VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
INFORMATIKOS KATEDRA

Transformerių tinklai dialogo būsenai sekti

Transformer Networks for Dialog State Tracking

Magistro baigiamasis darbas

Atliko: Lukas Cedronas (parašas)

Darbo vadovas: prof. dr. Olga Kurasova (parašas)

Recenzentas: prof. dr. Aistis Raudys (parašas)

Vilnius – 2023

Santrauka

Šiame magistro baigiamajame darbe tiriamas transformerių tinklų taikymas natūralios kalbos apdorojimo užduotims spręsti, konkrečiai – dialogo būsenos sekimo užduočiai. Darbas pradedamas dialogo būsenos radimo bei transformerių tinklų (bei jais paremtų modelių, tokių kaip BERT ir GPT) literatūros šaltinių apžvalga. Apžvelgus egzistuojančius sprendimus, darbe pasiūlomos naujovės – būsenos radimo tekste patobulinimai bei hibridinė architektūra – leidžiančios pasiekti geresnius rezultatus sprendžiant dialogo būsenos radimo uždavinį. Toliau aprašomi įvairūs bandymai, kurių metu išanalizuojami ir palyginami literatūros šaltiniuose bei šiame darbe pristatyti sprendimai. Bandymų metu tikrinamas įvairių giliojo mokymo metodų ir konfigūruojamų parametrų įtaka rezultatams. Darbo gale pateikiami rezultatai, išvados bei tolimesni darbai.

Raktiniai žodžiai: Transformerių tinklai, natūralios kalbos apdorojimas, dialogo būsenos sekimas, BERT, GPT

Summary

This master thesis focuses on the application of transformers in natural language processing (NLP), specifically in the task of dialog state tracking. The work begins with the overview of the literature of transformers (and their derivatives, such as BERT and GPT) and dialog state tracking problem. The thesis also presents novel approaches, including improvements in state identification from text and a hybrid architecture, aiming to achieve better results in dialogue state tracking. Through experiments and analysis, the study evaluates the proposed solutions, explores the impact of various parameters, and discusses future directions in this field.

Keywords: Transformers, Natural Language Processing, NLP, Dialog State Tracking, BERT, GPT

Turinys

Įvadas

Technologijos, leidžiančios žmogui kalbėti su kompiuteriu, šiais laikais yra bene kasdienybė. Visiems žinomi Amazon Alexa, Microsoft Cortana ir Siri – tai dialogo sistemos, ne tik gebančios kalbėti su žmonėmis, bet ir atlikti įvairias užduotis, pvz., sukurti priminimus, ieškoti įvairių dalykų internete ar atlikti rezervacijas. Dialogo būsenos sekimas (angl. *dialog state tracking*, šiame darbe kartais minimas kaip *DST*) yra vienas iš pagrindinių dialogo sistemų komponentų, leidžiančių saugoti bei atnaujinti dialogo būseną, reprezentuojančią sistemai žinomą informaciją apie dialogą, pvz., rezervacijų laikai, planuojami aplankyti restoranai ir panašiai.

Įvairūs virtualūs asistentai bei uždavinius atliekančios interaktyvios sistemos dažnai yra paremtos šiuolaikiniais natūralios kalbos apdorojimo (angl. *Natural Language Processing*) metodais. Tokie produktai kaip Alexa, Google Assistant ir pan. supranta žmogaus balsą ir jo sakomas komandas bei geba atlikti tam tikras užduotis. Tačiau dažnai to negana – pokalbyje atsiranda poreikis pasitaisyti suklydus, pratęsti savo ankstesnę mintį arba ką nors pasitikslinti, taip pokalbiui virstant dialogu. Sistemos, įgalinčios kompiuteriui turėti dialogą su žmogumi, vadinamos dialogo sistemomis.

Dialogo sistemas galima skirstyti į dvi dalis: pirmoji – tai sistemos, orientuotos į užduočių atlikimą, antroji – kitos, į užduočių atlikimą neorientuotos sistemos [McT21]. Į užduotis orientuotų dialogo sistemų, dar kitaip vadinamų virtualiais asistentais, pavyzdžiai yra Siri, išleista 2011 m., Amazon Alexa ir Microsoft Cortana, išleistos 2014 m. bei kitos, pvz., viešbučių rezervacijų sistemos ir pan. Į užduočių atlikimą neorientuotoms sistemoms galima priskirti pokalbių robotus ar kitokius virtualius asistentus [McT21], pvz., Microsoft Zo ar ChatGPT.

Dialogo būsenos sekimo problemos priskiriamos natūraliosios kalbos apdorojimo (NLP) problemų aibei. Pastaruoju metu NLP uždavinius pradėta spręsti naudojantis giliuosius neuroninius tinklus. Transformeriai, bei kiti iš jų išsivystę modeliai, pvz., BERT ar GPT, šiuo metu yra tekstinių problemų sprendimo standartas. Nenuostabu, jog transformeriai taip pat gali būti tinkami spręsti DST uždavinius – atlikta nemažai darbų (kai kurie jų pristatomi šiame darbe), kurie parodė, jog naudojantis transformeriais galima gauti tikslesnius rezultatus, nei kiti to meto populiarūs sprendimai. Tačiau transformerių, taipogi dialogų sistemų, tyrimo sritys yra milžiniškos – egzistuoja nemažai galimybių rezultatų gerinimui. Tad šio darbo siekis yra ištirti, kokios tos galimybės, bei atrasti naujas, jei jų yra.

Tikslai ir užduotys

Šio darbo tikslas yra išnagrinėti įvairias giliojo mokymo metodų bei natūralios kalbos apdorojimo technikas sprendžiant dialogo būsenos sekimo užduotį tranformeriais, bei pasiūlyti būdų pagerinti rezultatus, tai pagrindžiant atliekant įvairius bandymus.

Uždaviniai yra šie:

1. Išanalizuoti ir palyginti egzistuojančius dialogo būsenos sekimo metodus;
2. Įgyvendinti programą, leidžiančią atlikti bandymus su įvairiomis dialogo būsenos sekimo

būdais;

3. Įgyvendinti darbo eigoje rastus patobulinimus būsenos radimo tekste bei kandidatų generavimo architektūroms;

4. Sukurti architektūros, apjungiančios minėtus sprendimus, dizainą;

5. Palyginti minėtus sprendimus naudojantis tokias metrikas kaip nuostolių funkcijos, F funkcijos reikšmės apmokant bei žingsnio užklausos tikslumas.

Darbo struktūra

Darbas pradedamas dialogo sistemų pristatymu. 1.1 poskyryje glaustai pristatomas dialogo būsenos sekimo uždavinys. Taip pat pateikiami apibrėžimų, susijusių su dialogo būsenos sekimu, formalizavimas naudojantis žymėjimus iš aibių teorijos. Šio formalizavimo tikslas – padėti tvirtus pamatus tolimesniems darbams bei diskusijoms.

Literatūros analizė tęsiasi trumpa giliojo mokymo metodų apžvalga 1.2 poskyryje. Įtvirtinus giliųjų neuroninių tinklų pagrindus, pristatomi populiarūs būdai natūralios kalbos užduotims apdoroti. Galų gale 1.3 poskyryje prieinama prie transformerių – šiuo metu vienu iš populiariausių įrankių dirbant su natūralios kalbos apdorojimo užduotimis. Detalizuojama transformerių architektūra.

Paskutiniame literatūros apžvalgos 1.4 poskyryje pagaliau apjungiama dialogo būsenos sekimo bei transformerių teorija, pateikiant kelis konkrečius pavyzdžius, kaip transformerių tinklai naudojami spręsti šią problemą. Pristatomi darbai, kuriuose naudojami transformerių tinklai BERT ir GPT spręsti dialogo būsenos sekimo problemą.

Antroje darbo dalyje, aprašytoje 2 skyriuje, pristatomos galimos dialogo būsenos sekimo naudojantis BERT naujovės: būsenos radimo tekste patobulinimai bei hibridinė architektūra. Šiame darbe keliama hipotezė, jog šiame skyriuje apibūdinamos architektūros gali pasiekti tikslesnius rezultatus.

Trečiosios darbo dalies, aprašytos skyriuje 3, tema yra susijusi su išnagrinėtos literatūros sprendimų bei siūlomų naujovių įgyvendinimu sprendžiant dialogo būsenos sekimo problemą, išanalizuojant pateiktų sprendimų privalumus bei trūkumus. Atliekami eksperimentai keičiant įvairius parametrus, siekiant išsiaiškinti, kas turi didžiausią įtaką. Taikomos įvairios giliųjų mokymo metodų technikos bandant pagerinti rezultatus. Gauti rezultatai palyginami ir išanalizuojami. Galiausiai pateikiamos išvados bei tolimesni darbai.

