PROJEKTNA DOKUMENTACIJA

Umjetna inteligencija u adaptivnom učenju

Jelena Nemčić, Marin Ovčariček, Zvonimir Sučić, Ivana Žeger
Voditelj:

Zagreb, rujan 2020.

SADRŽAJ

1. Uvod

Adaptivno učenje u e-learning softverima tipično funkcionira tako da mjeri razinu znanja korisnika kroz početni skup pitanja, virtualnu simulaciju i/ili dodijeljene zadatke. Na temelju podataka prikupljenih iz odgovora korisnika, takav softver u stvarnom vremenu procjenjuje razliku između korisničkog znanja i znanja potrebnih za određenu kompetenciju, te odabire lekcije i zadatke za korisnika tako da minimizira količinu edukacijskog sadržaja koji tom korisniku prikazuje.

Konstrukcija algoritma za određivanje puta za adaptivno učenje tipično se radi na dva načina: 1) kreiranjem formalnog modela znanja za određenu domenu, a koji kreiraju eksperti iz te domene ili 2) koristeći algoritamski pristup baziran na teorijama Bayesian Knowledge Tracing i Item Response Theory koji na temelju odgovora polaznika edukacije procjenjuje vjerojatnost da je polaznik usvojio određenu vještinu/koncept (što ponovno zahtijeva unaprijed definirane vještine/koncepte).

U posljednje vrijeme, a s obzirom na dostupnost sve većih količina podataka (big data) ponovno su oživjele i dodatno se razvijaju tehnologije kreiranja grafova znanja (npr. pomoću dubokog učenja), a koji s obzirom na to da su kreirani statistički mogu biti mnogo kompleksniji i uže segmentirani (precizniji) u odnosu na one koje kreiraju eksperti nekog područja. Dodatno, prikupljanje velike količine informacija u različitim domenama omogućuje da kreiranje grafova znanja ne bude ograničeno samo na one tvrtke koje imaju golem broj korisnika kao što su Google ili Facebook. Na temelju inicijalnog istraživanja vjerujemo da se ova metoda može primijeniti i na kreiranje grafova znanja za adaptivno učenje, te time omogućiti s jedne strane znatno veću adaptivnost, a s druge strane veću jednostavnost kreiranja takvih grafova.

Ovo je posebno važno za područja edukacije izvan formalnog obrazovanja, gdje nisu strogo definirane ishodi učenja i testovi kojima se mjeri je li neki ishod učenja dostignut kod pojedinog polaznika. Dva primjera za to su instrukcije, gdje učeniku često nedostaju i predznanja iz drugih područja koje je ranije u školi trebao usvojiti, te korporativne edukacije, koje često obuhvaćaju ljude različitih struka i različitim znanjima

iz domene za koju nastoje dobiti certifikat.

2. Cilj i postupak

Stoga je cilj ovog projekta provjeriti sljedeće:

- Provjera mogućnosti kreiranja grafa znanja isključivo na temelju točnosti odgovora na zadacima iz jedne domene znanja koje korisnici daju i informacije o redoslijedu zadavanja zadataka pojedinom korisniku
- Ako je navedeno moguće, potrebno je provjeriti može li se kreirati graf znanja na temelju rješavanja zadataka za istu domenu na temelju parcijalnog broja zadataka (što realnije reprezentira dostupne zadatke za stvarne domene – rijetko su dostupna baš sva znanja iz neke domene da bi se mogla kreirati zadaci koji pokrivaju baš svaku informaciju u toj domeni)
- Ako je navedeno moguće, potrebno je provjeriti može li se isto napraviti i za neku domenu realnog znanja

Ukratko, ovim se projektom provjerava može li se graf znanja potreban za adaptivno učenje kreirati metodama dubokog učenja na temelju ponašanja korisnika na zadacima (probabilistički), a bez potrebe za time da eksperti unaprijed određuju koncepte/vještine u koje se grupiraju zadaci ili čak sam graf znanja.

Za potrebe ove provjere kreirat će se:

- umjetni, zatvoreni graf znanja
- zadaci koji pokrivaju sve informacije prisutne u tom zatvorenom grafu znanja

Potom će se zadaci dati testerima na rješavanje, tako da se:

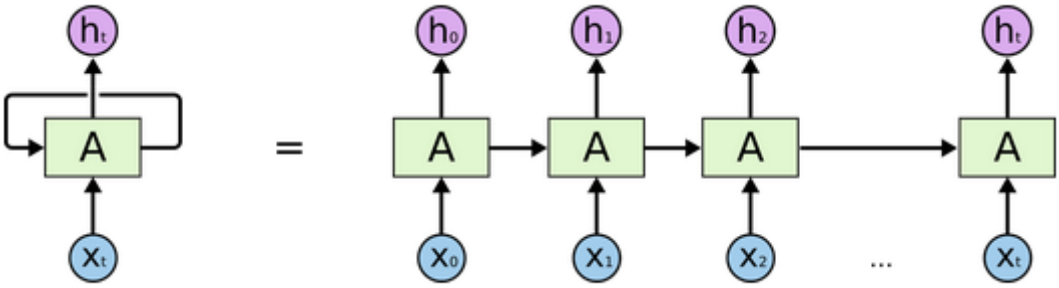
- varira redoslijed zadataka koje pojedini tester dobiva kako bi pokrio sve kombinacije
- bilježi točnost odgovora testera na zadatak
- u slučaju netočnog odgovora testeru se prikazuje točna informacija.

3. Postojeće ideje i koncepti

Istraživanjem fraza kao što su Bayesian Knowledge Tracing (BKT), Item Response Theory (IRT), Deep Knowledge Tracing (DKT), Intelligent Tutoring Systems (ITS), Massive Open Online Courses (MOOC), itd. pronađeni su mnogi radovi koji su služili kao sredstvo upoznavanja s tematikom rada i inpiracija za daljnja ostvarenja.

Među prvima pronađena je i proučena platforma adaptivnog učenja Knewton korištena za personalizaciju edukacijskog sadržaja (1).

Izazovnim područjem praćenja znanja (engl. *knowledge tracing*) moguće je strojno modelirati znanje korisnika pomoću njegove interakcije s računalom tijekom procesa učenja (2). Korisnicima je predložen sadržaj učenja ovisno o njihovim potrebama (sadržaj je okarakteriziran kao prelagan, pretežak, moguće ga je potpuno preskočiti ili ostaviti za kasnije). Predviđaju se buduće korisničke performanse prema prošloj aktivnosti. Mnoge metode pri tome uključuju korištenje Markovljevih modela s ograničenom funkcionalnošću. Također, neki modeli koriste logističku regresiju uz PFA (engl. *performance factors analysis*). U novije vrijeme, korištene su povratne neuronske mreže (engl. *recurrent neural networks, RNN*) pa cjelokupni model nosi naziv Deep Knowledge Tracing (DKT).
RNN-ovi su popularni jer imaju petlje i omogućuju opstanak informacije, za razliku od uobičajnih neuronskih mreža. Moguće ih je protumačiti kao mnogostruke kopije iste mreže od kojih svaka predaje poruku idućoj, kao što je vidljivo na slici 3.1.



Slika 3.1: Intuitivni prikaz povratne neuronske mreže (RNN)

Posebice je popularna kompleksnija varijanta RNN-a, LSTM (engl. *long short-term memory*). Interakcije (pitanje-odgovor) potrebno je pretvarati u vektore fiksne duljine (ideja je inpute predstaviti one-hot encodingom parova (oznaka zadatka, točnost)). Mapiranje u izlazne vrijednosti postiže se računom slijeda "skrivenih" stanja (uzastopnim enkodiranjem bitnih informacija proteklih zapažanja kako bi odgovarale budućim predikcijama). Izlazna vrijednost je vektor vjerojatnosti točnog rješavanja svakog od zadataka u modelu. Sposobnost predviđanja korisničkih performansi ispitana je na simuliranom skupu podataka. Umjetno su generirani korisnici koji rješavaju određen broj zadataka iz fiksnog skupa koncepata. Svaki korisnik ima latentnu "vještinu" za svaki koncept, a svaki zadatak ima koncept i težinu. Vjerojatnosti da korisnik točno riješi zadatak određene težine ako ima određenu vještinu modelirane su pomoću IRT-a. Također, otkriven je javno dostupan benchmark skup podataka, 2009-10 Assistments Data.

Ponađen je i proučen i sustav KnowEdu, kojem je cilj konstruirati grafove znanja za edukacijske potrebe i identificirati odnose između različitih koncepata (3).
Na početku se pokušavaju ustanoviti svi koncepti koji se nalaze u dostupnim nastavnim materijalima i tečajevima korištenjem strojnog učenja. Zatim se koriste CRF (engl. *conditional random field*) model, povratna neuronska mreža i varijanta LSTM mreže, GRU (engl. *gated recurrent units*) kako bi se dobio graf preduvjeta. Skup podataka prikupljen je iz više testova pri čemu je svaki ispitanik riješio svaki test. Jedan test sastoji se od više pitanja koja ispituju znanje istog koncepta i pomoću njega se izračuna odgovarajuća ocjena znanja tog koncepta za svakog ispitanika. Ocjene predstavljaju vještine ispitanika u tom području i one su ulaz modela. Svaki model kao izlaz daje, za svaku kombinaciju koncepata, vjerojatnost da je koncept A preduvjet za znanje koncepta B.

4. Bayesian Knowledge tracing (BKT)

Bayesian Knowledge Tracing je metoda praćenja razine znanja korisnika koja uzima u obzir vjerojatnosti pogađanja i slučajnog pogrešnog odgovora, također je osjetljiva i na redoslijed točnih/netočnih odgovora.

4.1. Općenito o BKT

Bayesian Knowledge Tracing koristi Hidden Markov Model i ima 4 osnovna parametra:

- p(L0) - vjerojatnost da je korisnik a priori savladao gradivo
- p(G) - vjerojatnost da je korisnik pogodio točan odgovor bez da ima potrebno znanje
- p(S) - vjerojatnost da je korisnik krivo odgovorio iako ima potrebno znanje
- p(T) - vjerojatnost da je znanje prešlo iz NE ZNA u ZNA nakon prilike da se primjeni znanje

Kao izlaz dobivaju se vrijednosti:

- p(L) - vjerojatnost ovladavanja vještinom (eng. probability of skill mastery)
- p(C) - vjerojatnost da će korisnik ispravno primijeniti vještinu u budućnosti (eng. probability of the student correctly applying the skill on a future practice)

$$p(L_t \mid obs = correct) = \frac{p(L_t) * (1 - p(S))}{p(L_t) * (1 - p(S) + (1 - p(L_t) * p(G))} \tag{4.1}$$

$$p(L_t \mid obs = wrong) = \frac{p(L_t) * p(S)}{p(L_t) * p(S) + (1 - p(L_t) * (1 - p(G))} \tag{4.2}$$

$$p(L_{t+1}) = p(L_t \mid obs = correct) + (1 - p(L_t \mid obs = correct)) * p(T) \tag{4.3}$$

$$p(C_{t+1}) = p(L_{t+1}) * (1 - p(S)) + p(L_{t+1}) * p(G) \tag{4.4}$$

4.2. Ideje

Prvobitna ideja je bila da se p(L0) računa iz inicijalnih pitanja, vrijednosti p(G) i p(S) bi se prema preporuci iz rada(trebalo bi pronaći kojeg i baciti referencu) stavile na interval [0,0.3], [0,0.1] te bi se p(T) postavio prema preporuci eksperta što ne želimo jer je cilj ovog projekta da smanjimo zadatke eksperata na minimum.
To je ukazalo na potrebu pronalaska algoritama koji bi uz pomoć nekog skupa podataka aproksimirali parametre za BKT.

4.3. Problemi i zadaci

- proučiti parametar p(T)
- proučiti kodove sa githuba kako bi se dobila ideja kako algoritam funkcionira
- napraviti malu implementaciju s malo pitanja i provjeriti radi li
- proučiti parameter fitting uz pomoć EM algoritma, stohastic gradient descenta ili neke druge metode
- kako napraviti input dataset, prikupiti podatke

4.4. Dobivanje BKT parametara

4.4.1. EM (expectation-maximization) algoritam

- iterativni algoritam za pronalaženje (aproksimiranje) najveće izglednosti (eng. maximum likelihood) ili maksimalne a posteriori (MAP) procjene parametara u statističkim modelima
- model ovisi o nepoznatim latentnim varijablama
- EM iteracija sadrži 2 koraka:

- korak očekivanja (E), koji stvara funkciju za očekivanje log-izglednosti koja se procjenjuje pomoću trenutne procjene parametara, procjenjuju se vrijednosti latentnih varijabli
- korak maksimizacije (M), koji izračunava parametre distribucije koji maksimiziraju očekivanu log-izglednost pronađenu u E koraku, ti se parametri zatim koriste za procjenu latentnih varijabli u sljedećem E koraku

– primjenjuje se kada želimo odrediti parametre distribucije (normalna, eksponencijalna, . . .)

– problem: za korištenje potrebo znati distribuciju podataka ili točne vrijednosti (eng. true values) traženih parametara

Kroz ovo istraživanje nije pronađena niti jedna implementacija EM algoritma za aproksimaciju BKT parametara niti je napravljena vlastiti implementacija zbog prevelikog praga znanja matematike.

4.4.2. Grid search i Simulated Annealing

Pronađen je kod napisan u Javi koji računa BKT parametre tehnikom simuliranog kaljenja `https://github.com/wlmiller/BKTSimulatedAnnealing`. U README na githubu se također spominjao kod koji je bio baza za to, on je koristio običan grid search kako bi izračunao parametre. Oba koda su prevedena u python i prilagođena našim skupovima podataka. Na kraju se ispostavilo da je "simulirano kaljenje" povoljnije te se grid search odbacio.

4.5. Rezultati

– napravljen google forms kviz sa 20 pitanja iz biologije, ispitanici moraju odgovoriti na svih 20 pitanja kako bi podaci ušli u dataset

– napravljena python skripta koja pretvara podatke dobivene iz google formsa u oblik prikladan za treniranje BKT-a i pronalaženje parametara

– pronađen je kod u Javi koji tehnikom simuliranog kaljenja aproksimira parametre za BKT uz pomoć danog dataseta, kod je preveden u python skript

– napravljena python skripta za BKT koja određuje vjerojatnost da je ispitanik naučio/ savladao gradivo

– uz pomoć skripte za aproksimaciju BKT parametara, nađene su njihove vrijednosti za svaku vještinu iz ASSISTMENTS dataseta i pohranjenje u google sheets tablicu

– dobiveni parametri algoritmom simuliranog kaljenja uspoređeni su s onima dobivenima pomoću grid search metode -> vrijednosti parametara su skoro iste, vrlo male razlike

– napravljen google forms kviz sa po 6 pitanja iz 5 koncepata, izračunati su parametri za taj dataset

– BKT kod i kod za aproksimaciju BKT parametara su se dalje koristili u bilježnicama za izgradnju grafa probabilističkim metodama

4.6. Poveznice

4.6.1. BKT

https://en.wikipedia.org/wiki/Bayesian_Knowledge_Tracing
http://www.cs.cmu.edu/~./ggordon/yudelson-koedinger-gordon-individualized-bayesian-knowledge-tracing.pdf
https://github.com/CAHLR/pyBKT/blob/master/README.md
https://www.learnlab.org/uploads/mypslc/publications/bca2008v.pdf
https://www.upenn.edu/learninganalytics/ryanbaker/paper_143.pdf
https://github.com/yemao616/Bayesian-Knowledge-Tracing
http://www.cs.cmu.edu/~./ggordon/yudelson-koedinger-gordon-individualized-bayesian-knowledge-tracing.pdf
https://www.fi.muni.cz/~xpelanek/publications/umuai-overview.pdf
https://medium.com/@joyboseroy/modelling-a-students-learning-34375b0131dd
https://www.math.vu.nl/~sbhulai/publications/data_analytics2018c.pdf

4.6.2. Pronalaženje parametara

https://www.fmrib.ox.ac.uk/datasets/techrep/tr00yz1/tr00yz1/node9.html
https://github.com/wlmiller/BKTSimulatedAnnealing/blob/master/computeKTparams_SA.java
https://www.upenn.edu/learninganalytics/ryanbaker/paper_143.pdf
https://educationaldatamining.org/files/conferences/EDM2018/papers/EDM2018_paper_14.pdf
http://yudelson.info/hmm-scalable/
https://www.educationaldatamining.org/EDM2015/proceedings/short364-367.pdf
https://concord.org/wp-content/uploads/2016/12/pdf/tracking-student-progress-in-a-game-like-learning-environment.pdf
https://tinyheero.github.io/2016/01/03/gmm-em.html
https://machinelearningmastery.com/expectation-maximization-em-algorithm/
http://rstudio-pubs-static.s3.amazonaws.com/1001_3177e85f5e4840be840c84452780db52.html
https://www.colorado.edu/amath/sites/default/files/attached-files/em_algorithm.pdf

5. Bayes graf

5.1. Uvod

Ideja je napraviti vlastiti model koji iz skupa podataka računa vjerojatnost savladavanja koncepata te se prema njima gradi graf znanja.
Prvi pristup je račun uvjetnih vjerojatnosti $p(Y_j \mid X_i)$, odnosno vjerojatnost da korisnik zna (savladao je) koncept Y ako zna X. Koristi se klasična formula Bayesove uvjetne vjerojatnosti čime se gradi matrica odnosa koncepata.

$$p(Y \mid X) = \frac{p(X \wedge Y)}{p(X)} \tag{5.1}$$

Vjerojatnost Y uz X nam govori koliko je poznavanje X bitno za poznavanje Y. Pretpostavka je da će korisnik s većom vjerojatnošću znati jednostavniji koncept ako zna kompliciraniji (nadogradnju jednostavnog), ako su X uz Y i Y uz X podjednaki (potreban je neki prag) može se pretpostaviti da su koncepti nezavisni ili bi se mogli spojiti kao jedan koncept.
Drugi pristup je bio pokušaj da se gleda kad su koncepti X i Y točni s obzirom koji od njih je položen prvi, pretpostavka je bila da bi se tako mogla odrediti relacija nadogradnje, odnosno prethodnika. Pristup nije uspio jer ili nije moguć ili nije bio dobro definiran / implementiran.
Najveći problem u oba pristupa je to što pokušavamo dobiti ovisnosti među konceptima,a ne pitanjima. Za razliku od pitanja za koncepte se ne može reći da ih se zna/ne zna jer se oni sastoje od više pitanja te se može samo gledati postotak riješenosti za svakog studenta / prosječan postotak riješenosti. Prosječan postotak rješenosti se može gledati kao pripadnost neizrazitom skupu ZNA odnosno 1- postotak riješenosti kao pripadnost skupu NE ZNA, to komplicira stvari kod Bayesovog zaključka. Potrebno je pronaći/proučiti postoji li ekvivalent Bayesovog zaključka kada se koriste neizraziti skupovi odnosno kontinuirane vrijednosti [0,1].
Ideja:

- iz dataseta izračunati parametre BKT-a algoritmom po izboru
- uz pomoć BKT-a odrediti je li neki student savladao gradivo na temelju odgovora na pitanja (vratiti se u diskretno područje)
- na temelju tih rezultata raditi matricu uvjetnih vjerojatnosti

Problemi:

- Je li dobro korisiti isti dataset za aproksimaciju BKT parametara i onda nad tim istim datasetom uz pomoć BKT-a određivati je li neki student savladao određeno gradivo?
- Koliko bi trebali biti veliki datasetovi za aproksimaciju i izgradnju grafa kako bi se dobili neki smisleni rezultati?

