *CoRR*, abs/1701.06279, 2017. URL `https://arxiv.org/abs/1701.06279`.

X. Rong. word2vec parameter learning explained. *CoRR*, abs/1411.2738, 2014. URL `http://arxiv.org/abs/1411.2738`.

dna2vec: vektorska reprezentacija k-torki različite duljine

Sažetak

Usporedba dijelova genoma obavlja se prilagođavanjem metode *word2vec* radu s k-torkama. Ispitivana metoda *dna2vec* ima omogućuje rad s varijabilnom duljinom k-torki u itervalu $[3, 8]$. Treniranje izvedbom *Skip-Gram* koja u obzir uzima kontekst genoma oko k-torke, dobija se pozicija k-torke u višedimenzionalnom vektorskom prostoru. Kosinusna udaljenost dvije vektorske reprezentacije k-torki daje mjeru njihove sličnosti. Ispitana je i dokazana korelacija kosinusne udaljenosti metodom *dna2vec* i rezultata algoritma Needleman-Wunsch za par k-torki. Dodatno su prodiskutirani razlozi dugotrajnog učenja te su predložena ubrzanja učenja te poboljšanje efikasnosi ispitanog modela.

Ključne riječi: dna2vec,k-torke,sličnost,strojno,učenje,neuronska,mreža

dna2vec: vector representation of variable length k-mers

Abstract

Adjustment of known method *word2vec* is used for comparing parts of genome. Innovation of method *dna2vec* is possibility to work with variable length k-mers. Training of *Skip-Gram*, the *word2vec* derivation, which takes context of k-mer in genome, results with multidimensional representation of k-mers. Cosine distance of two k-mer vector representations gives similarity measure. In this paper, correlation between *dna2vec* cosine distance and ~~algorithm~~ Needleman-Wunsch was tested and proven. Additionally, reasons behind long training were discussed and possible solutions for faster learning and better efficiency were submitted.

Keywords: dna2vec,k-mers,similarity,neural~~,~~network,machine~~,~~learning