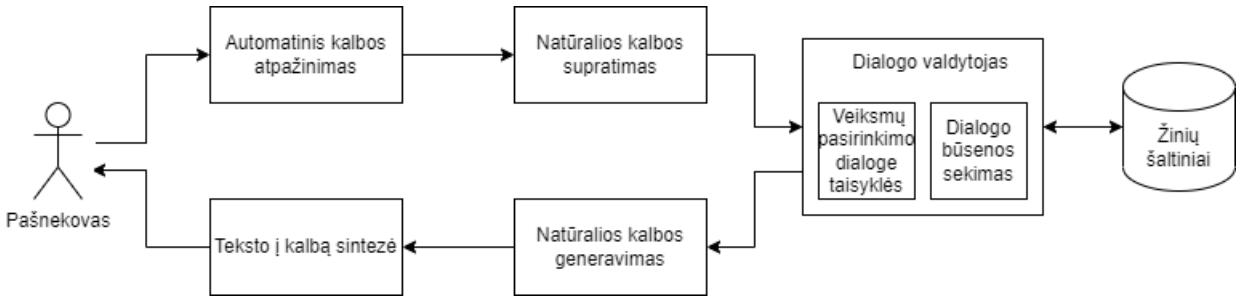
1. Literatūros apžvalga

Šiame skyriuje pateikta dialogo būsenos sekimo bei dirbtinių neuroninių tinklų, skirtų natūraliosios kalbos atpažinimui, literatūros šaltinių apžvalga.

Skyrius pradedamas bendru DST įvadu bei trumpai aptariami istoriniai statistikos metodais paremti sprendimai. Toliau pateikta giliųjų neuroninių tinklų, naudojamų spręsti įvairiems natūralios kalbos uždaviniams spręsti (tuo pačiu ir DST), apžvalga, nuo rekurentinių tinklų iki transformerių. Vėlesni poskyriai apžvelgia naujausius DST pasiekimus su giliųjų neuroninių tinklų modeliais.

1.1. Dialogų sistemos bei būsenos sekimas

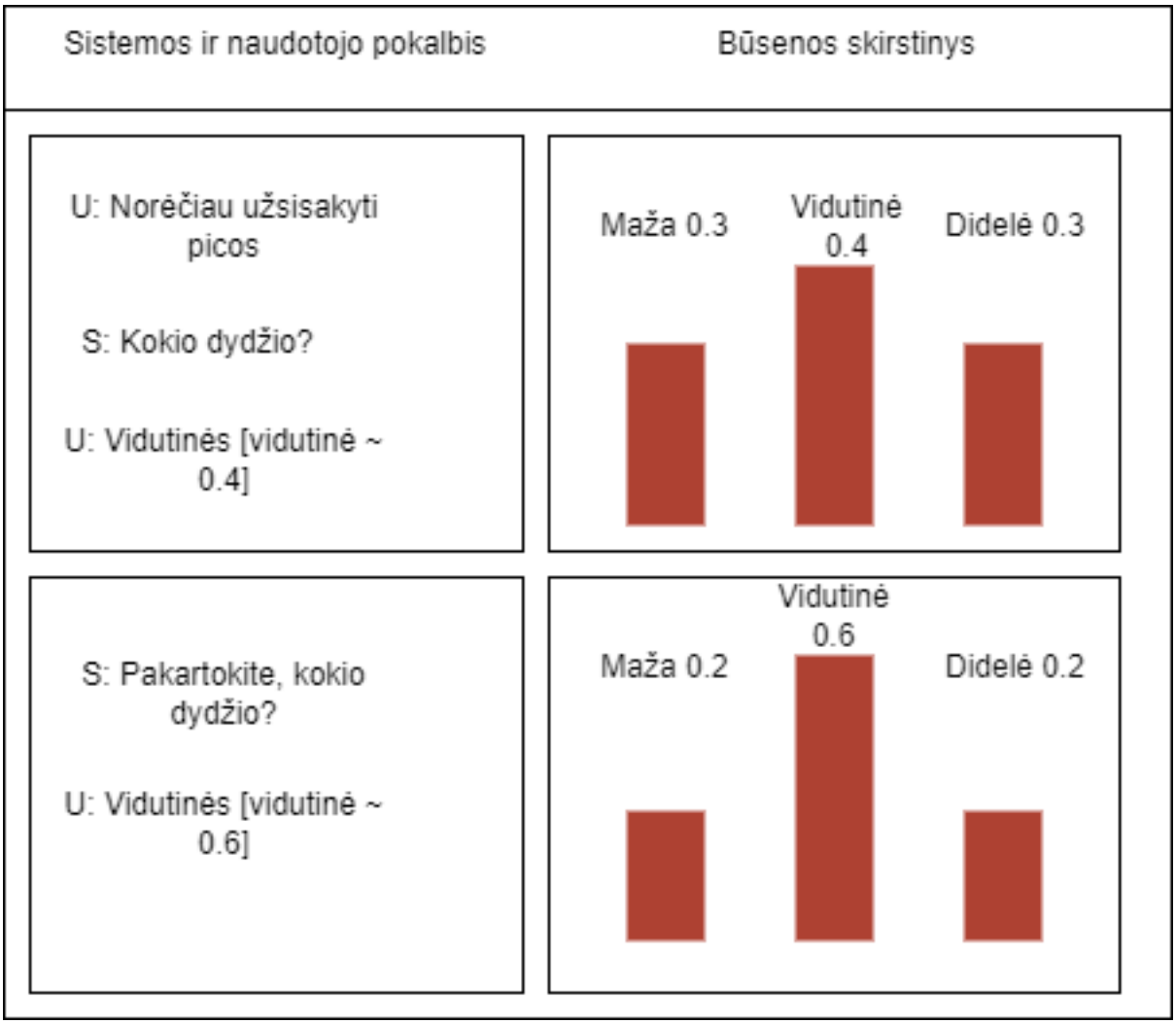
Dialogo būsenos sekimo – DST – komponentas yra esminė dialogo valdymo sistemų dalis, leidžianti suprasti pašnekovo išsakytus žodžius ir iš jų sukurti arba atnaujinti dialogo būseną [McT21]. DST komponentas yra itin svarbi sistemų, orientuotų į užduočių sprendimą (angl. *task-oriented systems*), dalis.



1 pav. Dialogo sistemų architektūra (paremta [McT21]).

DST komponentas saugo informacija apie dialogo būseną. Dialogo būsenoje konstruojama informacija, kurią sistema gavo iš pašnekovo dialogo metu. Ši informacija išreiškiama kintamaisiais (dar kitaip vadinamais angl. *slots*) ir jų reikšmėmis. Tačiau joks kalbos supratimo procesas negali garantuoti, jog kompiuteris teisingai suprato pašnekovą, todėl vietoje to yra saugomas būsenų skirstinys, kiekvienai būsenai priskiriantis tikimybę [WY07].

Vystantis dialogui, natūralios kalbos supratimo (NLU) metu gauta nauja informacija atnaujina būsenos tikimybes, siekiant gauti užtikrintą rezultatą, kaip matyti 2 paveikslėlyje. Vėliau šis būsenos rinkinys gali būti naudojamas dialogo valdytojui priimti sprendimą, kokį veiksmą atlikti toliau, pavyzdžiui, užsakyti kažkokią konkrečią picą.



2 pav. Būsenos kitimas vystantis dialogui (adaptuota pagal [McT21]).

Dažniausiai sutinkamos sąvokos DST kontekste yra šios:

- Kintamieji ir jų reikšmės (angl. *slot-values*) – žodžių poros, atitinkančios pašnekovo užklausą ir galimą reikšmę. Pavyzdžiui, *[wifi: pasiekiamas]*, kur *wifi* yra kintamasis, o *pasiekiamas* yra kintamojo reikšmė. DST siekia rasti teisingas reikšmes kintamiesiems dialogo eigoje.
- Tikslai (angl. *goals*) – konkretūs pašnekovo siekiai duotuoju momentu. Juos reprezentuoja kintamųjų-reikšmių aibė.
- Tikėjimas (angl. *belief*) – kintamojo reikšmių skirstinys, kitaip tariant, galimos reikšmės bei jų tikimybės.
- Tikėtina būsena (angl. *belief state*) – visų kintamųjų reikšmių skirstinių aibė.
- Tikėjimo sekimas (angl. *belief tracking*) – tikėtinos būsenos atnaujinimas dialogo eigoje.

Kai kuriuose formalesniuose DST apibrėžimuose, pvz. [HTW14a], kintamųjų [1] aibė būna fiksuota kiekvienam DST komponentui, taipogi ir jų galimos reikšmės. Šis sprendimas tinkamas nau-

[1] Čia ir toliau sąvoka *kintamieji* DST kontekste bus naudojama kaip angliško žodžio *slots* vertimas.

doti ribotais atvejais. Vėlesni proveržiai DST literatūroje parodo, jog galimi metodai, nevaržantys galimų kintamųjų reikšmių [XH18].

1.1.1. Statistiniai metodai dialogo būsenos sekimo užduočiai spręsti

Vieni iš pirmųjų bandymų automatizuoti pašnekovo siekių supratimą kompiuteriu rėmėsi statistiniais metodais. Pavyzdžiui, 1999 m. pristatyta Bajeso tinklais paremta architektūra modeliuoti pašnekovo intencijoms [HP99]. Autorių sukonstruotas Bajeso tinklas, modeliuojantis pašnekovo siekius kaip sąlygines tikimybes, iš abstraktesnių veda link tikslesnių pasinaudojus išgrynintomis sakinio charakteristikomis bei vizualinėmis užuominomis.