Pretpostavke:

- svi studenti odgovaraju na sva pitanja iz svih koncepata
- svi studenti moraju istim redoslijedom odgovarati na pitanja
- budući da se koriste metode s vjerojatnostima (Bayes) potrebna je veća količina podataka kako bi rezultati imali smisla

5.2. Problem kontinuiranih vrijednosti

Običan Bayesov zaključan radi sa diskretnih vrijednostima (true/false, 1/0). Dok je to istina kod izračuna vjerojatnosti i točnosti pitanja, isto se ne može reći za koncepte. Može se gledati da je koncept položen s nekom vjerojatnošću ili da određen pokušaj ispita pripada skupo "položen" s nekom pripadnošću, to je obično postotak točnih bodova i to je racionalan broj. Iako postoji Bayesov zaključak za kontinuirane vrijednosti nije jednostavan za izračunati jer se teško mogu dobiti latentne razdiobe bodova za neku populaciju. Pošto je Bayes kontinuiranoj domeni prekompliciran za računati potražena je altrenativa te se došlo do zaključka da se treba nekako vratiti u diskretnu domenu, odnosno napraviti model koji može odlučiti kad neki koncept je ili nije položen.
Najjednostavnije rješenje je ručno za svaki koncept zadati bodovni prag te bi svi korisnici koji zadovolje prag imali oznaku da su savladali koncept. To i nije najbolji pristup jer se ne uzimaju u obzir razne druge varijable kao što su npr. vjerojatnost da korisnik pogađa ili slučajno pogrješi.

Kako bi se rješio taj problem, pronađena je tehnika Bayesian Knowledge Tracing 4 i kombinirana sa vlastitom tehnikom raspodjele po Gaussu kako bi se bolje aproksimiralo trenutno znanje korisnika.

5.3. Skaliranje korisnika prema Gaussu

Kao alternativa BKT-u implementirano je skaliranje studenata prema Gaussu:

- za svakog studenta i za svaku vještinu izračunat je percentil kojemu taj student pripada u ovisnosti o odgovorima svih studenata
- postavi se threshold koji odgovara percentilu iznad kojeg se nalaze studenti koji su položili vještinu

Paralelno se za svakog studenta računa prolaznost (0 ili 1) prema BKT-u i prema Gaussu:

- ako se one poklapaju, to se uzima kao vrijednost za tog studenta
- ako je zbroj te dvije jačine veći od 0, uzima se da je student položio vještinu
- ako su različite gleda se jačina s kojom model tvrdi da je student pao/položio, ona se računa na sljedeći način:

```
bkt_certainty = (bkt_pL-bkt_threshold) / (1-bkt_threshold) if bkt_pass == 1
                            else -1*(bkt_threshold-bkt_pL)/(bkt_threshold)
gauss_certainty = (gauss_percentile-gauss_threshold) / (1-gauss_threshold) if gauss_pass == 1
                            else -1*(gauss_threshold-gauss_percentile)/(gauss_threshold)
```

Slika 5.1

5.4. Izvještaji

5.4.1. 22.07.2020.

Planovi, problemi:

- pronaći dobru vrijednost za cutoff kod micanja poveznica kod grafa
- iteriranje po retcima dataframeova - nije dobro, treba pronaći efikasnije načine obrade tablica
- isprobati program na malom datasetu

– isprobati program na biologija dataset, ali gledajući da je svako pitanje jedan koncept (napraviti i program koji bi pretvorio pitanja u koncepte)

– proučiti librarye za crtanje usmjerenih grafova (boje, labele, debljine poveznica)

– dopuniti program da crta usmjerene grafove

– pokušati naći implementacije EM i grid search algoritma za računanje BKT parametara

– pokušati pronaći dobro objašnjenog bayesa u kontinuiranoj domeni

– pronaći naprednije implementacije BKT-a koje uzimaju više parametara

– osmisliti strukture podataka koje bi predstavljale čvorove i povezivale pitanja,koncepte i ostale čvorove u jednu cjelinu, omogućavale obilazak grafa te nalaženje najkraćeg puta (po nekom kriteriju) od početnog do ciljnog čvora

– napraviti novi test koji bi imao mali broj pitanja iz različitih područja koja su međusobno povezana (npr. stanica, što je to -> građa stanice -> građa različitih organela) i na tome isprobati program

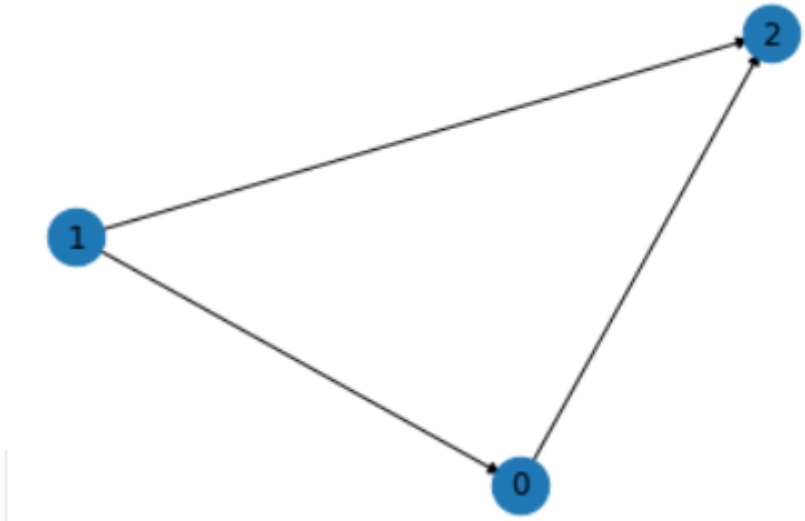
Rezultati

– napravljena skripta za izgradnju grafa pomoću uvjetnih vjerojatnosti i Bayesove formule

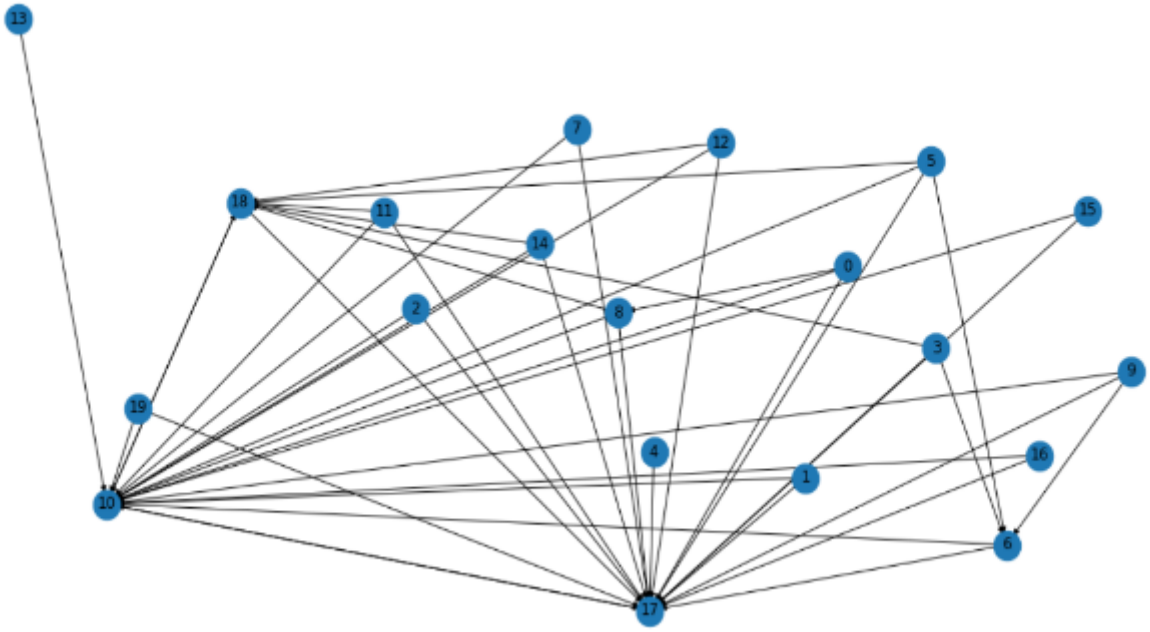
  - ulaz: datoteka koja sadrži BKT parametre za sve vještine i datoteka koja sadrži tablicu studenata i njihovih odgovora
  - izlaz: matrica koja predstavlja matricu susjedstva grafa, gdje svaki element predstavlja "jačinu"/"vjerojatnost" s kojom je koncept *i* povezan s konceptom *j*
  - skripta sadrži razred BKT i funkcije za računanje vjerojatnosti p(X), računanje zajedničke vjerojatnosti p(X and Y) i računanje matrice ovisnosti

– skripta je isprobana na dosad prikupljenim podacima testa za biologiju (74 odgovora) i dobiveni su realni rezultati, ali nismo imali sa čime usporediti budući da je to samo jedan koncept

– izrađeni su grafovi na malom umjetnom datasetu i datasetu biologija gdje su se pitanja gledali kao zasebni koncepti



Slika 5.2



Slika 5.3

– za potrebu prilagodbe pitanja u koncept napravljena je skripta koja uzima pitanja kao koncepte, računa vjerojatnosti te crta graf

– za crtanje grafova koristio se netowrkx library

– vrijednost cutoffa za izgradnju grafova bi prema procjeni trebala u intervalu [0.8,0.95]. Nismo našli nikakve službene izvore/preporuke tako da će se za sada cutoff morati ručno procijeniti.

– odrađena optimizacija koda gdje se boljim iteriranjem po dataframeu dobiva ljepši i brži kod, daljnja optimizacija je vjerojatno moguća, ali trenutno nije prioritet trošiti vrijeme na istraživanje kako to napraviti

5.4.2. 28.07.2020.

– napravljen je je novi kviz sa pet koncepata iz biologije

– svaki koncept ima 6 pitanja

– problemi: kviz ima više pitanja nego prošli, u prosjeku je mentalno zahtjevniji, znanje koncepta vjerojatno nije dobro pokriveno sa tih 6 pitanja, neki koncepti su su značajno zahtjevniji nego drugi te raspodjela pitanja po težini nije ista u svakom konceptu i vjerojatno nije optimalna budući da se trenutno svi odgovori vrednuju jednako

– u trenutku pisanja izvještaja imamo samo 25 odgovora na kviz, pokrenuli smo model da vidimo kako reagira na manjak podataka

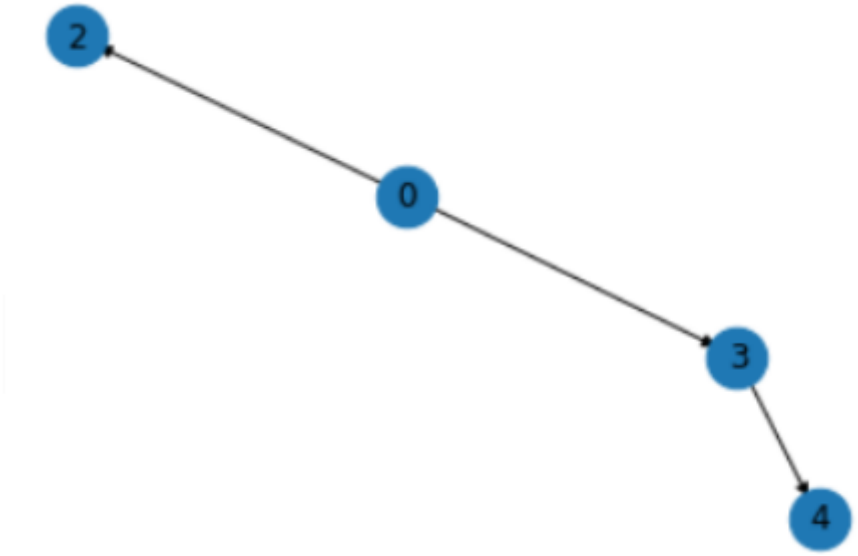
– rezultati su očekivano lošiji nego prije, u početku su od 5 koncepata na grafu bila prikazana samo 3 te smo dobili NaN vrijednosti matrici povezanosti, što se dogodilo jer prema pragu BKT-a niti jedan student nije položio predmet te se pojavilo dijeljenje sa nulom. Nakon što smo spustili prag prolaska sa 0.95 na 0.9, pojavio se dodatan čvor na grafu i nismo imali NaN, ovo ukazuje na problem osjetljivosti BKT-a na manjak podataka i na redoslijed točno/netočno odgovora studenata

– također same povezanosti grafa nisu imali smisla (npr. da je živčana stanica preduvjet za diobu stanice)

– ovi rezultati nisu iznenađujući jer je samo svojstvo BKT-a da je osjetljiv na redoslijed i uzastopno ponavljanje točno/netočno u nizu odgovora te su modeli bazirani na vjerojatnostima jako osjetljivi na manjak ulaznih podataka

– IDEJA: isprobati jednostavno skaliranje bodova studenata na temelju Gaussove raspodjele te odrediti prag prema kojem bi se dijelili na prolazak/pad. Ovako bi se maknula loša svojstva BKT-a, ali bi model postao još jednostavniji te bi vjerojatno imao slabija svojstva predviđanja.

– treba proučiti kako napraviti da linije povezanosti imaju različite debljine s obzirom na jačinu povezanosti, onda bi se mogao malo spustiti cutoff da se dobije bolji pregled ovisnosti između koncepata



Slika 5.4

Legenda:

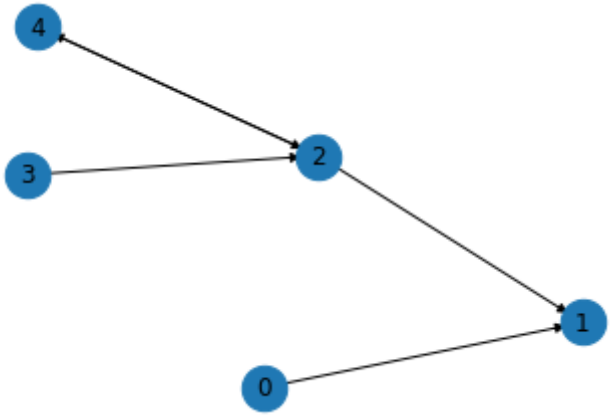
0: Stanica 1: DNA 2: Stanični metabolizam 3: Dioba stanice 4: Živčana stanica

5.4.3. 29.07.2020.

- napravljen program koji generira proizvoljan umjetan dataset
- obavezni ulazni argumenti su broj studenata i trojka (broj pitanja po konceptu, prosječni udio točnih odgovora u tom konceptu, standardna devijacija)
- skripta pretpostavlja da svaki student odgovara na sve koncepte i na sva pitanja
- prema gaussovoj raspodjeli svakog koncepta se svakom studentu pridjeljuje postotak točno riješenih zadataka te mu se prema toma generiranju rješenja
- dataset se sprema u klasičnom obliku kojeg naše skripte za crtanje mogu koristiti
- u trenutku pisanja ima 75 odgovora na novu anketu
- pokrenuta je obrada podataka i izračunavanje bkt parametara za trenutne odgovore
- cutoff threshold je 0.5

Legenda:

0 : Stanica 1 : DNA 2 : Živčana stanica 3 : Stanični metabolizam 4 : Dioba stanice



Slika 5.5

Zaključci: ljudi koji su znali diobu stanicu i stanični metabolizem generalno su znali i živčanu stanicu, ljudi koji su znali diobu stanice obično su znali i živčanu stanicu te ljudi koji su znali DNA su obično znali i stanicu/živčanu stanicu. Ovo ne prikazuje realan slijed učenja biologije, ali nije ni očekivano da prikazuje jer biologija nema tako strogi uvjetni slijed koncepata kao npr. matematika

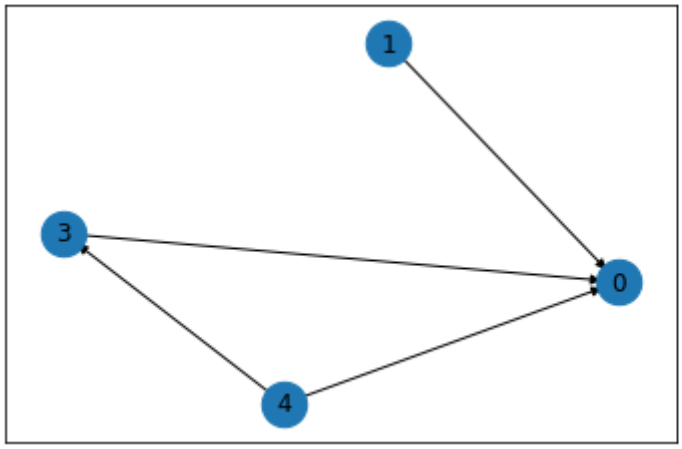
Rezultati umjetnih datasetova

BKT-annealing daje predzadnjem konceptu jako male vrijednosti parametara zbog čega nitko ne prolazi taj predmet, dobivaju se NaN vrijednosti te se ne crta taj čvor. Ovo ukazuje na veliku osjetljivost aproksimacije parametara na broj i redoslijed točnih odgovora studenata, ovo se može poboljšati pažljivim odabirom mean i stddev vrijednosti kod generiranja umjetnog dataseta (ne odabrati premali mean te pogotovo ne uzimati veliki stddev kod malog meana i uz mali broj studenata jer se može dogoditi da većina njih dobije male postotke riješenih zadataka što daje loše bkt parametre). Drugi način je staviti nekakvo zaglađivanje kako bi se onemogućilo dobivanje NaN vrijednosti s nekom malom konstantom (ovo samo sprječava errore, ali ne pomaže u točnosti modela). Zadnje što ostaje je pronaći i razviti alternativu BKT-u koja nije toliko osjetljiva.

5.4.4. 30.07.2020.

– ideja: trenutno je za model potrebno da svi studenti rješavaju sve koncepte, to nije pojava u realnim skupovima podataka pa se predlaže sljedeće rješenje - kod računa joint_probabilitya uzeti presjek studenata koji su rješavali koncept A i koncept B po njihovom student_id-u, sam izračun pX se ostavi kao što je i bio.

```
[[1.         0.41176471 0.         0.41176471 0.29411765]
 [0.77777778 1.         0.         0.22222222 0.11111111]
 [       nan        nan        nan        nan        nan]
 [0.77777778 0.22222222 0.         1.         0.33333333]
 [1.         0.2        0.         0.6        1.        ]]
[0, 1, 2, 3, 4]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.p
  if sys.path[0] == '':
```



Slika 5.6

Ova pretpostavka bi mogla vrijediti jer ako je realan skup podataka dovoljno velik te ako je presjek također dovoljno velik oni će imati dovoljno informacije da se da realni prikaz ovisnosti između koncepata

– ideja2: iako je bitno da studenti odgovaraju istim redoslijedom na pitanja taj uvjet se može izbjeći naknadnim sortiranjem po quesiton_id-u. Ovo nije prioritet za implementirati jer gotovo svi standardizirani ispiti daju pitanja istim redoslijedom te daju svakom studentu ista pitanja (treći uvjet za funkcioniranje modela)

– ideja3: napraviti skriptu iz koja bi bila skup svi do sad proizvedenih skripti. Ta skripta bi zapravo bila skup "main" funkcija dobivenih pozivanjem određenih funkcija iz ostalih bilježnica (ovdje se radi importanje, a ne kopiranje postojećeg koda)

– Mogućnosti sjedinjene skripte:

  - Izgradnja grafa
  - Generiranje umjetnog dataseta
  - BKT simulated annealing
  - Obrada podataka s google formsa

- Gledanje pitanja kao koncepata

5.4.5. 31.07.2020.

- napravljene su potrebne izmjene skripti kako bi se broj studenata na pojedinim vještinama mogao razlikovati, što više odgovara realnim podacima (neće svaki student odgovarati na pitanja iz svake vještine)
- kod računanja vrijednosti joint_probability uzima se presjek studenata koji su rješavali koncept A i koncept B (gleda se njihov student_id) i računa njihova uspješnost
- kod generiranja umjetnog dataseta dodan je parametar broj_studenata za svaki koncept
- napravljena je skripta koja sjedinjuje sve važne dosad napravljene skripte u niz funkcija koje se lako pozivaju kako bi se moglo raditi importanje .ipynb fileova potrebno je u istom kazalu gdje su fileovi dodati praznu __init__.py datoteku
- dodana je mogućnost odabira pragova cutoffa, gaussa, bkt-a kod pozivanja funkcije build_graph().
- trenutna greška u skripti pitanja->koncepti, izračun pX-a se pokreće dva puta, prvi put je sve u redu dok se drugi put neki keyevi ne nalazi u dictionaryu, uzrok tome je drugi poziv funkcije calculate, kod splitanja se maknu neki indeksi, greška je u pisanju koda, svugdje se gleda range(noStudents) umjesto list studenata, kod kfolda se određeni studenti miču i ostaju samo neki id-ovi i noStudents se smanjuje i može se desiti npr. da se makne student s id-om 1 što znači da ga više nema u dictionaryju ali for petlja će ga i dalje pokušati dohvatiti, treba napisati program da radi prema student id-u, ako se ne nađe bolje rješenje maknut će se k-fold izračun jer trentna implementacija nije dobra - POPRAVLJENO
- treba u oba crtanja grafa riješiti problem dijeljenja sa nulom - DJELOMIČNO POPRAVLJENO - radi, ali rješenje nije najbolje (ako pX bude nula, ne dijeli se nego se joint probability postavlja na 0)
- taj kod bi se možda mogao koristiti kao provjera povezanosti pitanja
- zadnje što preostaje nakon popravljanja i uljepšavanja koda koji pretvara pitanja u koncepte jest smisliti obradu google formsa tako da je omogućeno da bude

različit broj pitanja po konceptu, s tim da su pitanja grupirana odnosno nisu pomiješana po ispitu

5.4.6. 14.08.2020.

Prebačene su skripte iz colaba u .py fileove na git, obavaljeno refaktoriranje kako bi se malo više standardizirale konvencije imenovanja i ostala uljepšavanja koda.Napravljena main skripta koju se lagano može programirati da radi razne operacije uz pomoć funkcija koje pozivaju importane module.

5.5. Poveznice

https://ocw.mit.edu/courses/mathematics/18-05-introduction-to-probability-and-statistics-spring-2014/readings/MIT18_05S14_Reading13a.pdf
https://www.probabilitycourse.com/chapter9/9_1_2_MAP_estimation.php
https://www.sciencedirect.com/science/article/abs/pii/S0951832017300674
https://arxiv.org/pdf/1610.09156.pdf
http://www.dia.fi.upm.es/~mgremesal/MIR/slides/Lesson%206%20(Inference%20from%20Conditional%20Fuzzy%20Propositions).pdf
https://www.intechopen.com/books/fuzzy-logic/some-methods-of-fuzzy-conditional-inference-for-application-to-fuzzy-control-systems

Nakon početnog grupnog istraživanja grafova znanja i termina BKT, DKT, IRT i sl., dio grupe se odvojio na dubinsko istraživanje BKT-a, a dio na proučavanje koncepata stvaranja grafa znanja. U kasnijoj fazi, nakon pronalaska i refaktoriranja obećavajućeg modela preporuka zadataka pod nazivom ExRec, definiran je i zadatak vizualnog grupiranja zadataka koji pripadaju različitim konceptima. Među zadacima stvaranja grafa i grupiranja primijećene su velike korelacije; njihovo nadovezivanje i upotpunjavanje.

6. Stvaranje grafa

6.1. Početna istraživanja

Većina prvih proučenih radova i modela je napuštena zbog nepoklapanja s glavnom idejom našeg projekta, prevelike matematičke kompliciranosti ili kompleknosti računarske izvedbe te zbog namjernog ili nenamjernog uskraćivanja informacija o pozadini funkcionalnosti u objavljenim radovima.

6.1.1. Obrada prirodnog jezika

Početna istraživanja mogućnosti stvaranja grafa odvela su nas u smjeru metoda s korištenjem NLP-a (engl. *natural language processing*). Iako je smjer ocijenjen kao prezahtjevan za potrebe projekta, proučene su osnovne funkcionalnosti Python biblioteka koje olakšavaju takve postupke - spaCy, networkX, seaborn (4). NetworkX je kasnije korišten za vizualizaciju grafova u BKT-u, kao u i Lentil pokušajima grupiranja zadataka po konceptima.

6.1.2. K12EduKG

Za edukacijske aplikacije pronađen je K12EduKG, sustav automatske konstrukcije grafa znanja gdje čvorovi i veze predstavljaju međusobno povezane koncepte (5), no

nejasan je način pretakanja te ideje u kod koji bi odgovarao našem problemu. Autori daju premalo iskoristive informacije. Odlučeno je nastaviti istraživanje u drugačijim smjerovima.

Novi smjerovi istraživanja od tog trenutka uključuju Deep Generative Models i GraphRNN. Proučeni su općeniti koncepti metoda nepovezani s edukacijskim aplikacijama kako bi poslužili kao inspiracija za nadogradnju na naš problem.

6.1.3. Deep Generative Models

Model je temeljen na klasi modela grafičkih neuronskih mreža (engl. *graph nets*). Uči reprezentaciju grafova; čvorova i veza prema propagaciji informacije. Sekvencijalno generira nove strukture (čvor ili vezu). Generiranje je slijed odluka o dodavanju gradivnih dijelova strukture predstavljen vjerojatnostima u zasebnim modulima:

- dodati čvor ili ne,
- dodati vezu ili ne,
- izabrati jedan čvor da se spoji s nekim novim.

Drugačiji poredak struktura označava različite odluke. Za graf se koristi tzv. vektor ugradnje čvorova (engl. *node embedding vector*). Računaju se "skrivena stanja" iz ulaza čvorova i propagiraju se grafom za dobivanje informacije lokalnog susjedstva. Vektor poruke računa se za svaku vezu (pomoću potpuno povezane neuronske mreže, GRU ili LSTM) pa svaki čvor dobiva tu informaciju i osvježava svoj prikaz (6). Spominje se i mogućnost uključivanja uvjetne informacije za proces generiranja. Kao i u prethodnom primjeru, autori ne objavljuju sve potrebne informacije. Oblik vektora ostaje nepoznat, kao i način izračuna "skrivenih stanja" za vektor ugradnje.

6.1.4. Graph RNN

Autoregresivni model. Autoregresivni modeli su sekvencijalni modeli, ali i s unaprijednom propagacijom; generativni, ali i pod nadzorom. Imaju velik potencijal kao alternativa povratnim neuronskim mrežama i GAN-ovima, generativnim suparničkim mrežama (engl. *generative adversarial newtorks*) za obavljanje generiranja (7). Izlaz

su im prediktivne uvjetne vjerojatnosti $P(x_{t+1}|x_1, \ldots, x_t)$. Uvjetovanje je moguće samo na podatcima (ne npr. na šumu kao kod GAN-ova). Konkretno, kod Graph RNN dijelovi matrice susjedstva generiraju se sekvencijalno (npr. jedan po jedan stupac) pomoću RNN. Daljnjim proučavanjem prepoznati su mnogi nedostaci ovog modela. Složenost je O(N^2), gdje je N broj čvorova. Nadalje, zbog sekvencionalnosti, dva bliska čvora grafa mogu biti jako udaljeni u procesu generiranja u RNN što otežava performanse. Prilično je bitno osigurati invarijantnost na permutacije u čvorovima zbog izračuna vjerodostojnosti. Jedan od nedostataka pristupa svakako je i korištenje BFS algoritma (engl. *breadth-first search*) za redanje čvorova u grafu; vrlo efikasnog računski, ali neoptimalnog.

6.2. Graph Recurrent Attention Networks

6.2.1. Općenito o GRAN-ovima

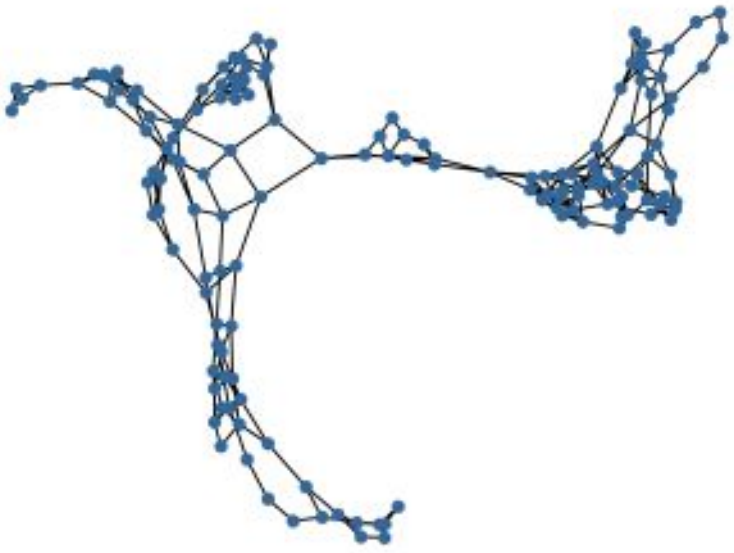
Istraživanjem rada (8) otkriveno je da gradi graf blok po blok. Mijenjanjem veličine blokova moguće je balansirati između kvalitete i efikasnosti. Blokovi predstavljaju određen broj redaka (stupaca) u matrici susjedstva (zadano je 2). Koriste se GNN koje bolje shvaćaju autoregresivnu povezanost (u odnosu na RNN) već generiranih dijelova grafa i onih koje još treba generirati.
Izlazna distribucija parametrizirana je korištenjem Bernoullijevih mješavina (korelacije generiranih veza unutar bloka).
Obećavajuća prednost ovog modela je što rješava neke probleme spomenutih GraphRNN. Složenost je manja, konkretno $O(N)$. Isto tako, GNN bolje razumije topologiju grafa; odluke o generiranju trenutnog bloka donosi izravno ovisno o strukturi grafa, ne koristi gore problematična "skrivena stanja" i efektnije modelira kompleksnost redanja čvorova.
Kao mogući problem prepoznata je općenita evaluacija generativnih modela.

6.2.2. Tijek rješavanja

Početni problem lokalne instalacije PyTorcha i potrebe za instalacijom kontradiktornih *requirementsa* riješen je prelaskom na Google Colab. Originalni proučeni program generira grafove proteina. Čvovori predstavljaju aminokiseline koje ih izgrađuju. Spajanjem čvorova vidljivo je kako se aminokiseline povezuju. Ideja je bila preinači originalni kod kako bi čvorovi predstavljali zadatke, a veze definirale povezanost zadataka unutar jednog koncepta.

6.2.3. Rezultati

Prilagođavanjem broja čvorova i epoha treniranja, popravljena je testna greška .pth datoteke koja sadrži informaciju o konfiguraciji. Vizualiziran je prikaz grafa proteina, što je vidljivo na slici 6.1:



Slika 6.1: Graf proteina

Nije odgovarala činjenica da je prikaz molekula proteina zapravo neusmjereni graf pa je matrica susjedstva čvorova simetrična, što je upola manje računskog posla od onog koji bi bio potreban za naš projekt jer se uzima samo donji trokut u izračunima. Cilj našeg projekta je dobiti usmjereni graf. U radu se spominje kako je moguće napraviti preinake matrice, izračunati i gornji trokut. Također, zbog specifične forme

dataseta, prilagodba na vlastiti dataset bila bi mukotrpna. Od pristupa se odustalo jer su pronađeni oni koji više obećavaju i kojima je potrebno manje prilagodbe.

6.3. Latent Skill Embedding (Lentil)

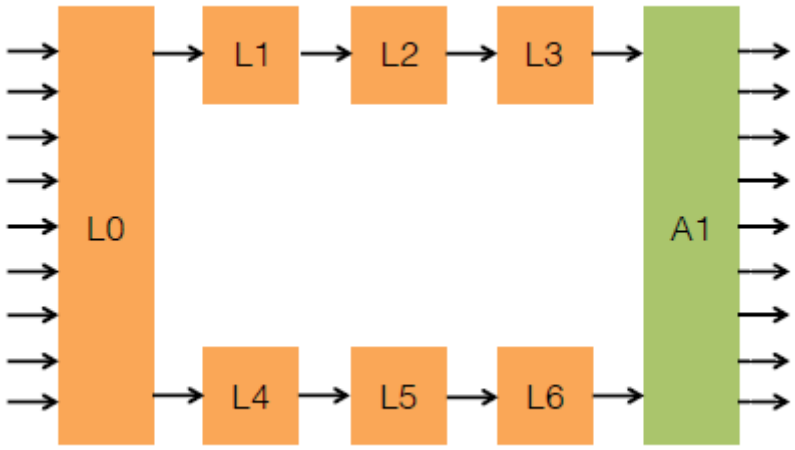
6.3.1. Općenito o Lentil-u

Pronađen je i istražen gotovo na samom početku prakse. Daljnjim istraživanjem pokazuje velik potencijal (9). Sažeto ga je moguće definirati kao sustav za personaliziranu preporuku nastavnih cjelina. Metoda je upravljana podacima, uči reprezentaciju sadržaja koja ne zahtijeva *a priori* znanje preslikavanja *content-to-concept*. Proširuje ideje *sparse factor analysis* (SPARFA) i multidimenzionalne Item Response Theory.
Model uči prikaz korisničkog znanja, kao i edukacijski sadržaj kako bi se prikazale preporuke, personalizirane upute o učenju. Probabilistički je model. Problem je formuliran kao regularizirani *maximum-likelihood embedding* korisnika, nastavnih cjelina (lekcija) i procjena znanja (ispita) u zajednički semantički prostor (latentni prostor vještina) iz skupa podataka o prošlosti interakcije korisnik-sadržaj (*access traces*). Nastavne cjeline i procjene znanja su moduli sadržaja. Fiksni su, a korisnici imaju putanje latentnim prostorom vještina. Stvara se multidimenzionalno okružje - studentsko znanje "leži" u kontinuiranom prostoru stanja, a preduvjeti lekcija moduliraju dobitke znanja lekcijskih modula. Ugradnja u taj prostor ne koristi simetrične udaljenosti komponenata, već se bilježi prirodni napredak težine procjene znanja i rasta korisničkog znanja.
Korisnik je prikazan kao skup latentnih vještina, nastavna cjelina kao vektor dobitaka vještina i skupa preduvjeta, a procjena znanja kao skup potrebnih vještina. Korisniku se procjenjuje znanje nekog modula (ne zna ili zna, 0 ili 1), a vjerojatnost da će proći je veća ako korisnik ima visoku razinu vještine koja nadmašuje potrebne vještine za procjenu. Korisnik može poboljšati vještinu vremenom (koje je diskretizirano). Naglašava se da bi za poboljšanje vještine trebalo gledati korisničko predznanje (ovisno o predznanju podešavati težine u jednadžbi modela).

6.3.2. Ideje

U radu je istražena i implementirana ideja određivanja slijeda lekcija pri praćenju znanja što se ispostavilo kao veoma primjenjivo na naš problem. Temelj eksperimenata s ovim modelom bila je vizualizacija slijeda zadataka. U kasnijoj etapi projekta, po-

vratkom na ovaj obećavajući model, u originalni se kod implementiralo grupiranje zadataka kada su poznate pripadnosti koncepata i zadataka. Velika početna prednost modela je što koristi pronađeni Assistments dataset koji je prvi pronađen u sklopu projekta i koji služi kao glavni temelj za usporedbu. Razmišljalo se na koji način je prigodnom veličinom dataseta moguće osigurati dovoljnu varijabilnost puteva učenja. U originalnim .ipynb bilježnicama primijećene su mnoge instance korisničkih putanja koje dijele iste lekcijske module na početku i module procjene na kraju, ali se sastoje od različitih lekcija tijekom učenja → stvaraju se mjehurići, *bubbles*, koji su zapravo eksperimentalni dokaz dobrih svojstava različitih pristupa učenju. Prikazani su na slici 6.2. Mjehurići se koriste za evaluaciju sposobnosti ugradnje preporuke slijeda lekcija koja vodi do uspješnog ostvarenja ciljeva učenja. Model logističke regresije s L2 regularizacijom korišten za procjenu vjerojatnosti da će korisnik slijediti preporučenu "granu" mjehurića. Unutar svakog mjehurića, korisnici koji su krenuli preporučenim putem spojeni su s najbližim susjedom iz grupe onih koji nisu slijedili preporučeni put prema razlici u *propensity scoreu*.



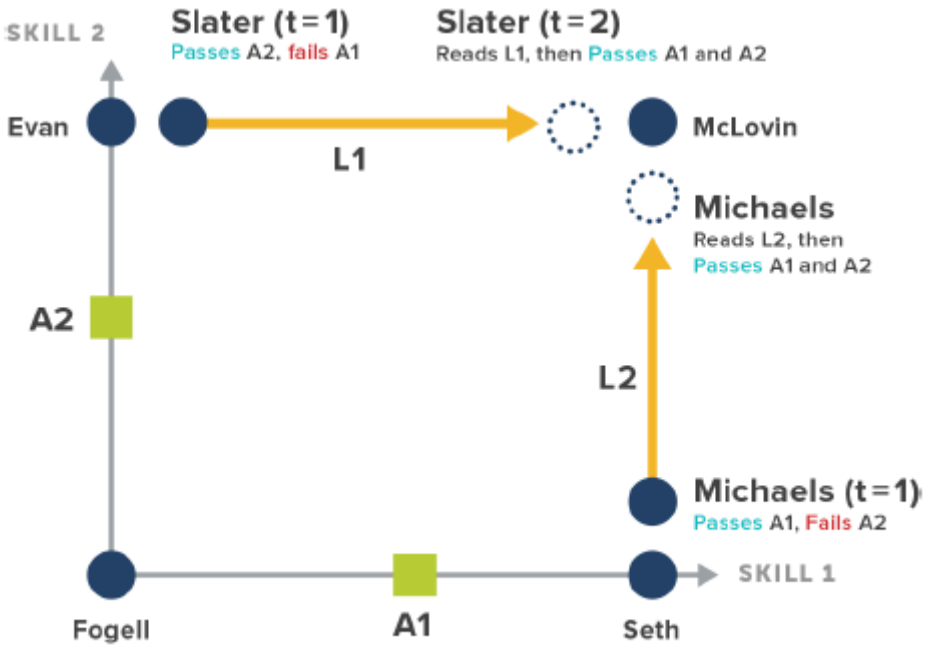
Slika 6.2: Prikaz mjehurića, različitih putanja tijekom učenja (9)

6.3.3. Problemi i zadaci

Kao početni potencijalni problem izvedbe modela pokazala se činjenica da su ocjene znanja modelirane na različitim osima grafičkih prikaza što za sobom povlači neovisnost lekcija i vještina, što ne mora u stvarnosti biti slučaj. Također, u prikazu putanja, u originalnim ispitnim podacima u radu vidljiva je pristranost korisničkih putanja jednoj vrsti temeljne, česte putanje zbog sličnosti pozadinskog znanja ispitanika.