Sakiniai yra dekonstruojami į gramatines dalis; žodžiai yra asocijuojami su jau žinoma informacija iš žodynų. Visa ši informacija naudojama susiaurinti galimų tikslų aibę, siekiant rasti labiausiai tikėtiną pašnekovo siekį turint tam tikrą informaciją.

Šie ir vėlesni modeliai naudojosi pastoviu informacijos patikslinimu, siaurinant galimų siekių aibę ir galiausiai randant labiausiai tikėtinus siekius. Tačiau didžiausia tokių statistinių modelių problema yra ta, kad juos sukonstruoti yra būtinos NLP žinios. Sakiniai turėjo būti analizuojami ekspertų, kad būtų galima sukurti komplikuotus modelius.

1.1.2. Formalūs uždavinio apibrėžimai

Dialogo būsenos sekimo uždavinys literatūroje nėra itin nuosekliai formalizuotas, tad dažnai įvairiuose literatūros šaltiniuose pateikiamas skirtingas žymėjimas ir apibrėžimai. Vienas iš šio darbo siekių yra pasiūlyti būdų, kaip formalizuoti probleminę sritį bei pasiūlyti patogų žymėjimą tam, kad būtų galima sumažinti dviprasmiškumą ir glaustai aprašyti naudojamus algoritmus, su viltimi jog ateityje bus galima ne tik tiksliai aprašinėti modelių savybes, bet jas ir įrodyti. Šiame darbe remiamasi žymėjimu, pateiktu darbe [LCO21].

1.1.2.1. Probleminė sritis

DST algoritmų sprendžiama problema yra dialogo D būsenos B_t žingsnyje (angl. *turn*) $t \in \mathbb{N}$ reikšmių radimas. Dialogo būseną kiekviename žingsnyje nusako domenas, kintamieji ir jų reikšmės. Domenas (angl. *domain*), literatūroje dar kartais vadinamas siekiu (angl. *intention*), plačiai nusako dialogo temą. Pavyzdžiui, jei naudotojas pradeda šnekėti apie restoranus, domenas yra *restoranas*. Kintamieji (angl. *slots*) yra labiau specifiški, nei domenai – pavyzdžiui, jei domenas yra *restoranas*, kintamieji galėtų būti tai, kas aktualu naudotojui – *rezervacijos laikas* ar *vieta*. Kintamieji turi konkrečias reikšmes, pavyzdžiui, *12 val.*, *senamiestis*, ar pan. Reikšmės gali būti kategorinės (pvz., skrydžio klasė – ekonominė, verslo) arba ne (pvz., laikas, adresai ir pan.).

Tegu aibė $\mathcal{D}$ nusako visas galimas dialogo domeno (intencijų) reikšmes, $\mathcal{S}$ – kintamuosius, $\mathcal{V}$ – konkrečias kintamųjų reikšmes. Tuomet būsena tam tikru dialogo žingsniu t gali būti apibrėžta kaip funkcija $B_t : \mathcal{D} \times \mathcal{S} \to \mathcal{V}$.

Galimus aibių $\mathcal{D}, \mathcal{S}, \mathcal{V}$ elementus nusako ontologija arba schema. Ontologija O apibrėžia galimas reikšmes bei sąryšius tarp reikšmių, t.y. $\mathcal{O} \subseteq \mathcal{D} \times \mathcal{S} \times \mathcal{V}$. Jeigu ontologija egzistuoja ir tam

tikram uždaviniui ji yra apibrėžta, tai palengvina DST problemos sprendimą – tuomet galima bandyti iš visų įmanomų kombinacijų parinkti tinkamas reikšmes, kaip pvz. darbe [LTB+20]. Kartais ontologija yra neapibrėžta – vadinasi, iš anksto nėra žinomi sąryšiai tarp galimų reikšmių; taip pat aibės $\mathcal{D}, \mathcal{S}, \mathcal{V}$ gali būti begalinės (aibės elementus sudaro visi įmanomi žodžiai). Tam, kad funkcija B_t liktų apibrėžta visiems elementams, galime sakyti, jog jei duotajame žingsnyje t nežinome, kokia yra $B_t(d, s)$, kur $d \in \mathcal{D}$ ir $s \in \mathcal{S}$, reikšmė, tuomet yra priskiriama Ø.

Jei ontologija O apibrėžta vienam elementui iš aibės $\mathcal{D}$, į ją galima žiūrėti kaip į ontologiją, specifišką tam tikram iš anksto žinomam domenui; tokiais atvejais ontologija apibūdinama porų (s, v) aibe. Šiame darbe daroma būtent tokia prielaida – yra vienas specifinis domenas, tačiau nėra svarbu, koks jis, o sprendimai kai ontologija turi skirtingus domenus šiame darbe nėra nagrinėjami. Tuo tarpu būsenos funkcija laikoma $B_t : \mathcal{S} \to \mathcal{V}$.

1.1.3. Dialogo būsenos sekimo pavyzdys, naudojant formalizavimą

Toliau pateikiamas konkretus pavyzdys paprastumo dėlei. Tarkime, turime dialogą:

SISTEMA: Kuo galėčiau padėti?
NAUDOTOJAS: Norėčiau atlikti rezervaciją restorane Vilniaus gatvėje šiam vakarui

Ontologijos aibės šiame pavyzdyje yra apibrėžiamos taip:
$\mathcal{D} = \{restoranas\}$,
$\mathcal{S} = \{vieta, laikas\}$,
$\mathcal{V} = \{Vilniaus\ gtv., Kauno\ gtv., diena, vakaras\}$,
$\mathcal{O} = \{(restoranas, vieta, Vilniaus\ gtv.), (restoranas, vieta, Kauno\ gtv.),$
$(restoranas, laikas, diena), (restoranas, laikas, vakaras)\}$.

Būsena B_0 žingsnyje $t = 0$ yra $\forall d \in \mathcal{D}, \forall s \in \mathcal{S} : B_0(d, s) = \emptyset$. Būsena B_1 žingsnyje $t = 1$ yra $B_1(restoranas, vieta) = Vilniaus\ gtv.$, $B_1(restoranas, laikas) = vakaras$. Toliau darbe vietoje B_t bus rašoma B, jeigu dialogo žingsnio skaičius nėra svarbus ar nežinomas.

Rasti būsenos reikšmes reiškia apskaičiuoti B tam tikroms rūpimoms reikšmėms $d \in \mathcal{D}$ ir $s \in \mathcal{S}$. Tai dar vadinama kintamųjų užpildymo (angl. *slot-filling*) uždaviniu. Tačiau dažnai duotajame žingsnyje net nežinome, kokios yra rūpimos reikšmės d ir s – taigi, neaišku, kokius kintamuosius užpildyti. Tuomet sprendžiama dar ir kintamųjų radimo problema. Pavyzdžiui, naudotojui pasakius „noriu užsirezervuoti vietą kur nors vakare pavalgyti“, algoritmas turi apskaičiuoti eilės tvarka: d (domeną, t.y. kontekstą – šiuo atveju, matyt, „restoranas“), s – kintamąjį (šiuo atveju pateikta informacija apie vienintelį kintamąjį – laiką), ir tik tuomet galima apskaičiuoti $B(d, s)$ reikšmę, kuri šiuo atveju yra „vakaras“.

1.1.4. Tikslumo metrikos

Tam, kad išmatuoti, kiek tikslus yra spėjimas, yra keli būdai. Norint išmatuoti turimo modelio spėjimų tikslumą, naudojami **jungtinis siekio tikslumas** (angl. *joint goal accuracy*) bei **žingsnio užklausos tikslumas** (angl. *turn request accuracy*). Jungtinis siekio tikslumas yra skaičiuojamas tikrinant bendrą dialogo būseną kiekvieną žingsnį [HTW14a]. Tikroji būsena B_t ir spėjama būsena

B_p sutampa tada ir tik tada, kai sutampa visos būsenos reikšmės. Galima tai įsivaizduoti kaip dviejų sąrašų lyginimą: du sąrašai, turintys šimtą tokių pat įrašų, išskyrus vieną, bus nebevienodi.

Jungtinis siekio tikslumas yra itin griežta metrika dėl to, kad būsena vystosi dialogo eigoje. Jei modelis padaro bent vieną klaidą, visos tolimesnės būsenos bus nevalidžios. Paprastesnis būdas yra žingsnio užklausos tikslumas, kuris tikrina, kokia yra nauja dialogo būsena tam tikrame žingsnyje, nekreipiant dėmesio į bendrą būseną.

Tegu $P_t = 1\, if \frac{|slots\_and\_values_t|}{|slots\_and\_values_t \cap expected\_slots\_and\_values_t|} = 1, else\, 0$ žymi, ar spėjimas žingsnyje t yra teisingas (kitaip tariant, jei atspėtų dialogo kintamųjų ir jų reikšmių aibė ir tikroji aibė sutampa, $P_t = 1$). Tuomet jei t yra iš viso žingsnių dialoge (tuo pačiu ir atliktų eksperimentų), $\sum_{i=0}^{t} P_i$ yra žingsnių, kurių metu sistema pateikė teisingus spėjimus, skaičius. Jungtinis siekio tikslumas tuomet yra santykis $0 \leq \frac{\sum_{i=0}^{t} P_i}{t} \leq 1$.