Isto tako, pronađena je greška u originalnom kodu; činjenica da se prijeđene putanje korisnika ne broje na odgovarajući način (iteriranje liste tupleova bezuspješno, uvijek 0).

Pri početnom ostvarivanju vizualizacije slijeda zadataka kao problem je okarakterizirana činjenica da crta vektore (vidljivo na slici 6.3), a ne pravi graf koji "povezuje" korisnike preko procjena ovisno o vještinama dobivenih nakon neke lekcije (više ukazuje na smjer potencijalnog puta, što isto donekle odgovara cilju projekta).



Slika 6.3: Grafički prikaz Lentil-a (korisnika, lekcija, testova i napretka ovisno o točnosti) (9)

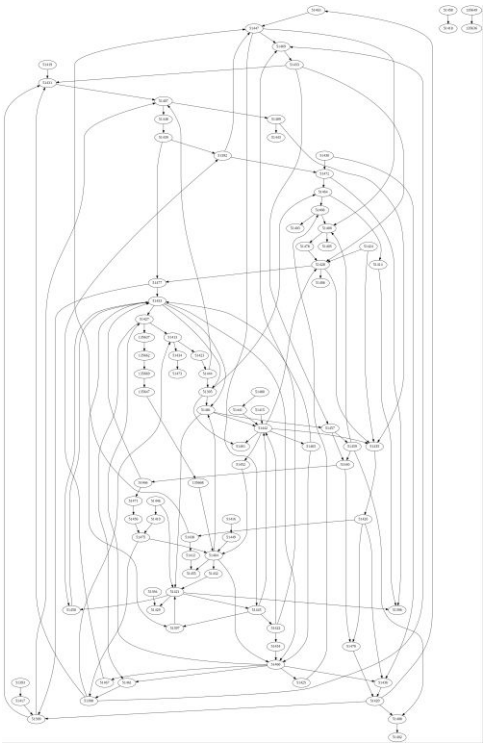
Izrađena je prilagodba na Google Colab, koja se pokazala izazovnijom od očekivanog zbog zastarjelosti originalnog koda i činjenice da je Google Colab službeno ukinuo podršku za Python 2.7 (iako je ipak i dalje moguće pokretati sadržaj).

Nakon uspješne vizualizacije slijeda rješavanja zadataka Assistments dataseta, ideja je bila pokušati prilagoditi model našem "Biologija" datasetu. Potrebno je bilo riješiti problem prisutnosti varijabli trajanja i vremenskih koraka u originalnim funkcijama jer mi to ne gledamo i ne trebamo te spajati čvorove zadataka ovisno o pripadnosti pojedinom konceptu ili još bolje, spajati koncepte.

6.3.4. Rezultati

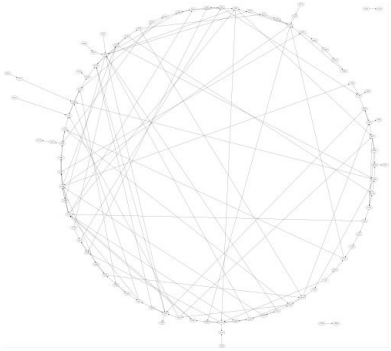
Nakon početnog ispravljanja grešaka u kodu, dobiven je string koji označuje povezanosti čvorova dataseta na način da su strelicom povezani zadaci koje jedan korisnik rješava jedan za drugim prema pojavi u datasetu. Vizualizacija toga na cijelom datasetu generirala se preko 8 sati jer PyGraphviz, Pythonovo sučelje Graphviza (alata za crtanje grafova), ne iskorištava mogućnosti ubrazanja u Google Colabu. Odlučeno je prilagoditi i smanjiti dataset (uzeti prvih 200 redaka). Nakon toga uspješno je izvršena vizualizacija dvije vrste grafova koji prikazuju povezanost zadataka:

- graf tijeka (engl. *flow grah*) - zadaci Assistment dataseta predstavljaju čvorove grafa koji se povezuju ovisno o korisničkom pristupu konkretnom problemu. Prikaz je vidljiv na slici 6.4.
- graf veza (engl. *connection grah*) - čvorovi su i korisnici i zadaci kojima oni pristupaju. Prikaz je vidljiv na slikama 6.7 i 6.8.

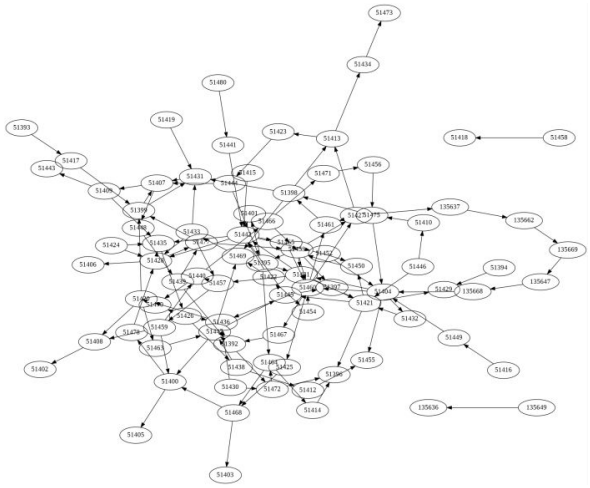


Slika 6.4: Originalni "dot" graf tijeka s podacima smanjenog Assistment dataseta

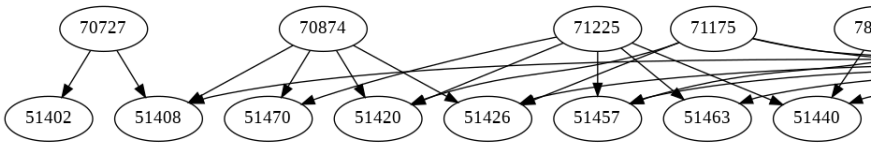
Eksperimentiranjem s PyGraphvizom moguće je dobiti različite vizualne prikaze istog grafa (odnosno rasporede - dot, neato, twopi, circo, fdp, sfdp), pa je ovisno o namjeni moguće izabrati najprikladniji. Neki od prikaza vidljivi su na slikama 6.5 i 6.6.



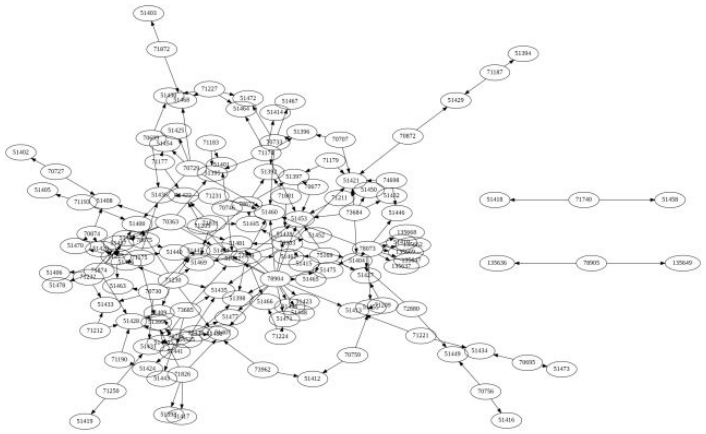
Slika 6.5: Originalni "circo" graf tijeka s podacima smanjenog Assistment dataseta



Slika 6.6: Originalni "sfdp" graf tijeka s podacima smanjenog Assistment dataseta



Slika 6.7: Originalni "dot" graf veza s podacima smanjenog Assistment dataseta

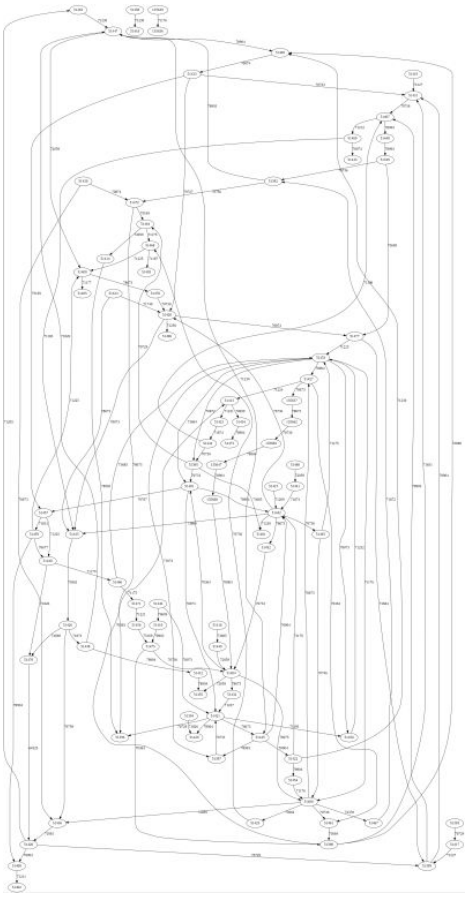


Slika 6.8: Originalni "sfdp" graf veza s podacima smanjenog Assistment dataseta

Nakon ovakvog prikaza moguće je bilo zaključiti prednosti i nedostatke pojedine vrste prikaza.

Graf veza mnogo je manje intuitivan i manje pregledan od grafa tijeka. Za daljnje eksperimentiranje, kao prikladniji graf uzet je graf tijeka zbog veće preglednosti i veće prilagođenosti našem problemu. No, iako je preglednost znatno bolja nego kod grafa veza, svejedno nije u potpunosti zadovoljen kriterij preglednosti. Zadaci različitih korisnika nisu obojeni različitim bojama, a ni pristup samim zadacima nije kronološki.

Za intuitivniju vizualizaciju putanja dodane su oznake pojedinih korisnika između čvorova, što je vidljivo na slici 6.9. Ideja je poboljšati dodavanjem različitih boja svakom korisniku.



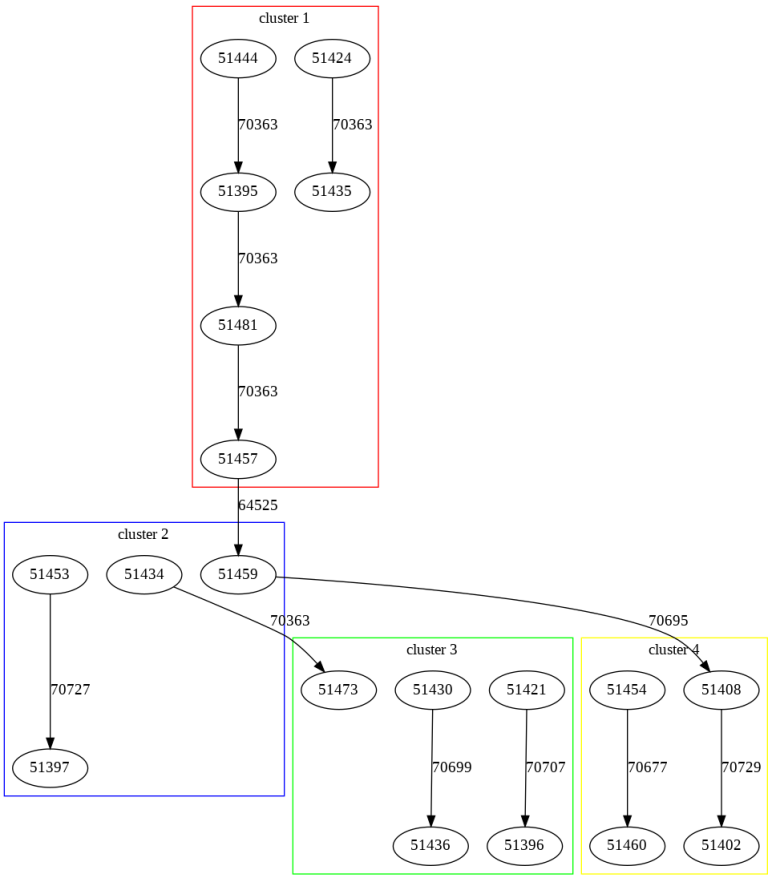
Slika 6.9: "dot" graf tijeka s podacima smanjenog Assistment dataseta uz dodane oznake korisnika

7. Clustering

Početno se razmatralo na osnovu čega bismo u modelu preporuka mogli napraviti ostvarenje vizualizacije grupiranja (engl. *clustering*) zadataka u koncepte ako nemamo označenu eksplicitnu pripadnost. Kao moguća ideje spominjao se redoslijed rješavanja, no zbog nerealnosti takve situacije najbrže je odbačena. Dodatno, moglo bi se gledati točnost pojedinih odgovora velikog broja ispitanika.

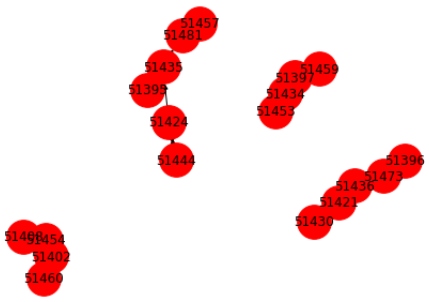
7.1. Latent Skill Embedding (Lentil)

Nakon kratkotrajnog napuštanja Lentil modela zbog bavljenja modelima koji više obećavaju, napravljen je povratak u sklopu istraživanja za zadatak grupiranja zadataka ako nemamo eksplicitno definiranu pripadnost konceptima.
Prvo je istražena mogućnost vizualnog grupiranja zadataka u okviru Lentil koda ako imamo pripadnost konceptima. Takva vizualizacija služila bi za usporedbu točnosti s modelom koji bismo osmislili. To je uspješno izvršeno. Dodan je stupac informacije o konceptima dataseta Assisstments u kod. Smanjenom datasetu za probu (20 pitanja) dodijeljene su različite oznake koncepata te je izvršeno grupiranje zadataka po konceptima izmjenom funkcije koja stvara graf tijeka manipulacijama s Pandasom (čvorovi su zadaci, na vezama oznake korisnika koje su ih riješile). Vizualizacija se nalazi na slici 7.1.



Slika 7.1: Grupiranje zadataka smanjenog Assistments dataseta prema pripadnosti konceptima

Isti rezultat dobiven eksperimentom s drugačijim pristupom (izračunom matrice prijelaza između stanja (čvorova) Markovljevog modela). Grupiranje zadataka je identično kao na slici 7.1, ali je prikaz manje interpretabilan jer u ovoj fazi ne prikazuje oznake korisnika. Vidljiv je na slici 7.2.



Slika 7.2: Grupiranje zadataka smanjenog Assistments dataseta prema pripadnosti konceptima izračunom matrice prijelaza

Konkretno, koristi se provjera različitosti pojedinih pitanja. Svaki zadatak teorijski je modeliran kao stanje Markovljevog procesa (diskretni stohastički proces je Markovljev proces ako vjerojatnost prijelaza u određenom diskretnom trenutku ovisi o stanjima u svim trenutcima prije). Stvara se matrica prijelaza između stanja (čvorova koji predstavljaju pitanja). Na osnovi takve matrice crta se graf. Kasnije se računa matrica vjerojatnosti prijelaza stanja pa stacionarna distribucija (vjerojatnosna distribucija koja se ne mijenja u Markovljevom lancu kako vrijeme prolazi).
Ponovno čitanje rada (9) inspiriralo je za ideju korištenja izračuna entropije za određivanje koncepata. Entropija se računa kao negativni skalarni produkt stacionarne distribucije i $Plog(P)$, gdje P predstavlja matricu vjerojatnosti prijelaza među stanjima. Sama korist izračuna entropije još nije prikazana. Entropija inače predstavlja "porast informacije u podatcima". Potencijalno, ideja je da, s obzirom da se računa za svaku matricu vjerojatnosti prijelaza, za svaku putanju različitih korisnika računati entropiju i utvrditi postoje li pravilnosti koje pomažu odrediti koncepte.

7.2. K-Medoids

Kao odvojeni pristup, napravljena je skripta koja radi clustering pomoću algoritma K-Medoids.
Kao vektor, odnosno element dataseta koji treba grupirati, uzeta je lista odgovora svih ispitanika na određeno pitanje (jedan element predstavlja jedno pitanje). Za metriku udaljenosti napravljena je funkcija koja gleda sumu razlika dva vektora (gleda se razlika između $x[i]$ i $y[i]$ za $i \in [0, brojispitanika - 1]$). Može se uzeti i kvadratni korijen te sume.
Definirana je i funkcija koja računa grešku kao sumu elemenata koji nisu dobro grupirani. Uzeto je da je "prava" klasa ona kojoj pripada najviše elemenata iz tog koncepta, a oni koji nisu u njoj, pridodani su ukupnoj sumi greške. Grupiranje se radi na standardiziranim i nestandardiziranim podacima i za obje verzije računaju se *rand index*, *adjusted rand index*, NMI vrijednost (*normalized mutual information*) te ranije spomenuta funkcija koja računa broj krivo grupiranih primjera.
Napravljena je i funkcija koja metodom lakta određuje idealan broj klasa, u slučaju da on nije otprije poznat.

Za različite datasetove skripta pokazuje funkcionalnost na različite načine.
Biologija dataset:
Za sad je algoritam isproban samo na datasetu Biologija (84 odgovora) jer je Assist-

ments malo prevelik, ali sljedeći korak je testiranje i na Assistments.
Dataset ima 30 pitanja, znači imamo 30 ulaznih vektora. Algoritam je isproban na standandiziranim i nestandardiziranim podacima, te daje nešto malo bolje rezultate na nestandandiziranim, ali to može biti i do dataseta. Rand index na nestandardiziranim podacima je 0.83, a adjusted rand index 0.41, za što kažu da spada u "srednje dobar rezultat". NMI vrijednost je oko 0.58, a minimum koji se dobiva funkcijom za sumu greške jest 9 (od ukupno 30 pitanja). Lošiji rezultati mogu biti i do dataseta Biologija koji je svakako daleko od idealnog, jer ni koncepti nisu dobro definirani (npr. ispitanik možda ne zna živčanu stanicu, ali je čuo odgovor na par pitanja iz psihologije). Korištenjem metode lakta za određivanje broja klasa dobije se 6 klasa, što je vidljivo na slici 7.3, dok je pravi broj klasa 5.

Slika 7.3: Prikaz određivanja klasa iz Biologija dataseta korištenjem metode lakta

Assistments dataset:
Testiranje na Assistments u početku je okarakterizirano kao nemoguće jer K-Medoids algoritam zahtijeva da su svi vektori iste dimenzionalnosti odnosno da su svi ispitanici odgovorili na sva pitanja. To u Assistments nije zadovoljeno. Eventualno se može probati s nekakvom redukcijom dimenzionalnosti vektora (npr. *principal component analysis*, međutim budući da ima pitanja na koja je samo jedan student odgovorio, na kraju bismo trebali završiti s dimenzionalnošću 1 (što je puno premalo da bi obuhvaćalo potrebne informacije).
U nastavku rada na ovom pristupu, napravljena je prilagodba za učitavanje smanjenog Assistments dataseta (20 pitanja). No, početna pretpostavka da testiranje s Assistemntsom neće biti moguće pokazala se ispravnom. Isprobana je metoda grupiranja slična K-Medoids, K-Means, ali nije pokazala željenu funkcionalnost.

7.3. Ostatak istraživanja

Mnogi radovi pokazali su se u cjelovitosti ne toliko relevantnima da bismo ih mogli u potpunosti primijeniti na naš problem, ali su predstavili mnoge iskoristive koncepte i algoritme vrijedne spomena.

U radu (13) spominje se da je razumijevanje krajnjeg cilja svakog studenta praćeno pomoću posebne varijable mastery_score koja se osvježava svaki puta kada se vrši procjena studentovog znanja.
Spominje se da identificiraju i vizualiziraju grupe povezanih koncepata. Točnije, *spreading algorithm* pronalazi uzorak u povezanosti koncepata i dokumenata te dokumenata i koncepata. Analizirana je struktura *spreading algorithma* (14).
Problem je što je potrebna neka vrsta reference, ekspertno određenog početnog grafa koji će se evaluirati tim algoritmom.

U radu (15) predstavljene su hijerarhijske veze pitanja i koncepata modelirane pomoću hinge loss skalarnog produkta ugrađenih koncepata i pitanja. Problem: povezanost koncepata i pitanja određena ekspertima i zapisana u matrici ugradnje, a eksperte želimo izbjeći. Korišten je t-distributed stochastic neighbor embedding (tSNE) algoritam. tSNE je algoritam strojnog učenja za vizualizaciju; nelinearna tehnika za redukciju dimenzionalnosti. Koristi se za ugradnju visokodimenzionalnih podataka za vizualizaciju u niskodimenzionalni prostor (2 ili 3 dimenzije). Slični objekti su s visokom vjerojatnosti modelirani točkama koje su blizu u prostoru, a različiti objekti udaljenim točkama (16). Problem: velika osjetljivost algoritma na promjenu parametara.

Rad (17) primarno radi poveznicu između samih koncepata, ali i između koncepata i objekata učenja (zadataka, primjera, itd.) uz pretpostavke da je broj objekata učenja poznat. Glavni problem i razlog napuštanja ovakvog modela je što se *preprocessing* objekata učenja radi pomoću *text-mininga*. Isto tako, rad pretpostavlja da je netko nekad označio povezanost koncepata i objekata učenja pa da može koristiti tu referencu. Nad tom referencom oblikuje se tzv. *contextual network* koja sadrži više vrsta čvorova. Za izračunavanje sličnosti koncepata koriste se dva načina: već spomenuti *spreading activation algorithm* i *PageRank with Priors*. *PageRank with Priors* koristi se kao kvantitativna mjera sličnosti čvorova prema nekom drugom čvoru. Čestu primjenu ima u sustavima kategorizacije. U konkretnom primjeru,

propagiraju se stvarne karakteristike modela domene znanja u eksplicitne veze među konceptima.

U radu (18) veoma je izraženo shvaćenje da mnogi radovi poistovjećuju koncepte i zadatke i to pojednostavljenje uzrokuje probleme.
Prepoznaju da grafovi znanja ciljaju da predstave znanje u obliku grafova tripleta; triju činjenica - (početni entitet, veza, završni entitet).
Razlikuju povezanost koncepta i zadataka, koncepata međusobno te zadataka međusobno. Ovisno o vrsti povezanosti, koriste različite funkcije gubitka.
Koncepte enkodiraju kao sfere, a zadatke kao vektore u istom semantičkom prostoru.
Metoda je evaluirana pomoću predikcije veza i klasifikacije tripleta. Potonja ocjenjuje točnost tripleta. Predikcija veza predviđa početni ili završni entitet iz tripleta, ovisno koji nedostaje. Algoritmu je potrebno dati rangirane preporuke zadataka, ne samo najbolji rezultat. Sve mogućnosti se rangiraju kao moguće upopunjavanje traženog praznog mjesta prema udaljenosti u određenoj funkciji gubitka. Za evaluaciju koriste se dvije mjere:

1. srednja recipročna mjera svih točnih instanci (MRR) i
2. razmjer točnih zadataka koji rangiraju ne više od N (Hits@N).

Glavni problem, kao i u većini istraženih metoda: znaju koliko je koncepata.

Rad (19) slične vježbe grupira u koncepte naziva “problem schemas”. Koristi se hierarchical graph neural network. Konkretno, ovdje je mreža dvoslojna: donji je sloj zadataka (svaki čvor je jedan zadatak), gornji je sloj problemskih shema. Povezanost zadataka i problemskih shema modelira se *assignment matrixom* koji se dobije pomoću *hierarchical clustering analysis (HCA)*. HCA je metoda analize grupiranja bez nadzora korištenjem aglomerativnih ili divizivnih strategija za izgradnju hijerarhije grupa. Za konstrukciju hijerarhijskog grafa zadataka spominje se iskorištavanje semantičke informacije zadataka i shema, ali kasnije se spominje treniranje mreže samo pomoću HCA pa je i to jedan od pristupa koji valja detaljnije istražiti.

Istražen je i alat Ampligraph (20; 21). Površna analiza definirala ga je kao dosta moćan alat. Funkcionalnosti obuhvaćaju stvaranje grafa znanja, treniranje ugradbenog modela na tripletima, evaluaciju modela, vizualno grupiranje pojmova. Problem: s obzirom da koristi triplete (subjekt, predikat, objekt), potrebno je poznavati povezanost pitanja.

Daljnja proučavanja vratila su nas na rad s početka, Deep Knowledge Tracing (DKT), (2). U njemu je spomenuto kako se DKT model može koristiti za zadatke otkrivanja latentne strukture ili koncepata u podacima. Problemu je pristupljeno na način da se utjecaj J_{ij} dodijeli svakom usmjerenom paru zadataka i i j, gdje je $y(i|j)$ vjerojatnost točnosti koju RNN dodjeljuje zadatku j ako je učenik u prethodnom koraku točno odgovorio na zadatak i. Opisano je vidljivo u jednadžbi 7.1.

$$J_{ij} = \frac{y(j \mid i)}{\sum_k y(j \mid k)} \tag{7.1}$$

Kao dodatni pristup prepoznat je i rad povezan s kasnije detaljno opisanim ExRec-om (22), Dynamic Key-Value Memory Network Knowledge Tracing (DKVMN) (23). U radu je spomenuto kako se bave otkrivanjem uzoraka (koncepata) zadataka pomoću korelacijskih težina. Korelacijske težine između zadataka i koncepata impliciraju snagu njihove povezanosti. Nije potrebno otkiriti ovisnosti među zadacima i onda definirati prag za grupiranje, kako je to obično u povezanim radovima. Svaki zadatak povezuje se s jednim konceptom prema najvećoj vrijednosti korelacijske težine. Spominje se i korištenje već spomenutog t-SNE algoritmna (16) za prikaz grafa projiciranjem multidimenzionalnih korelacijskih težina u 2D točke. Problem ovog pristupa je što nije jasno definirano kako stvaraju korelacijske težine.

8. Općenito o ExRec-u

U radu (22) prepoznata je neefikanost prevladavajućih sustava otvorenih online tečajeva (MOOC) zbog dodjeljivanja istih vježbi svim studentima. S druge strane, personalizirani sustavi preporuka zadataka mogu poboljšati efikasnost učenja prilagođavajući se razini znanja svakog studenta (učenika, korisnika). Takvo razmišljanje uvelike odgovara našem problemu. U originalnom radu krenulo se od temeljne pretpostavke da je ubrzavanje učenja temeljni cilj svakog personaliziranog sustava preporuke zadataka u obrazovanju. Rad predlaže personalizirani sustav preporuke zadataka za online učenje poboljšanjem performansi postojećih *knowledge tracing* modela. Napominje se da postojeći modeli praćenja znanja ne koriste informacije o pripadnosti zadataka konceptima. Konkretno, ovim radom modificira se *dynamic key-value memory network knowledge tracing (DKVMN)* model tako da se memorijska struktura temelji na listi koncepata određenog tečaja eksplicitno bilježeći vezu zadataka i koncepata tijekom procesa praćenja znanja. Model je korišten za izgradnju simulatora studenata korištenog za treniranje strategije preporuke zadataka pomoću podržanog učenja.

8.1. Povezani radovi

ExRec je sustav sastavljen od praćenja znanja i preporuke zadataka. Nadogradnja je već poznatih i istraženih sustava. Praćenje znanja inače često koristi prije istražen i implementiran Bayesovski model praćenja znanja (BKT). Modelira studentovo znanje koncepta kao binarnu varijablu i, koristeći skriveni Markovljev model, ažurira vjerojatnost njegova ovladavanja konceptom uzimajući u obzir rezultate rješavanja zadataka. Taj je model na razini koncepata i zanemaruje odnose između različitih koncepata.

Drugi pristup može se pronaći u također već istraženom modelu dubokog praćenja

znanja (DKT) s povratnom neuronskom mrežom. Modelira znanje studenta kao latentnu varijablu. DKT je korišten i za sustav preporuke.

Preporuke zadataka većinom koriste heurističke algoritme. Studentu se preporučuje zadatak ako je vjerojatnost da će ga točno riješiti oko 50%. Optimalnost takvog algoritma je upitna.
Koristi se i pristup određivanja ZPD-a (zona proksimalnog razvoja, engl. *Zone of Proximal Development*) na temelju trenutnog znanja učenika, a zatim se odabire najkorisniji zadatak pomoću algoritma više naoružanih razbojnika (engl. *multi-armed bandits algorithm*).
SPARFA framework se koristi za procjenu znanja svakog učenika iz njihovih prethodnih rezultata. Zatim koristi te profile znanja kao kontekst i primjenjuje kontekstualni algoritam naoružanih razbojnika za preporuku zadataka kako bi se maksimizirao učenikov neposredan uspjeh, odnosno njegov uspjeh na sljedećem zadatku. Problem ovog algoritma je što uzima u obzir samo sljedeći korak (kratkoročna nagrada) stoga njegova izvedba ne mora biti optimalna.

8.2. Pozadina sustava i dataset

Originalni ExRec sustav kao dataset uzima uzorke studentskih interakcija iz IPS-a (engl. *Intelligent practice System*). IPS je kineski online sustav za samostalno učenje. U IPS-u svaki tečaj ima na desetke gradiva, a student sam bira željeno gradivo. Svako gradivo ima 7 stadija iz kojeg je moguće izaći u bilo kojem trenutku (ili promijeniti gradivo):

1. vježbe za zagrijavanje prije nastave
2. vježbe na nastavi prije predavanja
3. video predavanja
4. vježbe na nastavi nakon predavanja
5. domaća zadaća
6. vježbe pregleda gradiva

7. vježbe pregleda raznih gradiva.

Svaka vježba (zadatak) ima tri hijerarhijske oznake koncepata (*tagove*) koje su im pridijelili eksperti. U stadijima od 1 do 5 uključeni su sadržaji jednog koncepta, dok 6 i 7 sadrže vježbe drugih koncepata zbog procjene naučenosti. Sustav sprema trajanje studentovog učenja u svakom stadiju, vježbe koje polaže i točnost rezultata.

Za potrebe našeg projekta napravljena je skripta koja pretvara Assistments data-set (skillbuilder.csv) u format potreban za generiranje preporuka.
Za svakog studenta rezervirana su 4 retka:

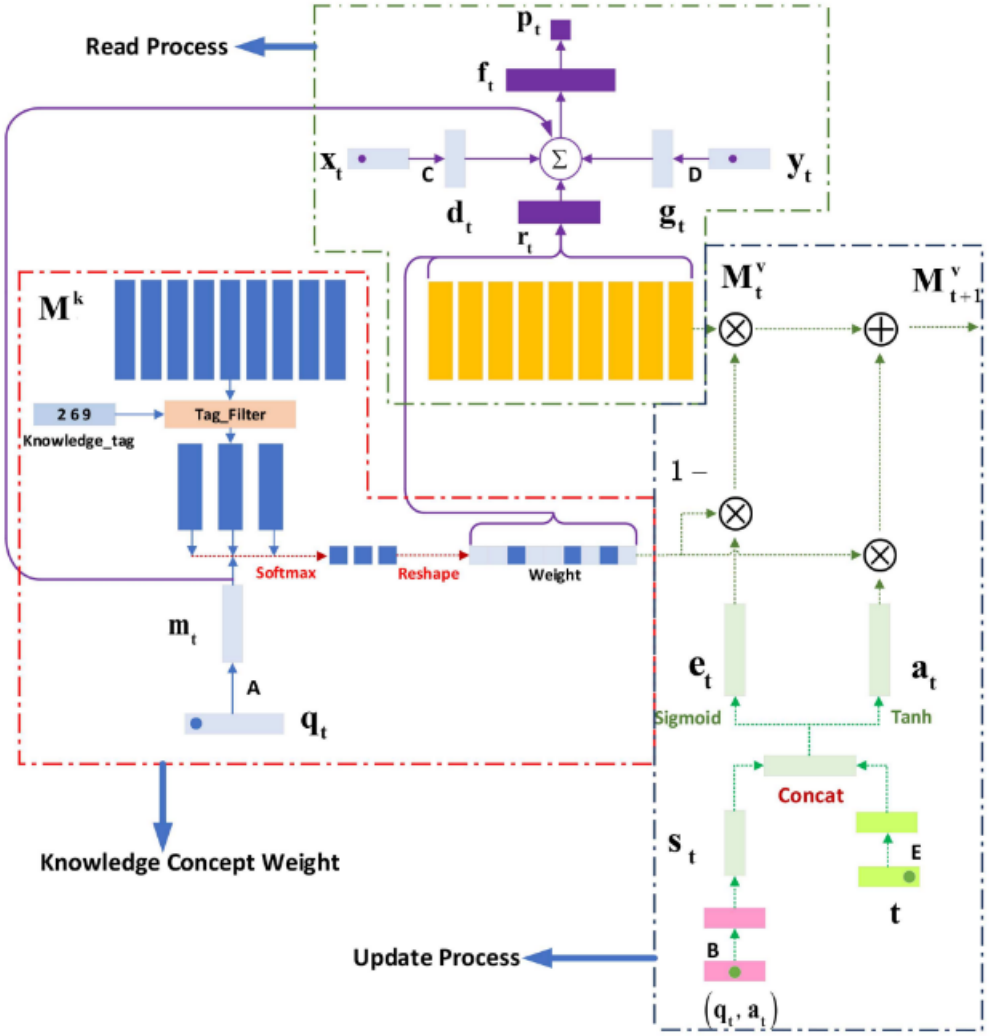
- prvi redak je broj odgovorenih pitanja/vježbi,
- drugi redak je lista identifikacijskih brojeva pitanja,
- treći redak redoslijed točnosti odgovora na pitanja, a
- četvrti redak sadrži koncept prve razine svakog pitanja.

8.3. DKVMN - model za praćenje znanja

DKVMN se generalno sastoji od statičke matrice (ključa) koji sprema latentne koncepte i dinamičke matrice (vrijednosti) koja sadrži razine savladnosti određenog koncepta. Model računa korelaciju vježbe i latentnog koncepta u ključu. Korelaciju koristi za čitanje studentovih razina savladanost koncepta i predviđa točnost ishoda rješavanja zadatka.

8.3.1. Concept Aware struktura

DKVMN ExRec praćenja znanja poboljšan u aspektima memorijske strukture, težina koncepata znanja te procesa pisanja i čitanja. Osnovni DKVMN dizajn je modificiran tako da memorijska struktura ovisi o listi koncepata nekog tečaja. Na slici 8.1 prikazana je struktura modela. M_t^k je matrica ugrađenih koncepata veličine MXN gdje je N broj memorijskih lokacija, a M je veličina vektora na svakoj lokaciji. N se postavlja na broj koncepata znanja određenog tečaja. Pošto se u radu koristio 1 koncept prve razine, 7 koncepata druge razine te 15 treće razine, N=23. Nakon toga, na svakoj lokaciji koncepta znanja se sprema studentovo znanje.



Slika 8.1: Struktura ExRec modela

8.3.2. Knowledge Concept Weight

Pošto je studentovo stanje koncepta znanja spremljeno u memoriji, kada se pojavi novi zadatak dohvaćaju se i osvježavaju samo memorijske lokacije povezane s tom vježbom. Za svaki koncept znanja se računa težina, težine se koriste kako bi se izračunala težinska suma trenutnih stanja koncepata znanja korisnika kako bi se predvidio njegov rezultat na vježbi. Također će se koristiti kako bi se osvježio korisnikovo stanje znanja nakon primitka rezultata od zadatka.

Prvo se dohvaćaju ugrađene vrijednosti od danog zadatka. Kao što se vidi na slici 8.1, kada zadatak q_t dođe u trenutku t prvo se transformira u ugrađeni vektor m_t kroz ugrađujuću matricu A. Tada se KCW računa pomoću algoritma 1 vidljivog na slici

8.2. Ukratko, DKVMN računa težinske veze između zadataka i skrivenih koncepata znanja, dok se u ovom kodu računaju samo veze između zadataka i poznatih povezanih koncepata, nepovezani koncepti se postavljaju na 0.

Algorithm 1 Knowledge Concept Weight Calculation

Input:
$\mathbf{q}_t$: embedding of the exercise arrived at time t
K_t: knowledge concept list of q_t
$\mathbf{M}^k$: the concept embedding matrix

Output:
$Weight$: Knowledge concept weight of the exercise arrived at time t

/ Calculate KCW */*
$R \Leftarrow [\,]$
for each $n \in K_t$ **do**
  $corr \Leftarrow \mathbf{m}_t^T \cdot \mathbf{M}^k[n]$
  $R.append(corr)$
end for
$R_s \Leftarrow Softmax(R)$
/ Reshape the weight vector to make its length equal to the number of concepts*/*
$Weight \Leftarrow [0, ..., 0]$
$i \Leftarrow 0$
for $i < 3$ **do**
  Weight$[K_t[\mathrm{i}]] \Leftarrow R_s[i]$
  $i \Leftarrow i + 1$
end for
return $Weight$

Slika 8.2: Algoritam računa KCW-a

8.3.3. Proces čitanja

Nakon što se izračunati KCW iskoristi za izračun težinske sume stanja koncepata znanja određenog korisnika, $r_t = \sum_{i=1}^{N} w_i M_t^v$, na sumu r_t nadodaju se značajke težine zadatka i "stage feature" d_t, g_t. Rezultat se šalje u potpuno povezani sloj sa aktivacijskom funkcijom Tanh kako bi se dobio "summary" vektor koji sadrži sve informacije o studentovom znanju povezanom s q_t i značajkama zadatka.

$$f_t = Tanh(W_0^T[r_t, d_t, g_t, m_t]) \tag{8.1}$$

Na kraju, f_t prolazi kroz potpuno povezani sloj koji daje kao izlaz vjerojatnosti da

bi student odradio zadatake q_t točno.

$$p = Sigmoid(W_1^T f_t) \tag{8.2}$$

8.3.4. Proces ažuriranja

Proces ažuriranja mijenja vrijednosti matrice M_t^v koja predstavlja trenutno stanje studentovog koncepta znanja k. Ovaj model je drukčiji od DKVMN po tome da se razmatra i trajanje rješavanja. Prema povezanim radovima, trajanje rješavanja zadatka je povezano sa razinom znanja studenta. Pošto je vrijeme rješavanja kontinuirana varijabla, ona se prvo diskretizira po njenoj distribuciji i onda se prikaže kao varijabla t te se koristi kako bi se ažurirala matrica M. Ostali procesi su isti kao i u DKVMN, sastoji se od podprocesa dodavanja i brisanja. Vektor brisanja dobije se kao $e = Sigmoid(E^T[s_t, t])$, dok se vektor dodavanja dobije kao $a = Tanh(D^T[s_t, t])$. Nova matrica M računa se kao

$$M_{t+1}^v(i+1) = M_t^v(i)[1 - w(i)e][1 + w(i)]. \tag{8.3}$$

Parametri ovog modela se računaju tako da se minimizira "cross-entropy loss" između pretpostavljenog studentovog dobvivenog i pretpostavljenog rezultata.

$$L = -\sum_t ((y_i \log p_t) + (1 - y_t) \log(1 - p_t)) \tag{8.4}$$

8.4. Preporuka zadataka podržanim učenjem

8.4.1. Općenito o podržanom učenju

Glavni elementi svakog algoritma podržanog učenja su, osim agenta i okoline, strategija, nagrada, vrijednost (predikcija nagrade) i opcionalno model. Strategija predstavlja agentov način ponašanja u određenom trenutku. Ona je preslikavanje iz spoznatog stanja okoliša u akcije koje je potrebno izvršiti u tim stanjima. Agentova je srž, ona sama je dovoljna da odredi ponašanje.