Kintamojo siekio tikslumas yra santykis tarp teisingai atspėtų bei visų s_i, v_j reikšmių rinkinyje tam tikram s_i. Vidutinis siekio tikslumas yra visų šių santykių vidurkis.

F reikšmė. Lyginant modelius, šiame darbe dažnai naudojama F reikšmė (konkrečiau – F2). Ši dažnai mašininio mokymo srityje naudojama funkcija yra pasirinkta todėl, kad apjungia jautrumą (angl. *recall*) ir preciziškumą (angl. *precision*). Preciziškumas, jautrumas ir tikslumas – dažnai naudojamos metrikos modeliui įvertinti. Tikslumas parodo, koks yra teisingų spėjimų santykis su visais spėjimais. Tačiau tuomet, kai duomenų rinkinys yra nesubalansuotas – pavyzdžiui, labai daug neigiamų pavyzdžių ir retai pasitaikantys teigiami – tikslumas rodo gan iškraipytus rezultatus: modelis gali niekada nepataikyti į teisingą spėjimą, bet jeigu jis visą laiką atspės neigiamus pavyzdžius, tikslumas bus didelis. Panašios problemos kyla ir su jautrumu, preciziškumu ir kitomis metrikomis. Tolimesnė lentelė pateikia problemos iliustraciją.

1 lentelė. Jautrumo, tikslumo, preciziškumo ir F funkcijos reikšmės palyginimas

Metrika	Tikrasis žymuo	Spėjamas žymuo	Metrikos reikšmė	Komentaras
Tikslumas	0 0 0 0 0 1	0 1 0 0 0 1	0,833	Visiškai prašauta viena teigiamų reikšmių, bet tikslumas gan nemažas
Preciziškumas	0 0 0 0 0 1	0 1 0 0 0 1	0,5	Metrika nepakankamai atsižvelgia į neigiamus spėjimus
Jautrumas	0 0 0 0 0 1	0 1 0 0 0 1	1,0	Iš metrikos atrodytų, jog spėjimai – idealūs
F1	0 0 0 0 0 1	0 1 0 0 0 1	0,66	Labiausiai subalansuota reikšmė

F reikšmė apjungia šias metrikas ir grąžina bendresnę reikšmę, subalansuodama preciziškumą ir jautrumą. F1 nuo F2 skiriasi tuo, kad F2 atveju, preciziškumas svarbus mažiau. Visos F_n funkcijos, kur $n \in NAT$ pasižymi tuo, jog kuo didesnis n, tuo didesnis dėmesys skiriamas teigiamų pavyzdžių atspėjimui. Jei $n \to$ inf, F funkcija tampa panaši į jautrumo funkciją.

$$F_2 = 5 * \frac{(precision * recall)}{(4 * precision) + recall} \quad (1)$$

Šiame darbe F funkcija skaičiuojama surinkus klasių tikimybes, jas suapvalinus link 1 arba 0 bei palyginus gautą vektorių su klasės tikrųjų reikšmių vektoriumi (kurio reikšmės yra 1-etai arba 0-iai).

1.2. Gilusis mokymas natūraliosios kalbos apdorojime

Toliau glaustai pristatomi gilieji neuroniniai tinklai – pagrindas, ant kurio yra įgyvendinti šiame darbe naudojami modeliai.

1.2.1. Gilieji neuroniniai tinklai

Šiame darbe naudojamų modelių (pvz., BERT) architektūros pagrindai yra neuroniniai tinklai. Neuroninio tinklo idėja įkvėpta biologijos: perceptronas, iš kurių susideda tinklai, yra supaprastintas matematinis neurono modelis.

Kiekvieno perceptrono funkcija yra priimti tam tikrą įvestį, pritaikyti tam tikrą funkciją (vadinamą aktyvavimo funkciją) visoms gautoms įvestims bei grąžinti išvestį. Perceptronai sudaro sluoksnius, ir paprasčiausiuose tiesioginio sklidimo tinkluose vienas sluoksnis perduoda savo išvesčių vektorių kitam po jo einančiam sluoksniui.

Tinklas turi įvesties sluoksnį, n vidinių sluoksnių bei išvesties sluoksnį. Taigi, informacija tiekiama per visą tinklą nuo įvesties iki išvesties imituojant biologinį neuroninių impulsų veikimą smegenyse (signalas, informacijos apdorojimas ir reakcija).

Sudėtingi modeliai, kai tinklai turi itin daug sluoksnių bei įvairiai sąveikaujant vienas su kitu gali spręsti sudėtingas problemas (pvz., teksto supratimas, paveikslėlių generavimas) vadinami gilieji neuroniniai tinklai, jų sprendžiamos užduotys – gilusis mokymas. Transformeriai susideda iš daug skirtingų dalių – enkoderių, dėmesio sutelkimo sluoksnių. Tačiau į kiekvieną iš sudedamųjų dalių galima žiūrėti kaip į neuroninį tinklą, galintį turėti daug vidinių sluoksnių.

Į neuroninius tinklus patogu žiūrėti kaip į matricas ar tensorius, kadangi tuomet tai leidžia atlikti įvairias tiesinės algebros apibrėžtas operacijas. Tuomet modelis apibūdinamas parametrais W ir b. Čia W yra svorių matrica. Reikšmė W_ij apibūdina jungties tarp perceptronų i, j „stiprumą": kuo didesnė reikšmė, tuo jungtis stipresnė, tuo tarpu mažesnė reikšmė reikštų jog informacija (impulsas) tarp i ir j sunyksta. Parametras b – vadinamas slenksčiu – tai vektorius, kur b_i apibūdina, kokio stiprumo signalo reikia i-tąjam perceptronui, kad jis būtų aktyvuotas. Jei b_i reikšmė neigiama, vadinasi, reikia didelių svorių W arba stipraus signalo x.

1.2.2. Modelių apmokymas

W ir b yra konfiguruojami neuroninio tinklo parametrai. Pakeitus tam tikrus svorius ar slenksčius, gaunamas vis kitoks rezultatas. Dažnai norima išgauti tam tikrą rezultatą, kartais vadinamu tikslu. Skirtumas tarp tikslo ir išvesties vadinama paklaida. Apmokant neuroninius tinklus, sie-

kiama sumažinti bendrą modelio paklaidą. Modelio parametrai gali būti konfiguruojami pagal tai, kokia yra ta paklaida. Atgalinio sklidimo algoritmas leidžia apskaičiuoti, kaip paklaida keičiasi, keičiant parametrus. Šis algoritmas kiekviename sluoksnyje apskaičiuoja nuostolių funkcijos išvestinę.

Patys parametrai yra atnaujinami naudojantis gradientinio nusileidimo algoritmu. Žinant paklaidos funkcijos išvestinę, galima pasakyti, kaip ji greitai keičiasi, ir į kurią pusę didėja, o į kurią – mažėja. Paprasčiausia įsivaizduoti dvimatėje erdvėje, kai yra vienas kintamasis: jei norime atnaujinti x reikšmę, randame funkcijos išvestinę pagal x. Tada jei matome, kad tam tikrame fiksuotame taške išvestinės reikšmė yra neigiama, prie x reikšmių pridėję teigiamą reikšmę (nes žinome kad funkcija mažėja didinant x) priartėsime prie funkcijos minimumo. Jei išvestinės reikšmė teigiama, tada darome atvirkščiai.

Kaip pavyzdys, BERT, kuris susideda iš kelių komponentų kurių visi turi milijonus parametrų, yra apmokomas labai panašiai. Dažnai BERT turi galutinį klasifikavimo sluoksnį – tai gali būti kelios (arba viena) klasės, pavyzdžiui, teksto emocija. Tuomet išvestis būtų tikimybė, jog tekstas turi tam tikrą emociją. Vadinasi, žinodami tikrąją klasę, galime rasti paklaidą ir taikyti tą patį atgalinio sklidimo algoritmą.

1.2.3. Apmokymo metodai

Kartais naudojami įvairūs metodai, leidžiantys apmokymą įvairiais būdais pagerinti, pvz., jį paspartinti. Apmokymo metu kiekvienas įrašas paduodamas į neuroninį tinklą siekiant išgauti rezultatą ir apskaičiuoti klaidos funkciją. Tačiau šį procesą galima paspartinti: dažnai keli įrašai sugrupuojami į vieną rinkinį, angl. *batch*. Svarbu paminėti, jog kuo didesnis rinkinys, tuo daugiau atminties reikia, norint rinkinį vienu ypu apdoroti.

Atsitiktinio praretinimo transformacija. Tai – vienas reguliarizacijos metodų, leidžiantis išvengti persimokymo. Apmokant su tam tikra tikimybe atsitiktinai parinkti neuronai yra ignoruojami. Originaliame BERT straipsnyje aprašomas šios technikos naudojimas visuose sluoksniuose.

Duomenų aibių išskyrimas. Tam, kad išvengti permokymo, gali būti naudojamas apmokymo, testavimo ir validacijos išskyrimas. Modelis yra apmokomas tik naudojantis apmokymo aibe. Apmokius modelį vieną kartą su visais duomenimis (kitaip tariant, po vieno epizodo), nuostolių funkcija tikrinama su visais duomenimis iš validacijos aibės. Šis metodas gali padėti nustatyti, kada modelis persimoko: duomenų iš validacijos aibės nėra apmokymo aibėje, tad persimokymo metu validacijos nuostolių funkcija gali pradėti didėti, taip parodant, jog modelis geba teisingai klasifikuoti tik apmokymo aibės elementus, tačiau prastai klasifikuoja elementus iš kitų aibių.