Cilj podržanog učenja definira nagrada koja preslikava opaženo stanje okoliša (ili par stanje-akcija) u jedan broj koji odgovara poželjnosti tog stanja. Cilj agenta je maksimizirati ukupnu nagradu. Nagrada definira što je trenutno dobro, a funkcija vrijednosti definira što je dugoročno dobro. Vrijednost stanja je očekivanje ukupne količine nagrade koja bi se mogla akumulirati u budućnosti počevši od tog stanja. To očekivanje još se naziva i Q funkcija.

Model imitira ponašanje okoline. Za dano stanje i akciju, model može predvidjeti rezultantno iduće stanje i nagradu.

U ovom radu okolina se izgrađuje simulatorom temeljenim na DKVMN-CA (CA = *concept aware*). Personalizirani agent za preporučivanje zadataka trenira se dubokim učenjem, točnije korištenjem GRU-a (engl. *Gated Recurrent Unit*): poboljšane verzije standarnih RNN.

Proces odlučivanja modelira se kao Markovljev proces odlučivanja s djelomično vidljivim stanjima (POMDP). POMDP je generalizacija Markovljevog procesa odlučivanja. Modelira agentov proces odlučivanja; vezu agenta i okoline. Formalno, POMDP je skup 7 varijabli (stanje, akcija, uvjetna vjerojatnost prijelaza među stanjima, nagrada, opažanja, uvjetne vjerojatnosti opažanja, koeficijent umanjenja nagrade). Agent u svakom stanju bira akciju koja će maksimizirati buduću nagradu umanjenu za faktor umanjenja.
Osnovna je pretpostavka u podržanom učenju da agent vidi stvarno stanje svijeta. No, agentova opažanja u stvarnom svijetu nisu nužno isto što i pravo stanje: uvodi se vjerojatnost stanja $p(o_i|s_i)$ - vjerojatnost da se za opažanje o_i svijet nalazi u s_i. Sljedeće stanje u svijetu ovisi o trenutnom stanju i agentovoj akciji. Dva stanja svijeta mogu rezultirati u jednakom opažanju agenata. Za opažanja ne vrijedi Markovljevo svojstvo (sljedeće opažanje stanja ne ovisi isključivo o trenutnom opažanju i akciji). Potrebno je uzeti u obzir putanju agenta.

8.4.2. Primjena na ExRec

Konkretno, u ExRec-u stanje modela je studentovo konkretno latentno znanje, a akcija je preporuka zadatka. Strategija preporuka direktno funkcionira na sirovim opažanjima studentove povijesti rješavanja zadataka. U trenutku t agent ne može vidjeti studentovo znanje s_t. No, opažanje mu je o_t - zadatak i točnost rješavanja uvjetovani pomoću s_t, $p(o_t|s_t)$. Agent preporučuje a_t na temelju povijesti studentovih zadataka $h_t = (o_1, a_1, o_2, a_2, ..., o_{t-1}, a_{t-1})$. Nakon završetka preporučenog zadatka, latentno znanje prelazi u s_{t+1} pomoću $p(s_{t+1}|s_t, a_t)$.
Agentu je potrebna strategija kako bi izabrao najbolju akciju za određeno stanje. Nagrada akcije a_t je usrednjena suma vjerojatnosti da će student točno riješiti sljedeći

zadatak, q, nakon što riješi predloženi zadatak u stanju s_{t+1}. Usrednjava se brojem zadataka. Taj sljedeći zadatak predviđen je simulatorom. Algoritam preporučuje zadatak q, čija je nagrada maksimalna. Opisano je vidljivo jednadžbom 8.5.

$$r_t = \frac{1}{K} \sum_{i=1}^{K} P_{t+1}(q_i) \tag{8.5}$$

Krajnji cilj je maksimizirati nagradu određene strategije gdje su trajektorije $(s_1, o_1, a_1, ...)$ preuzete iz distribucije trajektorija induciranih strategijom π. Nagrada je izražena jednadžbom 8.6.

$$R = \mathbb{E}_\tau \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \right] \tag{8.6}$$

POMDP se rješava pomoću TRPO algoritma (engl. *Trust Region Policy Optimization*). TRPO je algoritam optimizacije robustan na velik raspon zadataka dajući monotono poboljšanje izmjenom malog broja parametara. Algoritam stalno iznova optimizira lokalne aproksimacije očekivane povratne vrijednosti strategije s kaznom KL-divergencije.

8.5. Evaluacija performansi

Originalni ExRec model evaluiran je na IPS datasetu provodenjem 50 eksperimenata podjelom korisnika na skupove za treniranje i testiranje u omjeru 70:30. Kao mjera performansi odabrana je AUC krivulja. Procjenjena je i efikasnost korištenjem dodatnih značajki modela, poput težine zadataka, faze rješavanja i trajanja rješavanja čija upotreba dokazano poboljšava performanse modela.

8.5.1. Preporuke zadataka

Evaluacija porasta znanja studenta

Algoritmi za treniranje strategije preporuke zadataka pomoću podržanog učenja većinom su heuristički; zadatke ocijenjene kao prelagane ili preteške potrebno je izbjegavati. No, optimalnost takvih algoritama nije zadovoljena jer se u obzir uzima samo

kratkoročna nagrada. U ExRec-u se u obzir uzima dugoročna nagrada.
Kao strategije algoritma podržanog učenja u originalnom ExRec-u korištena su dva algoritma, Expectimax i RL. Kod Expectimaxa prvo se računa iznos predviđenog znanja u slučaju da se preporuči određen zadatak. Sustav kao preporuku izabire zadatak kojim bi maksimizirao predviđeno znanje u tom trenutku. RL razmatra dugoročnu nagradu akcije maksimizacijom Q funkcije.
Za usporedbu algoritama, autori rada izvlače 15 studenata iz dataseta. Za oba algoritma inicijaliziraju se simulatori pomoću slijeda prethodno riješenih zadataka svakog studenta. Preporuča se 50 zadataka. Sprema se prosjek predviđenog znanja svih studenata u svakom koraku preporuke.

Evaluacija preporuka

Za proučavanje RL strategije autori su osmislili dodatni eksperiment kojim se uzme student i njegovih prijašnje riješenih 5 zadataka te pokrene simulator. Preporuči mu se novih 5 zadataka pomocu RL. Vizualno se prikaže tih 10 zadataka u obliku (broj zadatka, povezani koncept, točnost). Promatra se povezanost točnosti zadataka i odgovarajućih koncepata. Ako ne postoji znanje o nekom konceptu, grafički prikaz je crne boje. Kako znanje raste, boja je svjetlija.
Kad student netočno riješi zadatak, algoritam mu preporuči povezani koncept. Ako padne i preporučeni zadatak, sustav mu nudi ponovno rješavanje.
Nakon povećanja znanja o nekom konceptu, prebacuje se na neki drugi koncept. Ako je zadatak okarakteriziran kao lagan, znanje se ne povećava pretjerano iako je točno riješen.
Ponovno se preporučuje početni zadatak i stvar se ponavlja dok se ne riješi točno.

8.5.2. Rezultati i problemi

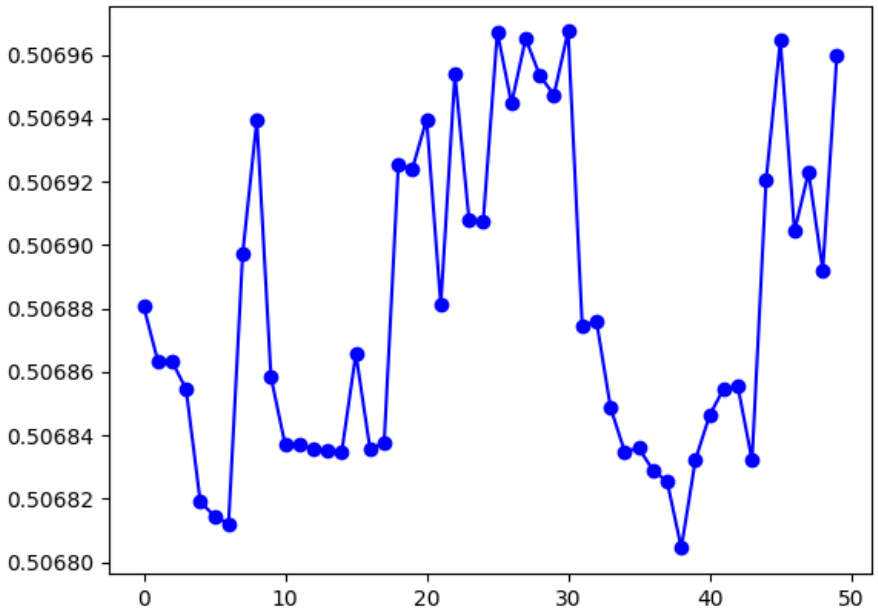
26.8.2020.

Prilagođena je skripta new_kt.py kako bi davala parametre za biologiju (smanjen je seq_len), napravljene su skripte koje stvaraju potrebne .pkl fileove bez da su putevi do datoteka hardkodirani. Također, napravljena je skripta koja dataset dijeli na dio za treniranje i dio za validaciju u formatu koji treba za pokretanje new_kt.py. Nakon što su broj koncepata i jedinstvenih pitanja postavljeni da odgovaraju datasetu biologije new_rs.py se može normalno pokrenuti.
Za razliku od Assistmentsa vrijednosti porasta znanja u svakom koraku su oko 0.507

te slabo osciliraju.

```
Preporuceni put: [(3, 0), (0, 0), (3, 0), (0, 0), (1, 0), (0, 0), (0, 0), (1, 1), (1, 1), (1, 0), (2, 0),
(3, 0), (0, 0), (0, 0), (0, 0), (0, 1), (1, 0), (3, 1), (1, 1), (3, 0), (0, 1), (1, 0), (1, 1), (2, 0), (3,
0), (1, 1), (0, 0), (2, 1), (0, 0), (0, 0), (1, 1), (1, 0), (3, 1), (2, 0), (2, 0), (3, 1), (2, 0), (2, 0),
(1, 0), (0, 1), (0, 1), (2, 0), (3, 1), (0, 0), (1, 1), (1, 1), (2, 0), (0, 1), (2, 0), (1, 1)]
[0.50688064, 0.5068635, 0.50686336, 0.50685453, 0.5068191, 0.5068145, 0.5068121, 0.50689733, 0.5069394,
0.50685835, 0.506837, 0.506837, 0.5068357, 0.5068351, 0.50683475, 0.50686574, 0.5068357, 0.50683767,
0.50692517, 0.50692415, 0.50693965, 0.5068811, 0.50695384, 0.50690794, 0.50690746, 0.5069672, 0.5069446,
0.5069651, 0.5069534, 0.50694734, 0.50696737, 0.5068743, 0.5068761, 0.5068487, 0.50683486, 0.50683594,
0.50682896, 0.50682545, 0.50680447, 0.5068323, 0.5068463, 0.5068547, 0.50685537, 0.5068321, 0.50692075,
0.50696445, 0.50690436, 0.50692296, 0.5068921, 0.50696003]
```

Slika 8.3



Slika 8.4

I dalje nije jasno koje vrijednosti hiperparametara poput seq_len i veličina memorije bi se trebali koristiti te koje je pravo značenje "porasta/vrijednosti" znanja u svakom trenutku. Za iste parametre i dataset new_rs.py uvijek daje drugačiji put preporuke što nije poželjno. Potrebno je i omogućiti izvedbu RS-a po chunkovima Assistmentsa koji nisu hardkodirani. Iz nekog razloga kada se koristi biologija trebaju se zakomentirati sve "isnan" funkcije, dok su iste potrebne kod Assitmentsa.

01.09.2020.

Napravljena je skripta koja automatizira pokretanje ExRec-a za jednostavne datasetove poput "Biologije". Trenutno je potrebno samo znati ime dataseta i put do direktorija u kojem je dataset. Ostalo je shvatiti kako prilagoditi model kt-a, generiranje train i validation dijelova dataseta kako se ne bi stvarali privremeni .csv fileovi. Također treba još proučiti shvatiti zašto kod učitavanja assistmentsa treba raditi dodatne provjere Nan-ova, dtype-a te imena stupaca, dok u isto vrijeme te iste provjere rade grešku kod dataseta "Biologija". Trenutne ideje za potencijalne preporuke zadataka:

- svi zadaci- nedostatak je što je jako sporo
- svi ostali zadaci iz koncepata kojih se korisnik dotakao- brže, ali i dalje prepsoro
- potencijalno korištenje clusteringa/BKT-a za preporuku manjeg skupa zadataka

04.09.2020. - eksperimentiranje s parametrima kt dijela

Provedeni su eksperimenti parametara, argumenata kt dijela - broja epoha, stope učenja, brzine smanjivanja stope učenja, momentuma koji ubrzava gradijentni spust, omjera rezanja tenzora kako gradijent ne bi eksplodirao te utjecaj *batch sizea* dataseta.
Najznačaniji zaključak izveden je za broj epoha - povećanjem rastu iznosi izlaznih vjerojatnosti. Eksperimentirano je s vrijednostima 1, 5, 100 i 1000. Najznačaniji rezultati dobivaju se za 1000, no dugotrajnost treniranja je jako izražena. Stoga je većina sljedećih eksperimenata provođena sa 100 epoha. Ostali parametri većinom zahtijevaju neznatne izmjene u odnosu na originalno postavljene.
Isprobane su vrijednosti [0.005, 0.05, 0.5]. Stopa učenja ne smije imati prevelik iznos jer rezultat pokazuje sumnjivo previsoku izlaznu vjerojanost, blizu 1, što ukazuje na prenaučenost sustava. Potrebno je odgovarajuće intervalno smanjivanje stope učenja, koje ne smije biti preveliko jer prebrzo konvergira, niti premalo jer je vrijeme izvođenja tada presporo iako su rezultati intuitiviji. Ako je potrebno birati, bolje manja stopa učenja i sporiji interval promjene nego veća stopa i veće promjene.
Istražen je i momentum parametar. Sa samim stohastičnim gradijentnim spustom, SGD, ne računamo točnu derivaciju funkcije gubitka, već estimiramo na manjim *batchevima*. Iz tog razloga, ne ide se uvijek u optimalnom smjeru jer su takve derivacije zašumljene. Kod SGD-a su isto tako problem tjesnaci strmiji u jednoj dimenziji (česti su blizu lokalnih minimuma). SGD ih teško uočava. Dodavanje Momentum algoritma

ubrazava njihove prelaske. Također, derivacije su glađe, manje zašumljene. Provedeni su eksperimenti s vrijednostima između 0.5 i 0.95. Što je parametar Momentuma manji, slijed podataka je više skokovit. Algoritam usrednjava na manjem broju podataka pa smo bliže zašumljenim podacima. Utvrđeno je da je najbolji momentum 0.9, što je vrijednost koja se često uzima u strojnom učenju.
Isprobane su i različite veličine maxgradnorm parametra. Naime, kako gradijenti ne bi eksplodirali tijekom treniranja, koriste se tehnike *gradient clippinga*. Eksplodiranje gradijenta dogodi se kad je gradijent prevelik, što dovodi do akumuliranja pogrešnih gradijenata. Kao rezultat dobije se nestabilna mreža. Iz tog razloga, u TensorFlow-u koristi se naredba *tf.clip_by_global_norm()* - reže tenzore u treniranju u omjeru sume njihovih normi, a omjer definira upravo maxgradnorm argument. Povećanjem i smanjenjem vrijednosti [5, 100], otkriveno je da se rezultati ne mijenjaju značajno te je stoga najbolje ostaviti originalno (50).
Što se tiče *batch sizea*, za "Biologiju" se pokazao boljim manji *batch* [5, 15, 32 -> 5], prilagođeniji činjenici da je sam dataset malen.

07.09.2020. - eksperimentiranje s parametrima kt i rs dijela

Variranjem kt parametara *memory_key_state_dim, memory_value_state_dim* te *final_fc_dim* zaključeno je da njihova veličina ne utječe na izlazne vrijednosti, već samo doprinosi brzini treniranja (ako manja veličina, brže se trenira). Već je prije zaključeno i opet potvrđeno kako paramteri *memory_size, n_questions* i *seq_len* moraju točno odgovarati vrijednostima dataseta: broju koncepata, broju pitanja te duljini niza treniranja.
U rs dijelu, dosadašnji eksperimenti većinom su izvođeni s jednom epohom treniranja RLTutora. Variranjem epoha na 10 i 100, nije došlo do značajnijih poboljšanja (ni povećanja ni stabilnosti) u rezultatu izlaza.
S druge strane, ako se parametar *hidden_dim* podesi na "Biologiji" prikaldnijih 5, s obzirom na postojanje 5 koncepata u datasetu (u odnosu na originalno postavljenih 32) zajedno s *batch_sizeom*, dobije se porast izlaza. Ipak, prilikom različitih pokretanja, izlazi imaju trend pada.
Ako se *batch_size raw_policyja LoggedTRPO* smanji s originalnih 4000 na prihvatljivijih 32, uz parametar *batch_size* kt-a i hidden_dim rs-a postavljen na 5, dobije se veća stabilnost. Ako se pritom broj epoha treniranja kt dijela poveća s 1 na 100, dobiju se znatno veće vrijednosti izlaza u odnosu na prethodna razmatranja (npr.
outList [0.7687031 0.8171221 0.83575428]

umjesto dosadašnjih
outList [0.5994091 0.59953147 0.59946656].
U Categorical GRU strategiji podržanog učenja (originalno postavljenoj), napravljeni su eksperimenti s postavljanjem različitih aktivacijskih funkcija koje u treniranje uvode nelinearnosti. Originalno je postavljen tanh. Postav sigmoide većinom povećava izlazne vrijednosti i njihov razmak, ali uvodi trend pada. Softmax ne pokazuje zadovoljavajuće rezultate; smanjuje izlazne vrijednosti i njihov razmak. Rectify (linear rectifier) aktivacijska funkcija preporučuje stalno isti zadatak u putanji, ali ne uspijeva se doći do točnog rješenja. Leaky_rectify se ponaša previše nepredvidljivo; vrijednosti skaču i padaju. Selu i linear aktivacijske funkcije ne pokazuju pravilnosti pri izvođenju. Elu (engl. *exponential linear unit*) pokazuje dosta dobra svojstva. Smanjuje *bias* (pristranost pomaka) u odnosu na rectify odmicanjem srednje aktivacije prema 0. Brže konvergira nuli i proizvodi točniji rezultat (jer nema negativan gradijent za negativne vrijednosti). Primjeri rješenja za "Biologiju":
Preporuceni put: [(1, 1), (0, 1), (2, 0)]
outList [0.68188673 0.68203878 0.77824879]
Preporuceni put: [(2, 1), (1, 1), (3, 1)]
outList [0.75084096 0.79288167 0.797158].
Eksperimentirano je i s vrijednostima *discounta* (između 0.5 i 0.99) u *Logged TRPO* i *n_steps* (2, 5, 10, 50) u definiciji okoline. Zaključeno je kako nije moguće definirati pravilnosti jer su rezultati previše skokoviti, previše osciliraju.
Provođene su i provjere ponašanja različitih strategija. Strategija je smatrana najvažnijim aspektom podržanog učenja agenta i zahtijeva određeni dugotrajniji *tuning*. U ExRec-u je originalno korištena strategija Categorical GRU koja sadrži GRU koji predviđa na temelju kategoričke distribucije. Kategoričke strategije su namijenjene diskretnim akcijskim prostorima. Očekuju da akcijske vrijednosti predstavljaju vjerojatnosnu distribuciju prema akciji. Iz takve distribucije svaka je akcija uzorkovana.
Uz spomenutu strategiju, najviše obećavajućom pokazala se Categorical MLP strategija koja sadrži MLP (engl. *multi-layer perceptron*) koji predviđa na temelju kategoričke distribucije. Uz zanemarivanje originalnih vrijednosti i postav *hidden_size = (5, 5)* dolazi do stalnog rasta izlaza, ali preporuka često uključuje isti zadatak koji korisnik nikako da točno riješi.
Preporuceni put: [(0, 1), (3, 0), (3, 0), (3, 0)]
outList [0.59386724 0.5939555 0.5939993 0.5940211]
Ako se koristi Categorical MLP uz *hidden_size = (5, 5)* i elu aktivacijsku funkciju, dolazi do veće stabilnosti uz trend rasta izlaznih vrijednosti. Naime, prilikom različtih

pokretanja, sustav preporučuje često preporučuje isti put za isti prijeđeni *trace*, što dosad nije bio slučaj.
Preporuceni put: [(3, 1), (2, 1), (1, 1)]
outList [0.59849751 0.59855151 0.59854913]
Preporuceni put: [(2, 1), (3, 1), (1, 1)]
outList [0.59207004 0.59207171 0.5922001]

08.09.2020. - optimizatori

TRPO algoritam, već spomenut prilikom objašnjavanja pozadine funkcioniranja podržanog učenja u ExRec-u iterativni je pristup optimizaciji strategije sa zagarantiranim monotonim napretkom.
Koristi *Natural Policy Gradient* prisup koji analitički rješava funkciju cilja. Problem je što se u izračunu koristi inverz Hesseove matrice drugih derivacija log vjerojatnosti strategije (Fisherove informacijske matrice), a on je jako kompliciran ako je strategija paramtrizirana brojnim parametrima. Također, inverz je često nestabilan. Kako se on ne bi morao eksplicitno tražiti, koristi se aproksimacija izraza u krnjoj verziji pristupa, *Truncated Natural Policy Gradient*. Za optimizaciju tog izračuna koristi se metoda konjugatnih gradijenata.
Koncept je vrlo sličan gradijentnom uzdizanju, ali ga je moguće izvesti u manje iteracija. U gradijentnom usponu, uvijek se prati najstrmiji gradijent. Putanja od konačne do početne točke može biti neoptimalna, "cik-cak". Takvu neefikasnost moguće je izbjeći konjugatnim gradijentom (ako je funkcija cilja kvadratna). Ako model ima N parametara, moguće je naći optimalnu točku u najviše N uzdizanja. U prvom potezu slijedi se smjer najdubljeg gradijenta i definira se optimalna točka u tom smjeru. Sljedeći smjer mora biti ortogonalan prema nekoj transformacijskoj matrici A (konjugatan) svim prethodnim smjerovima. Takav izračun mnogo je manje zahtjevan od izračuna inverza Hesseove matrice.
Ideja je bila provjeriti kako TRPO funkcionira s ostalim optimizacijama, tj. različitim pristupima aproksimaciji.
Ako se koristi optimizator prvog reda, za originalne parametre, rs dio produljuje vrijeme izvođenja (s par sekundi na preko 2 min). Vrijednosti vrlo malo osciliraju oko 0.50. Ako se, pak, parametri izmijene prema onima koji su generirali dobre rezultate prethodnih dana (u kt broj epoha 100, batch_size = 5, u rs hidden_dim = 5, batch_size = 32, aktivacijska funkcija = elu), dobiju se zadovoljavajući rezultati.

Preporuceni put: [(0, 0), (2, 0), (0, 1)]
outList [0.57768267 0.68047982 0.70207518]
Preporuceni put: [(0, 1), (3, 1)]
outList [0.72096705 0.72513175]
No, dobri rezultati nisu uvijek slučaj. Literatura spominje da optimizatori prvog reda nisu točni u područjima zakrivljenosti. Naime, s derivacijama prvog reda često se površina kojom putuje prilikom traženja gradijenta aproksimira glatkom. Ako je vrlo zakrivljena, potezi su jako loši.
Preporuceni put: [(3, 1), (2, 0), (2, 0), (2, 0)]
outList [0.3331522 0.29257703 0.2808274 0.27639025]
Preporuceni put: [(2, 0), (3, 1), (2, 0), (0, 0)]
outList [0.52635247 0.487258 0.38557106 0.35406634]
Za kontroliranje stope učenja originalno je korištena Adam metoda. Eksperimentima s Adadeltom te SGD + Momentum metodama nije došlo do većih odstupanja u rezultatima.
Osim gradijenta prvog reda i konjugatnog gradijenta, proučen je i isproban LBFGS optimizator. Metoda je to optimizacije koja koristi derivacije drugog reda. Kvazi-Newtonova je metoda; metoda aproksimacije Hesseove matrice.
Jedna od najpopularnijih metoda aproksimacije Hesseove matrice je BFGS metoda temeljena na kompletnoj povijesti gradijenata. LBFGS metoda (engl. *limited memory BFGS*) je temeljena na zadnjih m gradijenata. Popularna je jer je potrebno zadržati samo tih zadnjih m gradijenata (obično 10 do 20) što doprinosi smanjenju skladišnog prostora. Za razliku od potpunog BFGS-a, LBFGS nikad eksplicitno ne formira ili sprema procjenu Hesseove matrice.
Iako u teoriji obećavajuće, u praksi se većinom primjećuje preporuka samo jednog zadatka ili je pak vidljiv trend pada koji nije u skladu s našim zahtjevima.
Preporuceni put: [(0, 1), (2, 1)]
outList [0.60839105 0.60837907]
Preporuceni put: [(1, 0), (1, 1)]
outList [0.60479879 0.60473955]
Preporuceni put: [(1, 1)]
outList [0.59713161]

8.6. Upute za pokretanje

Za pokretanje ove verzije ExReca potrebno je imati dataset koji ima slijedeće stupce:

- user_id - identifikacijska oznaka korisnika
- problem_id -identifikacijska oznaka zadatka
- skill_id -identifikacijska oznaka koncepta
- correct - točnost rješenog zadataka (0 ili 1)

U skripti "sve_u_jednom.py" su napisane sve potrebne naredbe za pokretanje ExReca nad nekim datasetom. Prvo je potrebno dati putanju do .csv filea dataseta, zatim se u funkciji"get_chunks(path)" skripte chunk_analysis dataset dijeli na manje dijelove radi lakšeg pokretanja, ta funkcija ima i opcionalan argument chunk_size s kojim se kontrolira veličina chunka. Ako je chunk_size veći nego dataset, uzima se cijeli skup podataka. Funkcija vraća objekt tipa pandas dataframe. Dalje se željeni dataframe šalje u konstruktor klase ChunkInfo skripte chunk_analysis gdje se obrađuju određena svojstva dataframea i pakiraju se varijable koji će se koristiti za pravilno funkcioniranje ExReca.

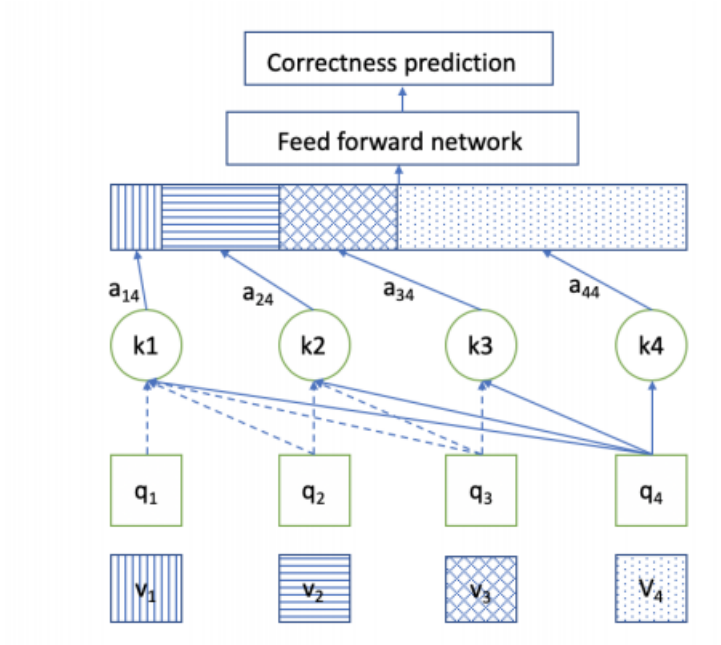
Prvi dio ExReca je Knowledge Tracing dio i za njegovo funkcioniranje potrebno je podijeliti dataset na train i validation dio to rade create_from_dataframe i create_from_csv funkcije skripte train_and_validate_creator. Ta funkcija ima opciju čitanja dataseta iz csv-a ili dataframea. Također ima opciju želi pisati nove datasetove kao .csv fileove ili da ih vraća kao dataframe varijable. Omjer train:valid je po defaultu stavljen na 7:3, ali postoji i argument funkcije kojim se i to može mijenjati. KT dio exreca računa parametre koji su potrebni za funkcioniranje Recommendation System dijela i vraća ih u varijablu params. Params se zajedno sa argumentima dobivenih iz chunk_analysisa šalju u RS dio gdje se generira pretpostavljeni "put" odabira i točnosti korisnikovih zadataka te ujedno daje i procjenjenu razinu znanja u svakom trenutku.

8.7. Poveznice

http://incompleteideas.net/book/first/ebook/node9.html
https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be
https://en.wikipedia.org/wiki/Partially_observable_Markov_decision_process
https://medium.com/@jonathan_hui/rl-trust-region-policy-

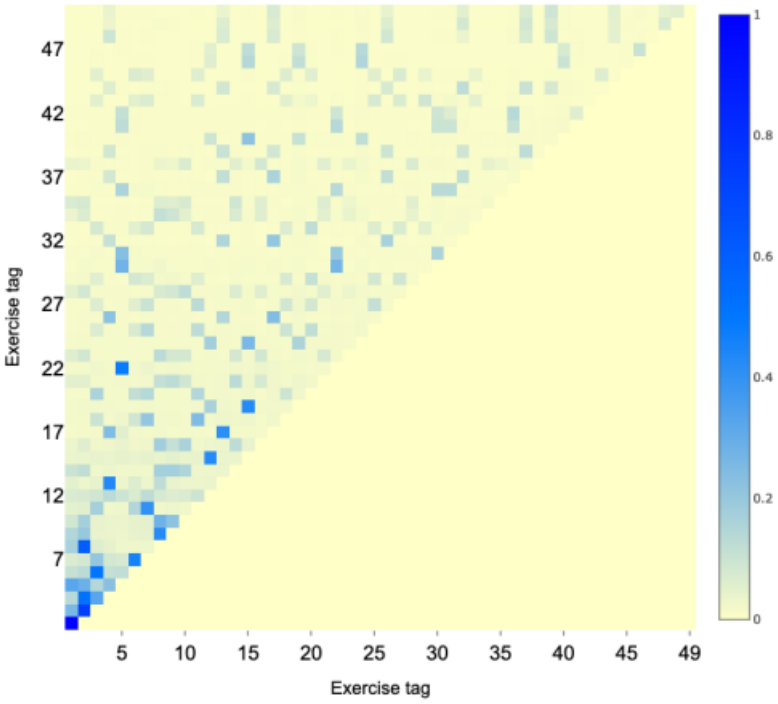
optimization-trpo-explained-a6ee04eeeee9
https://arxiv.org/abs/1502.05477
https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d
https://www.tensorflow.org/api_docs/python/tf/compat/v1/train/MomentumOptimizer
https://deepai.org/machine-learning-glossary-and-terms/gradient-clipping
https://stackoverflow.com/questions/49987839/how-to-handle-none-in-tf-clip-by-global-norm
https://lasagne.readthedocs.io/en/latest/modules/nonlinearities.html
https://garage.readthedocs.io/en/v2020.06.0/_apidoc/garage.tf.policies.categorical_gru_policy.html
https://nervanasystems.github.io/coach/components/exploration_policies/index.html
https://stable-baselines.readthedocs.io/en/master/modules/trpo.html
https://medium.com/@jonathan_hui/rl-trust-region-policy-optimization-trpo-part-2-f51e3b2e373a
https://stats.stackexchange.com/questions/284712/how-does-the-l-bfgs-work/285106

9. Self-attentive knowledge tracing

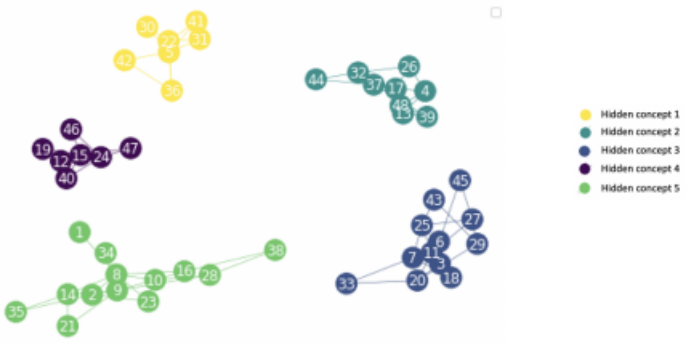


Slika 9.1

Pronađen je rad (24) koji istražuje novi smjer knowledge tracinga `https://github.com/TianHongZXY/pytorch-SAKT`. Predloženi model SAKT prvo identificira relevantne koncepte znanja iz prošlih interakcija te predviđa korisnikove performanse na tim konceptima. SAKT daje težinske vrijednosti prethodno odgovorenim pitanjima (pitanja su poistovjećena s konceptima) istovremeno predviđajući rezultate studenta na određenom pitanju. U radu se tvrdi da je prema AUC bolji za 4.43% od state-of-the-art metoda uprosječeno po svim korištenim skupovima podataka. Napomenuto je da DKT i DKVMN ne generaliziraju dobro u slučaju rijetkih, raspršenih podataka (kao što je slučaj s podacima interakcije studenata s nekoliko koncepata znanja stvarnog svijeta). Također, glavna komponenta (self-attention) se može paralelizirati što daje znatnu prednost po brzini naspram modela temeljenih na RNN-ovima.



Slika 9.2: Izračunata matrica relevantnosti između zadataka



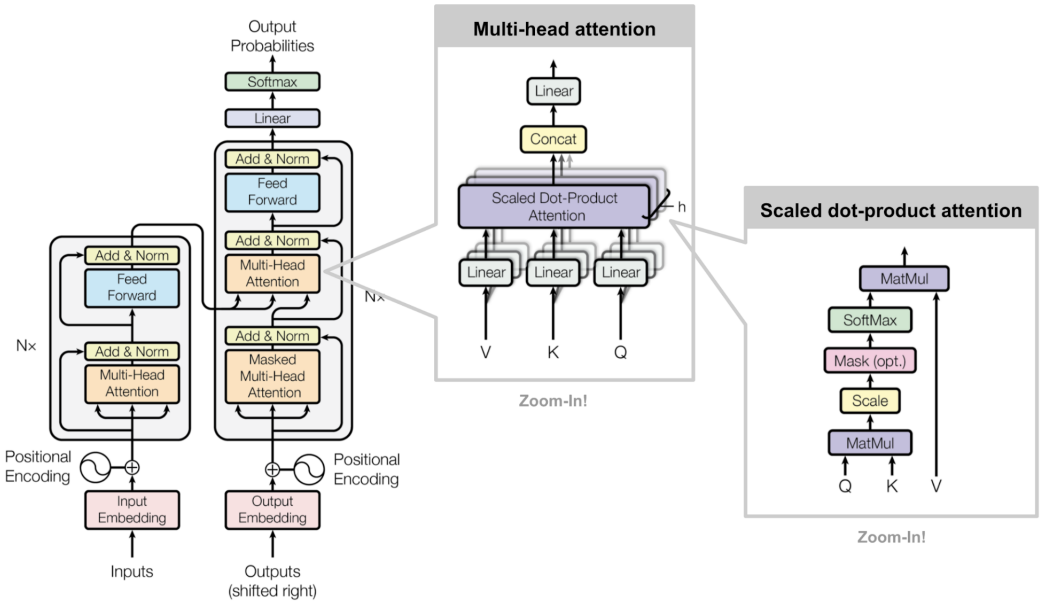
Slika 9.3: Pronađeni latentni koncepti

9.1. Općenito o funkcioniranju attention modela

Bit mehanizama pažnje (engl. *attention mechanism*) je što su oni imitacija mehanizma ljudskog vida. Kad vid detektira objekt, tipično ne skenira cijelu scenu nego se fokusira na određeni dio koji odgovara osobnim potrebama promatrača. Kad osoba primijeti da se željeni objekt tipično pojavljuje u određenom dijelu scene, naučit će se u budućnosti

fokusirati na takve dijelove.

Ovakvi mehanizmi najčešće su korišteni u obradi prirodnog jezika. Najpoznatiji model korišten u obradi prirodnog jezika poznat je kao transformer (slika 9.4). Transformeri su *sequence-to-sequence* arhitektura, tj. neuronska mreža koja pretvara određeni slijed elemenata, kao što je slijed riječi u rečenici, u neki drugi slijed. Takvi modeli sastoje se od enkodera i dekodera. Enkoder uzima ulazni slijed i mapira u višedimenzionalni prostor. Takav apstraktni vektor odlazi dekoderu koji ga pretvara u izlazni slijed. Tipično su u mehanizmima pažnje enkoder i dekoder LSTM-ovi, no kod transformera to nije slučaj.



Slika 9.4: Arhitektura transformera

Izračunavanje pažnje u računalnom smislu može se opisati kao mapiranje upita (engl. *query*) i skupova parova ključ-vrijednost (engl. *key-value*) izlazu. Prvo se uzmu upit i svaki ključ te se računa sličnost među njima kako bi se dobila težina. Kao funkcija sličnosti najčešće je korišten skalarni produkt. Drugi korak je provođenje normalizacije tih težina, najčešće softmax funkcijom. Potrebno je povezati težine s odgovarajućim vrijednostima. Rezultat je konačna pažnja.