Žinių pernešimas. Šiame darbe naudojamas žinių pernešimas, angl. *knowledge transfer*. Tai technika, kai tam tikras modelis, apmokytas spręsti kažkokią užduotį, yra apmokomas spręsti kitą, bet panašią užduotį, galimai – su visiškai kita duomenų aibe, tikintis, kad bus pasiekti geresni rezultatai. Šios technikos bandymai su modeliais pasitelkiant žinių pernešimą aprašyti 3.3.5.4 poskyryje.

Gradientų kaupimas. Kartais apmokant didelį modelį nepavyksta nustatyti didelės rinkinio (batch) reikšmės dėl resursų stokos. Į pagalbą galima pasitelkti gradientų kaupimo metodą. Jo metu

gradientai yra saugomi n iteracijų ir nuostolių reikšmės atnaujinamos tik n-tąją iteraciją. Tai yra beveik ekvivalentu rinkinio sudarymo (angl. *batch*) metodui, ko pasekoje kuo didesnė parametro reikšmė, tuo apmokymas labiau sutrumpėja.

1.2.4. Tinklai natūraliai kalbai apdoroti

Ilgainiui statistinius taisyklėmis paremtus metodus sprendžiant NLP uždavinius pakeitė gilieji neuroniniai tinklai, pvz., RNN [MST+15]. Šis poskyris skirtas apžvelgti svarbiausių modelių, skirtų spręsti NLP uždavinius, raidą. Kiekvienas atradimas – egzistuojančių modelių ir metodų patobulinimas, tad norint tinkamai suprasti moderniausius architektūrinius giliojo mokymo sprendimus naudinga žinoti, kas iki jų atvedė.

Rekurentiniai neuroniniai tinklai. Ilgą laiką literatūra apie RNN buvo gan negausi, kaip atkreipiama dėmesį I. Sutskever ir kt. [SMH11]. Anot autorių, viena iš priežasčių – nykstančių gradientų problema (angl. *vanishing gradients problem*). RNN vystantis toliau, ši problema buvo pažabota, bei 2011 m. buvo pademonstruota, jog tokie tinklai puikiai tinka teksto generavimui [SMH11].

RNN – tai dirbtinis neuroninis tinklas, kuris leidžia modeliuoti duomenų seką. RNN duomenų įvesčiai (angl. *input*) ne tik grąžina atsakymą, bet jį išsaugo taip vadinamoje paslėptoje būsenoje (angl. *hidden state*). [SMH11] autoriai pateikia tokią RNN formuluotę:

$$h_t = tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h), \tag{2}$$

$$o_t = W_{oh}h_t + b_o. \tag{3}$$

Čia x_t įvestis laiko momentu t, h_t – paslėpta RNN būsena laiko momentu t, W_hx – matrica, reprezentuojanti svorius tarp įvesties ir vidinių sluoksnių, atitinkamai W_hh vidinių sluoksnių svorių matrica, W_oh – išvesties ir vidinių sluoksnių svorių matrica, b_h ir b_0 yra poslinkio (angl. *bias*) parametrai.

Taigi, iteruojant duomenų seką, vidinė būsena h_t yra vis atnaujinama su $t-1$-ąją RNN išvestimi bei panaudojam t-jam rezultatui gauti.

Rekurentiniai tinklai buvo sėkmingai pritaikyti ir DST srityje [HTY14; MST+15].

Enkoderiai ir dekoderiai. Kaip žinia, skirtingos kalbos tą patį dalyką gali išreikšti skirtingo ilgio sakiniais (pvz. 5 žodžius turintis angliškas sakinys „I do not want to“ į lietuvių kalbą gali būti išverstas į paprasčiausiai vieno žodžio sakinį – „nenoriu”). Tai sukelia problemų kalbos atpažinimo ar teksto vertimo kompiuteriu uždavinius spręsti RNN architektūra – dažnai paprasto RNN tinklo nepakanka, kadangi tradicinio RNN tinklo įvesties bei išvesties sekų dydžiai yra vienodi.

Būdas, patobulinantis RNN (bei LSTM) architektūrą gebėjimu spręsti uždavinius, kai įvesties bei išvesties sekos gali būti skirtingos, buvo pristatytas 2014 m. [CMG+14], lygiagrečiai [BCB14a] autoriams panaudojant dėmesio (angl. *attention*) mechanizmą kalbų vertime neuroniniais tinklais. Iš esmės pasiūlyti du modeliai: vienas – suspaudžiantis seką į tam tikros fiksuotos dimensijos vektorių, kitas – iš vektoriaus pagaminantis bet kokią kitokią seką. Tradiciškai, „suspaudžiantis“ modelis

vadinamas enkoderiu (angl. *encoder*), „išplečiantis“ – dekoderiu (angl. *decoder*).

Enkoderis įvesties seką transformuoja į vektorių c, vadinamą *kontekstu*. Šis vektorius yra galutinė vidinė būsena laiko momentu t [CMG+14]:

$$c = h_t = f(h_{t-1}, x_t). \tag{4}$$

Ši formulė yra 2-osios formulės bendresnė forma: h_t, x_t reiškia tą patį, f – bet kokia netiesinė aktyvavimo funkcija. Tuo tarpu dekoderio vidinė būsena priklauso nuo šio konteksto c bei išvesties y_{t-1}:

$$h_t = f(h_{t-1}, y_{t-1}, c). \tag{5}$$

Dekoderio išvestis y_t priklauso nuo išvesties vienu žingsniu atgal y_{t-1} bei būsenos h_t. Enkoderis ir dekoderis mokomi kartu, siekiant bet kokiai įvesties sekai x rezultate pateikti seką y.

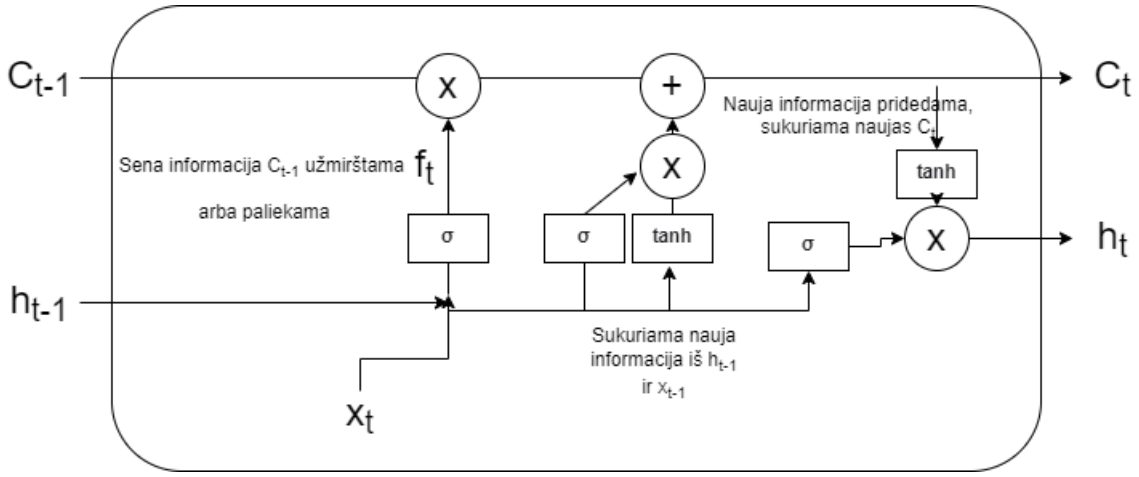
[CMG+14] autoriai parodė, jog enkoderiai-dekoderiai lenkia tradicinius statistikos metodais paremtus kalbų vertimo modelius, kurių rezultatai labai priklauso nuo to, kiek dažnai tam tikros frazės yra sutinkamos tekste. Teisingai išversti retas frazes statistiniais metodais paprasčiausiai neužtenka duomenų, tačiau RNN enkoderiai/dekoderiai yra sėkmingesni atrandant lingvistinius požymius, lemiančius sakinių struktūrą, tad jie nėra apriboti žodžių dažniu tekste.

Ilgoji trumpalaikė atmintis. Ilgosios trumpalaikės atminties, vadinamos *LSTM* (angl. *long short-term memory*), architektūra, pristatyta dar 1997 m., yra atsakas nykstančių bei sprogstančių gradientų RNN tinkluose problemai [HS97]. Žemiau yra trumpai paaiškinama gradientų nykimo bei sprogimo problema.

Apmokant gilius neuroninius tinklus (pvz., RNN) sklidimo atgal (angl. *back-propogation*) būdu klaidos funkcijos reikšmės yra linkusios arba artėti link 0, arba link begalybės. Tai gali sukelti problemų mokymo metu atnaujinant svorių reikšmes – jei skirtumas, gautas apskaičiavus klaidos funkcijos priklausomybę nuo svorių (gradientas) yra nykstamai mažas, svorių reikšmės beveik nepasikeičia. Jei gradientas didėja link begalybės, svorių reikšmės pakis itin smarkiai – galimai niekada nepasiekdamos optimalių reikšmių.