Svaki input u model mora imati tri reprezentacije: ključ, upit i vrijednost. Kako bi se dobile te reprezentacije, svaki input mora biti pomnožen sa skupom težina za ključeve, upite i vrijednosti. Mehanizam pažnje za svaki input prvog LSTM-a (enkodera) uzima u obzir istovremeno različite inpute dodjeljujući im različite težine. Dekoder onda kao input uzima enkodiranu rečenicu i težine. Težine definiraju koliko svaki element slijeda utječe na ostale. Softmax se primjenjuje na težinama kako bi se stisnule u

interval [0, 1]. Mehanizam pažnje ponavlja se nekoliko puta s linearnim projekcijama upita, ključeva i vrijednosti. Sustav tako uči različite reprezentacije upita, ključeva i vrijednosti. Linearne reprezentacije su zapravo umnošci upita, ključeva i vrijednosti s matricom težina W.

Upiti, ključevi i vrijednosti su drugačiji ovisno o poziciji modula pažnje u strukturi - jesu li u enkoderu, dekoderu ili između njih. Multi-head attention modul koji povezuje enkoder i dekoder osigurava da enkoderov ulazni slijed bude uzet u obzir s dekoderovim ulaznim slijedom u određenoj poziciji. Nakon opisanog slijedi unaprijedni sloj koji ima iste parametre za svaku poziciju i može biti opisan kao odvojena, identična linearna transformacija svakog elementa danog slijeda.

Transformeri ne koriste RNN-ove, već se za izvlačenje globalnih ovisnosti inputa i outputa koriste samo self-attention mehanizmima. Self-attention mehanizmi dopuštaju inputu da međudjeluje s ostalima (self) i otkrije kome da posveti više pažnje. Enkoder i dekoder su sastavljeni od modula koji se mogu nadograđivati jedan na drugog nekoliko puta. Moduli se sastoje od Multi-Head Attention Mechanism i unaprijednih slojeva. Koristi se i rezidualni sloj za bolju optimizaciju (Add&Norm). Ulaz i izlaz prvo su ugrađeni u n-dimenzionalni prostor.

Budući da ne postoje RNN-ovi u modelu, potrebno je nekako pamtiti slijed koji se daje modelu; svakom dijelu slijeda dati relativnu poziciju u odnosu na red elementa - pozicijsko enkodiranje. Te pozicije dodane su ugradbenoj reprezentaciji (n-dimenzionalnom vektoru) svakog dijela slijeda. U treniranju je bitan shift slijeda dekodera. Ako se ne provodi, model nauči samo kopirati input dekodera. U slučaju shifta, model predviđa što će biti idući element. Osim shifta, transformer koristi masku na ulazu kako bi se izbjeglo viđenje potencijalnih budućih slijednih elemenata. To se mora provoditi zbog nedostatka RNN-ova.

Koncept pažnje skaliranog skalarnog produkta (engl. *Scaled Dot-Product Attention*) za račun sličnosti koristi skalarni produkt, kao što je spomenuto u prijašnjem tekstu. Ima dodatnu dimenziju za prilagodbu koja onemogućava da unutarnji produkt postane prevelik.

Kod Multi-Head Attention strukture upit, ključ i vrijednost prvo prolaze linearnu transformaciju i onda ulaze u račun pažnje skaliranog skalarnog produkta. Pažnja se računa h puta, otuda naziv Multi-Headed. Svaki put kada upit, ključ i vrijednost prolaze linearnu transformaciju, mijenja se parametar W, koji predstavlja težine. Rezultati h iteracija skaliranog skalarnog produkta spajaju se na kraju.

Ako se ne radi s nizovima riječi u obradi prirodnog jezika, potrebne su manje izmjene arhitekture, npr. moguće je maknuti embedding sloj ako podaci već jesu broj-

čane vrijednosti. Umjesto njega, moguće je primjeniti neku linearnu transformaciju.

9.2. Konkretna izvedba SAKT-a

Konkretna izvedba sustava predviđa hoće li korisnik moći odgovoriti na sljedeće pitanje ovisno o odgovorima na prethodna pitanja. Inputi su parovi (pitanje, točnost), konkretno: $x_1, x_2, ..., xt-1$, kao i slijed pitanja jednu poziciju naprijed, $e_2, e_3, ...e_t$, a izlaz su točnosti odgovarajućih pitanja: $r_2, r_3, ...r_t$. x_t je ugrađen u model kao $y_t = e_t + r_t$ x E, gdje je E ukupan broj pitanja. Takav slijed, $y = y_1, ..., y_t$, transformira se u $s = s_1, ...s_n$, gdje je n maksimalna duljina koju model može koristiti. Ako je $t < n$, izvršava se *padding*, a ako je $t > n$, slijed se rastavlja u podnizove duljine n.
Za određivanje redoslijeda koristi se pozicijsko enkodiranja (bitno jer znanje korisnika napreduje polako vremenom). Implementira se kao vrijednost dodana svakom elementu interakcijskog vektora ugradnje (čije je stvaranje opisano iznad) prilikom treniranja. Izlaz sloja ugradnje je ulazna matrica ugradnje, kao i matrica ugradnje pitanja.
Self-attention sloj računa skalirani skalarni produkt - računa relativne težine prema točnosti riješenosti prethodnih pitanja kako bi se predvidjela točnost trenutnog pitanja. Računaju se upiti (upit = sljedeće pitanje) i parovi ključ-vrijednost te linearne projekcije istih u različite vektorske prostore pomoću projekcijskih matrica W. Relevantnost svake od prijašnjih interakcija s trenutnim pitanjem računa se pomoću skalarnog produkta upita i ključeva (ključevi = težine između elemenata prethodnih interakcija). Kako bi se informaciji pristupilo iz različitih potprostora, vrše se linearne projekcije upita, ključeva i vrijednosti h puta korištenjem različitih projekcijskih matrica (*multiple heads*).
Svi dosadašnji izračuni su još uvijek linearna kombinacija vrijednosti prethodnih interakcija. Za uvođenje nelinearnosti i uzimanja u obzir interakcija između različitih latentnih dimenzija, koristi se unaprijedni sloj (i ReLU aktivacijska funkcija).
Rezidualne veze koriste se za propagiranje značajki nižih slojeva višim slojevima. Dakle, ako su značajke nižih slojeva značajne za predikciju, rezidualne veze će pomoći njihovom propagiranju završnim slojevima gdje se predikcija i obavlja. U *knowledge tracingu*, korisnik pristupa nekom pitanju kako bi ojačao određeni koncept. Rezidualna veza pomaže propagiranju ugradnje nedavno riješenih pitanja završnom sloju.
Normalizacija ulaza stabilizira i ubrzava neuronske mreže.
Za predviđanje performansi studenata, koristi se potpuno povezan sloj sa sigmoidalnom aktivacijskom funkcijom.

9.3. Problemi sa SAKT-om

Za vrijeme pručavanja i prilagođavanja SAKT-a našim potrebama naišli smo na puno grešaka i nejasnih linija koda. Najveći problem je slaba dokumentacija koda, nedostatak originalnog dataseta te manjak iskustva sa PyTorchom. Na kraju smo odlučili odustati od SAKT-a zbog manjka vremena. PyTorch na različite načine izvršava operacije s CPU i CUDA tenzorima. Gradijenti treniranja na CUDI mogu se prosljeđivati slojevima samo za operacije s tenzorima, ali ne s ostalim tipovima podataka. Imali smo problem što kod zahtijeva grafičke kartice (GPU CUDA) koje naši laptopi nemaju, a debug mode ne postoji na Google Colabu. Također, dosta često Colab ne ispisuje print() funkcije koje dolaze prije exceptiona tako da nismo ni na taj način mogli tražiti greške. U sljedećem potpoglavlju su nabrojane greške, neodumice i kako smo ih ispravljali ako će netko u budućnosti opet proučavati taj kod.

9.3.1. Greške i ispravci

U funkciji getitem skripte dataset.py se kod uzimanja listi nekad izbacuju prvi/ zadnji članovi ([1:],[:-1]), pošto to nema logike izmjena je takva da se uzima cijela lista ([:]). U datasetu se uzima num_skill kao najveći index u listi zadataka te se kasnije svugdje u programu stavljalo +1, to je izmijenjeno tako da se gleda broj zadataka te je maknut +1. U dodavanju podataka u varijablu 'x' su se dodavale True /False vrijednosti što je bacalo grešku te je sada 1/0. Također nije jasno zašto se na 'x' appendaju točnosti problema onoliko puta koliko i postoji različitih zadataka. U klasi DataPrefetcher dolazilo je do greške gdje objekt liste nema metodu .to(device=self.device, non_blocking=True). Pronađeno je da se takva metoda može pozivati nad objektom Tensor te je iskorištena metoda torch.cat().

Nadalje se javlja greška u skripti run.py, funkciji run_epoch - "can't convert cuda:0 device type tensor to numpy. Use Tensor.cpu() to copy the tensor to host memory first.". Tamo je potrebno sve varijable čije su memorijske lokacije na grafičkoj kartici zbog CUDA operacija vratiti na procesor (npr. problems.cpu()).

Daljnje greške nisu mogle biti ispravljene zbog nepoznavanja što bi ti dijelovi koda trebali raditi. Primjerice u istoj metodi kod ugniježdenih petlji ima pozivanja size(1) koji je zapravo nepostojeći te se dobiva greška da se očekuje 0 ili -1. Također, prema izgledu petlje čini se kako bi dodavanje offseta kroz neko vrijeme dovelo do indeksa koji su veći od maksimalnog za dana polja/tensore. Kako bi se to izbjeglo, zakomentirane su originalne linije pristupanja elementima te se umjesto iteriranja po listi i dodavanja

ta ista lista appenda na drugu.

Sljedeća greška se događala u student_modelu gdje se zbrajaju dva tensora različitih veličina - "output with shape [50400, 200] doesn't match the broadcast shape [1, 50400, 200]", kako bi se maknula dimenzija "1" nad tim objektom je napravljena operacije unsqueeze(0).

Greška koju nismo uspijeli ispraviti je "CUDA error: CUBLAS_STATUS_ALLOC_FAILED when calling 'cublasCreate(handle)",te dolazi u retku" res = self.multi_attn(query=self.layernorm(problems), key=x, value=x,key_masks=None, query_masks=None, future_masks=None)". Ono što smo uspjeli saznati je da to vrlo vjerojatno dolazi kada embedding layer dobiva krive indekse i izađe izvan intervala legalnih indeksa, odnosno, moguće je da postoji nekonzistentnost između broja oznaka i broja izlaznih jedinica sloja.

9.4. Sakt #2

Pronađen je još jedan github repozitorij `https://github.com/thosgt/kt-algos` koji u sebi ima implementaciju SAKTA-a. Također uz to ima implementaciju DKT-a i još neke pomoćne skripte. SAKT dio je prilagođen da se može pokretati za assistments i biologiju na Google Colabu. Za assistments se dobivaju velike vrijednosti AUC za vrijeme treniranja što se poklapa s prethodnim radom. Trenutno još nije sigurno kako bi se predikcije iz SAKT-a mogle iskoristiti za preporuku sadržaja.

9.4.1. Generiranje candidate-exercises

Dio SAKT-a za vrijeme učenja razvija tzv. *attention matricu*. Ona modelu daje neku mjeru "relevantnosti" između pojedinih zadataka. Ideja je izvući tu matricu iz modela i iskoristiti njene vrijednosti za uzimanje dijela zadtaka kao potencijalnu preporuku u ExRec-u. Napravljena je klasa koja prima attention matricu, funkciju normalizacije, funkciju praga te vrijednost praga. Funkcija normalizacije služi kako bi se svaki redak matrice normalizirao po nekom pravilu i onda kasnije uz pomoć funkcije praga odredili zadaci za preporuku. Klasa je napravljena tako da neće preporučiti zadatke koji su već bili. Recommendation system dio ExRec-a uzima u obzir točnost riješenih zadataka te je vremenski zahtjevan. Alternativa tome je koristiti razvijenu klasu tako da je njen izlaz konkretna preporuka zadataka umjesto skup potencijalnih zadataka koje Recommendation System još treba obraditi.

Eksperimentirano je s vremenom uzimanja vrijednosti attention matrice. Uzimanje

je postavljeno nakon svakog *train_batcha* i na kraju faze treniranja. Problem je što bi se, zbog korištenja softmax normalizacije pri preporučivanju, vrijednosti u pojedinim retcima trebale postaviti unutar intervala [0, 1], no to nije slučaj. Trenutno razmišljanje je da je to zbog izbacivanja nepotrebnih dimenzija tenzora. Zbog manjeg poznavanja PyTorch funkcionalnosti i načina ophođenja s visokodimenzionalnim tenzorima, moguće je da dimenzije okarakterizirane kao nepotrebne zapravo to nisu.

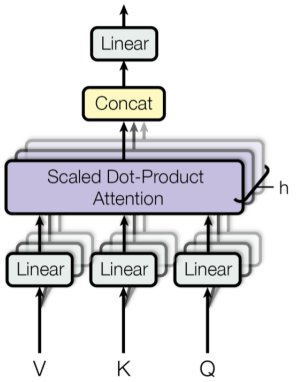
Softmax je inače preporučen kao funkcija koja se koristi pri problemima multinomijalne klasifikacije (u više od dviju klasa), a kod binarne klasifikacije češće je korištena sigmoida. Eksperimentiranje s aktivacijskim funkcijama dovelo je do zaključka da je uz korištenje softmaxa potrebno upotrebljavati niži prag kako bi se dobile smislene preporuke - zadaci. Potrebno je još ispitivanja i isprobavanja raznih kombinacija funkcija i pragova.

9.4.2. Usporedba SAKT-a i DKT-a

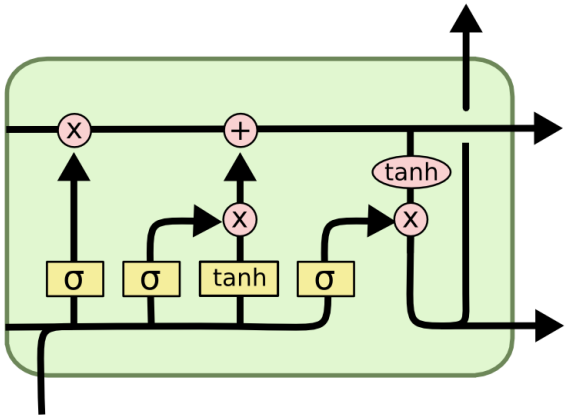
DKT koristi BPTT algoritam (engl. *backpropagation through time*). BPTT je gradijentna tehnika za treniranje nekih vrsta RNN-ova. Standardni BPTT zahtijeva kompletnu povijest aktivacija i inputa u unaprijednom prolazu kako bi ih koristio za izračun gradijenta u povratnom prolazu. Takav pristup je računalno skup i zahtjevan za memoriju.

Kod krnjeg BPTT algoritma (engl. *truncated backpropagation through time*), ulazi su podnizovi fiksnih duljina. Za unaprijedni prolaz, skriveno stanje prethodnog podniza prosljeđuje se kao ulaz sljedećem. Kod izračuna gradijenta, vrijednosti su odbačene na kraju svakog podniza u povratku. Ovakav pristup smanjuje računske i memorijske zahtjeve.

Kod DKT-a se ne koristi Multi-Head Attention Mechanism (slika 9.5), već LSTM (slika 9.6). RNN-ovi su prilično dobar način za obuhvaćanje vremenskih ovisnosti u nizovima.



Slika 9.5: Multi-Head Attention Mechanism



Slika 9.6: LSTM

Sličnost implementacije KT modela DKT i SAKT veoma je uočljiva. Modeli dijele identičnost početka embedding sloja. U daljnjem postupku DKT koristi LSTM, dok SAKT Multi-Head Attention Mechanism. Za obilježavanje slijeda, SAKT mora koristiti pozicijsko enkodiranje, dok LSTM kod DKT-a takav posao obavlja implicitno u svojoj arhitekturi.

Primjećeno je da je gubitak treniranja DKT-a većeg iznosa nego kod SAKT-a i kod Biologije i kod Assistmentsa (oko 1 u prvih 240-700 koraka uz DKT, dok je uza SAKT oko 0.6).

Treniranje DKT-a na Assistentsu traje otprilike minutu po epohi, dok Biologija treba par sekundi po epohi. Treniranje SAKT-a po epohi traje nešto malo više od minute po epohi, dok je za Biologiju potrebno i manje od sekunde po epohi.

Za SAKT je potrebno puno više korištenja CUDA arhitekture nego za DKT.

9.5. Poveznice

https://medium.com/@Alibaba_Cloud/self-attention-mechanisms-in-natural-language-processing-9f28315ff905
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a
https://www.geeksforgeeks.org/activation-functions-neural-networks/
https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04
https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

10. Literatura

[1] Knewton, https://www.knewton.com/blog/mastery/what-are-knewtons-knowledge-graphs/

[2] Deep Knowledge Tracing, http://papers.nips.cc/paper/5654-deep-knowledge-tracing.pdf

[3] KnowEdu, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8362657

[4] Building Knowledge Graph using NLP https://www.analyticsvidhya.com/blog/2019/10/how-to-build-knowledge-graph-text-using-spacy/

[5] K12EduKG, https://aic-fe.bnu.edu.cn/docs/20181205101832069569.pdf

[6] Deep Generative Models, https://arxiv.org/abs/1803.03324

[7] Deep Autoregressive Models, https://eigenfoo.xyz/deep-autoregressive-models/

[8] Graph Recurrent Attention Networks, https://arxiv.org/abs/1910.00760

[9] Latent Skill Embedding (Lentil), https://arxiv.org/pdf/1602.07029.pdf

[10] Entropy Rates of a Stochastic Process, http://math.ubbcluj.ro/~tradu/TI/coverch4_article.pdf

[11] Entropy Based Measured for Graphs, https://www.slideshare.net/bamparopoulos/entropy-based-measures-for-graphs

[12] Spectral Bayesian Knowledge Tracing, https://www.educationaldatamining.org/EDM2015/proceedings/short360-363.pdf

[13] Interactive Learning in Intelligent Tutoring Systems, https://www.researchgate.net/publication/333828640_INTERACTIVE_LEARNING_IN_A_CONVERSATIONAL_INTELLIGENT_TUTORING_SYSTEM_USING_STUDENT_FEEDBACK_CONCEPT_GROUPING_AND_TEXT_LINKING

[14] Spreading Activation Algortihm, https://en.wikipedia.org/wiki/Spreading_activation

[15] Deep Hierarchical Knowledge Tracing, http://www.personal.psu.edu/ffm5105/files/2019/edm19.pdf

[16] T-disturbed Stochastic Neighbor Embedding, https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding

[17] Automatic Concept Relationships Discovery for an Adaptive E-course, https://www.educationaldatamining.org/EDM2009/uploads/proceedings/simko.pdf

[18] Differentiating Concepts and Instances for Knowledge Graph Embedding, https://www.aclweb.org/anthology/D18-1222/

[19] HGKT : Introducing Problem Schema with Hierarchical Exercise Graph for Knowledge Tracing, https://arxiv.org/pdf/2006.16915v2.pdf

[20] Ampligraph, https://docs.ampligraph.org/en/1.3.1/

[21] Ampligraph Clustering Tutorial, https://github.com/Accenture/AmpliGraph/blob/master/docs/tutorials/ClusteringAndClassificationWithEmbeddings.ipynb

[22] Exercise Recommendation System (ExRec), https://files.eric.ed.gov/fulltext/ED599194.pdf

[23] Dynamic Key-Value Memory Networks for Knowledge Tracing, https://arxiv.org/abs/1611.08108.pdf

[24] A Self-Attentive model for Knowledge Tracing, https://arxiv.org/abs/1907.06837.pdf