Gradiento reikšmės nyksta arba sprogsta, kai klaidos funkcijos kitimas skaičiuojamas daug kartų – pvz., neuroniniui tinklui turint daug sluoksnių. Tuomet gradientas W po n žingsnių taps proporcingas W^n. Jei $W < 1$, W^n artės prie 0. Jei $W > 1$, W^n artės link begalybės.

Pagrindinis LSTM architektūros tikslas – išlaikyti gradientą kiek galima pastovų. [HS97] pristatyta architektūra RNN tinklų vidinę būseną pakeičia LSTM elementu, arba ląstele (angl. *cell*). Šis elementas neleidžia būsenai per daug keistis „apsaugant“ ją nuo neaktualių pokyčių, tuo pačiu „apsaugo“ kitas būsenas nuo neaktualių pokyčių iš šios būsenos.



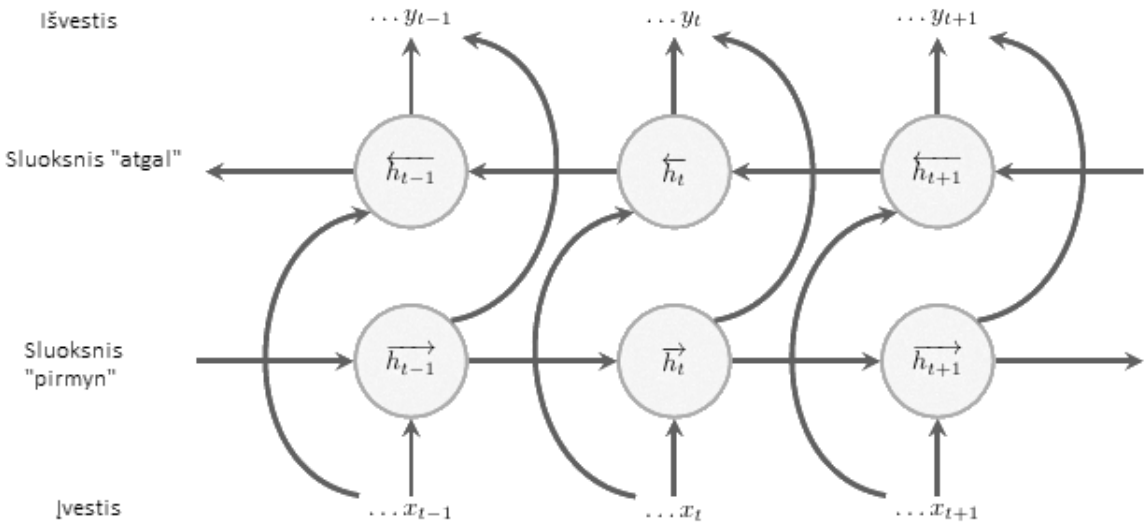
3 pav. LSTM pavyzdys (adaptuota pagal [Ola15]).

Į LSTM kamerą laiko momentu t paduodama būsena h_{t-1}, kameros išvestis c_{t-1} bei įvestis x_t. LSTM apibrėžia vartus, į kuriuos patekusi informacija arba paliekama, arba užmirštama. Įvairios operacijos su būsena, įvestimi bei praeitos kameros išvestimi leidžia „užmiršti“ nereikalingą informaciją bei kontroliuoti, kaip keičiasi būsena h_t.

Nors LSTM buvo sukurti kaip sprendimas nykstančių gradientų problemai, ši architektūra ilgainiui pakeitė paprastus RNN, kadangi LSTM geba apdoroti milžiniškas sekas, turinčias daug informacijos – pvz., vaizdo įrašus – kai tuo tarpu RNN tinklai dėl savo trumpos atminties (kylančios dėl nykstančių gradientų problemos) nesugeba susidoroti su plačiai pasklidusia informacija. Turint ilgą tekstą tikėtina, jog paprastas RNN tinklas nesugebės savo vidinėje būsenoje išlaikyti informacijos priklausomybės tarp pirmų bei paskutinių žodžių.

LSTM plačiai pradėti naudoti kaip alternatyva RNN. Pavyzdžiui, LSTM sukombinuoti kartu su enkoderiais-dekoderiais [SVL14] buvo itin sėkmingi sprendžiant vertimo iš anglų k. į prancūzų k. užduotį – su šia architektūra pavyko pasiekti iki tol geriausius rezultatus. Modelis ne tik puikiai gebėjo sukurti bet kokio ilgio tekstą nepriklausomai nuo įvesties ilgio, bet taip pat demonstravo puikius rezultatus ilgų tekstų vertime, taip pagrindžiant LSTM ilgalaikės atminties efektyvumą.

Abipusiai rekurentiniai neuroniniai tinklai. Apdorodami duomenis nuosekliai, tradiciniai RNN naudojasi tik prieš tai buvusia informacija. Buvo pastebėta, jog tam tikroms problemoms spręsti verta duomenis skaityti ne tik iš priekio, bet ir iš galo tuo pačiu metu, pavyzdžiui, kalbos atpažinime [GJM13].



4 pav. Abipusių RNN architektūra (adaptuota pagal [GJM13])

4 pav. pateiktas abipusių RNN pavyzdys. Į abipusius RNN tinklus galima žiūrėti lyg į du RNN tinklus, apdorojančius paduotą informaciją iš skirtingų pusių. Taigi, tam tikru laiko momentu t turės informaciją apie paslėptą būseną ne tik iš h_{t-1}, bet ir h_{t+1}. Apdorojant ilgą sakinį, RNN informaciją apie tam tikrą fiksuotą žodį išgaus ne tik iš žodžių sekos, esančios nuo sakinio pradžios iki žodžio, bet ir iš sekos nuo fiksuoto žodžio iki sakinio pabaigos.

Abipusius RNN galima sukombinuoti su LSTM, taip išgaunant geresnius rezultatus, nei naudojantis vienpusiu RNN su LSTM sprendžiant kalbos atpažinimo uždavinius [GJM13].

Dėmesio mechanizmas. Iki 2015 m. teksto suvokimo kompiuteriu problema buvo sunkiai įkandama – dažnai iki tol sprendimai būdavo paremti galybe žmonių sukonstruotų taisyklių; jų veikimas būdavo apribotas vos keliais šimtais veikiančių tekstinių pavyzdžių. Didelis perversmas įvyko teksto suvokimo problemos sprendimui pritaikius dėmesio mechanizmą [HKG+15].

Nors RNN tinklai (kartu su LSTM) dėl gebėjimo saugoti būseną yra tinkami spręsti užduotis, RNN turi ir trūkumų, dėl kurių, kaip parodė [BCB14a], ne visada tinkami spręsti sudėtingas natūralios kalbos supratimo užduotis. Anot autorių, pagrindinė RNN bei jais paremtų enkoderių/dekoderių problema – tai, jog įvesties vektoriaus dydis yra fiksuotas, kas reiškia, jog jeigu įvestis yra ilgas tekstas, visa informacija bus „suspausta“ į fiksuoto dydžio vektorių, kas gali vesti link informacijos praradimo.

Autoriai [BCB14a] pasiūlė mechanizmą, kurio metu iš įvesties duomenų sekos yra parenkami svarbiausi elementai, turintis didžiausią įtaką išvesčiai, kitaip nei RNN enkoderis-dekoderis, kuris generuodamas išvestį surenka informaciją iš visos įvesties. Kitaip tariant, dėmesio mechanizmas išskiria svarbesnes dalis, kurios nulemia atsakymą.

Kiekvienam i-tajui modelio išvesties sekos elementui y_i apibrėžiamas *konteksto vektorius* c_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \tag{6}$$

h_i yra informacija apie i-tąjį įvesties žodį. Šis daugiklis yra dvikrypčių RNN paslėptų būsenų

jungtis (RNN būsenoje saugoma informacija apie prieš tai buvusią įvestį, tuo tarpu dvikrypčių RNN būsenoje – įvesties iš abiejų pusių). Nors ir h_i saugoma informacija apie visą įvestį, kintamasis vis tiek labiausiai atspindi i-tąjį įvesties sekos elementą dėl RNN savybių, lemiančių ilgalaikį istorinės būsenos nykimą.

α_{ij} yra svoris, nulemiantis, kiek svarbus h_j yra išvesčiai y_i. Šių svorių suma yra 1. Šis svoris suskaičiuojamas radus visus sąryšius e_{ij} tarp enkoderio įvesties informacijos h_j ir dekoderio s_i, kitaip tariant – rūpimo išvesties žodžio, ir padalinus iš jų visų sumos:

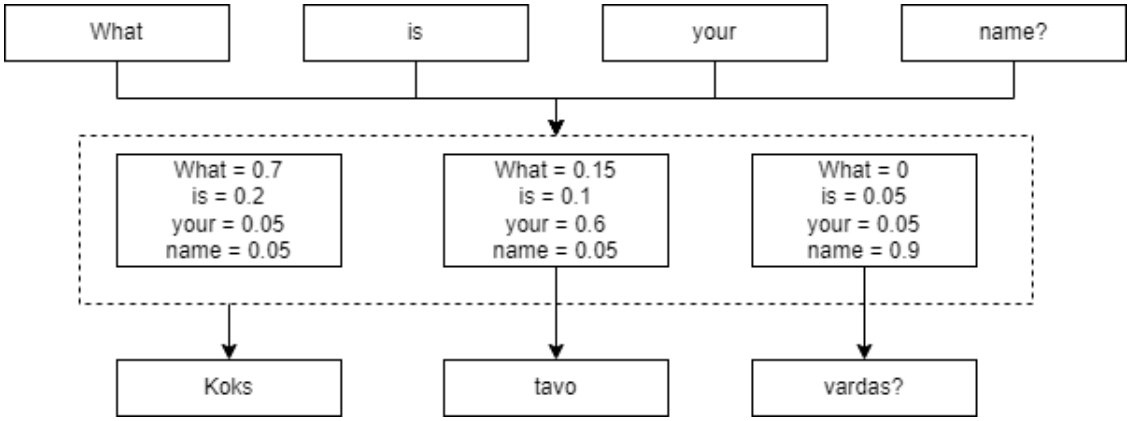
$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_k exp(e_{ik})}. \tag{7}$$

e_{ij} rasti gali būti naudojamas papildomas neuroninių tinklų modelis, išmokytas rasti sąryšį tarp s_i ir h_j:

$$e_{ij} = a(s_i, h_j), \tag{8}$$

kur a – funkcija, reprezentuojanti neuroninį tinklą.

Taigi, vietoje užkoduoto fiksuoto dydžio konteksto gaunami kontekstai, laikantys svarbiausią informaciją. Ši informacija perduodama dekoderiui, kuris nusprendžia, į ką kreipti dėmesį generuojant išvestį.



5 pav. Dėmesio mechanizmo pavyzdys.

5 pav. pateiktas supaprastintas dėmesio mechanizmo pavyzdys. Sakinys „What is your name?“ yra verčiamas į lietuvių kalbą. Tarp įvesties ir išvesties esantis dėmesio mechanizmo sluoksnis kiekvienam galimam i-tajam lietuviškam žodžiui pasirenka aktualiausią anglišką žodį pagal didžiausią i-tojo dėmesio vektoriaus reikšmę. Pvz., idant išgauti pirmajam lietuviško sakinio žodžiui didžiausias dėmesys teikiamas pirmajam angliškam žodžiui.

[HKG+15] autoriai sukūrė ir palygino kelis modelius, gebančius atsakyti į užduotą klausimą. Užduoties atsakymas yra suformuluotas kaip tikimybė, jog tam tikras žodis yra atsakymas:

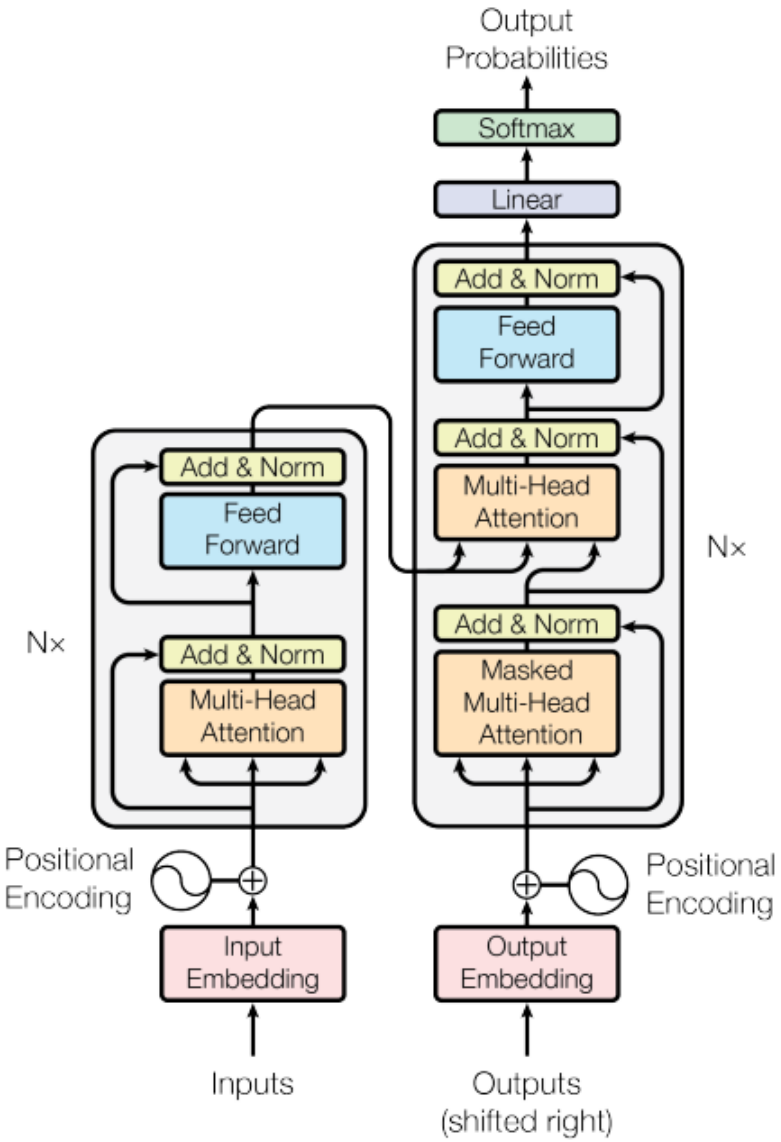
$$p(a|d,q) \propto exp(W(a)g(d,q)) \tag{9}$$

čia q – užduotas klausimas (angl. *query*), a – atsakymas, d – dokumentas, kuriame žodis yra ieškomas, W – svorių matrica, $g(d,q)$ – funkcija, kuri dokumentą bei užklausą paverčia į vektorinės

formos įvestį.

1.3. Transformerių architektūra

Šiame poskyryje bus išsamiau išnagrinėta transformerių architektūra, remiantis [VSP+17] straipsniu, suskaidžius modelį į svarbiausias dalis ir į jas pažvelgus detaliau.



6 pav. Transformeriaus architektūra. Diagrama iš [VSP+17]

Enkoderis ir dekoderis. Transformerių architektūra yra paremta tuo pačiu enkoderio bei dekoderio modeliu, panašiai į RNN tinklais įgyvendintą enkoderio-dekoderio architektūrą.

Esminis skirtumas tarp transformerių enkoderio-dekoderio architektūros ir RNN enkoderio-dekoderio architektūros yra tas, jog transformeriai neturi vidinės būsenos mechanizmo, reikalaujančio duomenis apdoroti paeiliui, vienas po kito.

Enkoderis sakinio reprezentaciją $(x_1, ..., x_n)$ užkoduoja į $z = (z_1, ..., z_n)$, o dekoderis naudodamasis z sugeneruoja $y = (y_1, ..., y_n)$.

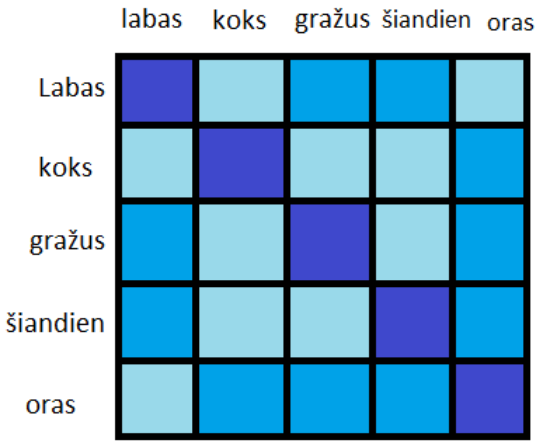
Žodžių šablonai. Kiti keli svarbus transformerių sluoksniai yra skirti tam tikros informacijos apie žodį užkodavimui vektoriuje – šablonų (angl. *embedding*) sudarymui. Kiekvienas žodžio vektorius yra sudaromas taip, jog panašių žodžių vektorių reprezentacija taip pat turėtų didelį panašumą (žr. *Input Embedding* sluoksnį). Negana to, vektoriuose taip pat saugoma informacija apie jų vietą sakinyje (žr. *Positional Encoding* sluoksnį). Tai reikalinga tam, kad išlaikyti sakinio struktūrą (žodžio vieta palyginus su kitais žodžiais yra svarbi kalboje). Šio sluoksnio taip pat užtenka pakeisti rekurentinių tinklų vidinę būseną.

Dėmesys bei savidėmesys. Transformerių architektūroje randamas *savidėmesio* (angl. *self-attention*) sluoksnis. Dėmesio mechanizmas naudojamas transformeriuose labai panašus į jau anksčiau pristatytą mechanizmą [BCB14a].

Savidėmesys (angl. *self-attention*) – tai procesas, kurio metu dėmesio reikšmės yra randamos tarp žodžių tame pačiame sakinyje. Savidėmesio intuicija yra tokia: kiekvienas žodis sakinyje yra daugiau ar mažiau susijęs su kitais. Vienas žodis gali būti labai susijęs su kitu, pavyzdžiui, nuo kitų žodžių gali priklausyti net tam tikro žodžio reikšmė. Pvz, sakinys: „planuoju kitąmet vesti“ žodžio „vesti“ reikšmė gali būti drastiškai skirtinga priklausomai nuo to koks žodis eina vėliau: „planuoju kitąmet vesti paskaitas“ arba „planuoju kitąmet vesti sužadėtinę“, tuo tarpu kiti žodžiai – planuoju, kitąmet nėra tokie svarbūs.

Savidėmesio mechanizmas kiekvieną žodį paverčia į vektorių, kuriame atsispindi, kiek kiekvienas kitas įvesties žodis yra svarbus jo reikšmei. Naudojantis šia informacija, dekoderis geba generuoti žodžius iš vektorių, turinčių visą kontekstinę informaciją.

Enkoderis bando rasti panašumą tarp visų įvesties žodžių tarpusavyje. Visų žodžių vektorinės formos (šablonai) yra sudauginamos vienos su kitomis, o gautai matricai pritaikoma $softmax$ funkciją, siekiant išvengti per didelių reikšmių. Taip gaunama taškų (angl. *score*) matrica.



7 pav. Žodžių tarpusavio panašumo matrica.

7 paveikslėlis reprezentuoja žodžių tarpusavio panašumo matrica. Kuo tamsesnė spalva, tuo žodžiai panašesni.

Naudojantis šia matrica kiekvieną žodį galima reprezentuoti kaip kitų žodžių vektorine suma. Jei kiekvieną matricos reikšmę laikysime kaip svorį (skaliarinė reikšmė nuo 0 iki 1), mažiau svarbūs žodžiai turės mažesnį svorį.

Autoriai įveda tris naujus terminus: užklausa (angl. *query*), raktas (angl. *key*) ir reikšmė (angl. *value*), taip dėmesio mechanizmą sugretindami su duomenų bazių užklausų bei reikšmių gavimo procesu. Užklausa tėra rūpimas vektorius, pavyzdžiui, praeitas dekoderio sugeneruotas žodis, raktas – į užklausą panašiausias žodis, reikšmė – vektorius, saugantis kontekstinę informaciją apie žodį. Transformeriuose, raktą ir reikšmę sukuria enkoderis, užklausą – dekoderis.

Apibendrintai, dėmesys gali būti išreiškiamas šia formule:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V, \tag{10}$$

čia Q – užklausa, K – raktas, V – reikšmė, d_k – rakto dimensija.

Daugialypis dėmesys. Į bet kokį duotą žodį sakinyje galima žiūrėti iš kelių perspektyvų. Pavyzdžiui, turint žodį „vienas“, mus gali dominti skaičiai, ir dėmesys kreipiamas į kitus skaičius sakinyje, tačiau gali būti ir kitų atvejų, pvz., dėmesys atkreipiamas į objektą, kuris yra vienas.

Taigi, kadangi tam pačiam žodžiui gali būti atkreipiamas kitoks dėmesys, transformerių architektūra įgyvendina keletą skirtingų dėmesio sluoksnių. Daugialypis dėmesys (angl. *multi-headed attention*) – tai keli dėmesio sluoksniai, kurių išvestis yra sujungiama į vieną. Kiekvienas sluoksnis turi skirtingus svorius ir išmoksta atkreipti dėmesį į skirtingus dalykus.

Transformeriai dėmesio mechanizmą naudoja trijose vietose:

1. Komunikacijoje tarp enkoderio ir dekoderio – enkoderis užkoduoja raktų ir jų reikšmių matricą, dekoderis naudoja užklausą, kad išgautų enkoderio sugeneruotus žodžius ir jų kontekstą reprezentuojančius vektorius.

2. Enkoderio savidėmesio sluoksnis – čia enkoderis pats sukuria užklausas, raktus ir reikšmės iš tos pačios įvesties.

3. Dekoderio savidėmesio sluoksnis – sukuriamas panašiai, kaip ir enkoderio savidėmesio sluoksnis, su vieninteliu skirtumu – dekoderiui uždrausta kreipti dėmesį į ateities žodžius, kadangi kitaip nėra išmokstama generuoti nieko naujo, verčiau – visi žodžiai yra atkartojami. Toks žodžių (jų vektorinių reprezentacijų) slėpimas matricoje vadinamas užmaskavimu (angl. *masking*).

2019 m. buvo išanalizuoti daugialypio dėmesio mechanizmo skirtingų „galvų“ (angl. *heads*) rolės [VTM+19]. Parodyta, jog nemažą dalį didžiausią įtaką turinčių daugialypio dėmesio komponentų galima išskirti į tris atliekamas funkcijas:

1. Pozicinė – dėmesys skiriamas į artimiausią (gretimą) žodį,

2. Sintaksinė – dėmesys skiriamas į sintaksiškai susijųsį žodį,

3. Retų žodžių – dėmesys skiriamas į rečiausius žodžius.

Taip pat buvo parodyta, jog dauguma dėmesio komponentų galima pašalinti – ypač enkoderio – beveik be tikslumo rezultatuose praradimo. Autoriai parodė, jog kuo labiau dėmesio komponentas yra specializuotas, tuo jis svarbesnis.

Tiesioginio sklidimo sluoksnis. Po dėmesio sluoksnio informacija keliauja į tolimesnius tiesioginio sklidimo (angl. *feed-forward*) sluoksnius. Šis sluoksniai skirti vykdyti tiesines transformacijas:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2. \tag{11}$$

Nepaisant to, jog šie sluoksniai turi ganėtinai daug mokymo parametrų, literatūroje apie juos kalbama mažai. 2020 m. buvo parodyta, jog šis sluoksnis yra taip pat labai svarbus informacijos išgavime, kadangi jis reprezentuoja rakto-reikšmės *atmintį* [GSB+20]. Raktai atitinka tam tikras sintaksės bei semantines struktūras tekste, o reikšmė – išvesties žodžių skirstinį.

Transformeriai yra tinkamas įrankis spręsti galybei natūralios kalbos apdorojimo užduočių. Straipsnyje [VSP+17] rašoma, jog transformerių tinklai buvo išbandyti kalbų vertimo užduotyse, bei rezultatai (vertinami BLEU taškais) pranoko visus kitus iki tol esančių modelių rezultatus.

1.3.1. Iš anksto apmokyti modeliai

1.3.1.1. Abipusio enkoderio transformerių reprezentacijos modelis BERT

BERT (angl. *Bidirectional Encoder Representation from Transformers*[2]), lietuviškai – abipusio enkoderio transformerių reprezentacijos modelis – buvo pristatytas 2018 m. kaip alternatyva GPT [DCL+18]. Pristatytas modelis labai panašus į GPT: BERT taip pat yra paremtas transformerių tinklais, tačiau, skirtingai nei GPT, BERT susideda iš enkoderių. BERT tikslas, kaip ir GPT, sukurti generalizuotą natūralios kalbos užduotis sprendžiantį modelį, tačiau BERT architektūra išsprendžia abipusių tekstinių reprezentacijų problemą.

GPT naudojasi dekoderio idėja užmaskuoti būsimus žodžius, kad žodžių spėjimas būtų paremtas tik buvusiais žodžiais. Tačiau kaip žinia žodžio reikšmei daryti įtaką gali ir vėlesni žodžiai sakinyje. BERT modelyje to išvengiama pasitelkus du mechanizmus:

1. Atsitiktinis žodžių maskavimas. Užmaskavus tam tikrus žodžius, modelis yra išmokomas atspėti, kokie yra tie žodžiai;

2. Sakinių eiliškumo spėjimas. Modelis išmokomas atspėti ar tam tikras sakinys yra prieš ar po kito.

Modelis apmokomas BooksCorpus korpusu bei portalo Wikipedia straipsniais anglų kalba. Mokymo tikslas – atspėti sekantį žodį.

Apmokytas modelis vėliau pritaikomas tam tikrai specifinei užduočiai pridedant papildomą išvesties sluoksnį ir jį kartu su kitais BERT parametrais išmokant dar kartą siekiant gauti norimus atsakymus.

[2]Modelio kodas pasiekiamas https://github.com/google-research/bert

BERT rezultatai tam tikrose užduotyse, pavyzdžiui, GLUE, pranoko daugumą kitų modelių, net ir GPT. Taip pat parodyta jog BERT be gebėjimo matyti žodžius į priekį (taip pat kaip ir GPT) užduotis atlieka prasčiau, nei pradinis BERT modelis. Negana to, autoriai parodė, jog didinant modelį (parametrų skaičių), gaunami geresni rezultatai ne tik didelės skalės užduotyse, bet ir mažesnės (t.y. tuomet, kai duomenų kiekis apmokant modelį antrą kartą spręsti specializuotas užduotis yra mažas).

1.3.1.2. Tekstą generuojantys modeliai

Ilgą laiką įvairių natūralios kalbos apdorojimo problemų sprendimą ribojo tai, jog kiekvienam modeliui apmokyti reikėjo galybės duomenų. 2018 m. buvo pasiūlytas pirmasis generalizuotas iš anksto apmokytas generatyvus iš anksto apmokytas modelis GPT (angl. *Generative Pre-trained Transformer*), kurį galima pritaikyti įvairiems uždaviniams [RNS+18].

GPT modelis paremtas transformerių architektūra, tačiau be enkoderio dalies. Modelis apmokomas milžinišku kiekiu duomenų – įvairiais tekstais – siekiant išmokti atspėti sekantį žodį naudojantis stochastiniu gradientų nusileidimu.

Atlikus šį pirmąjį mokymo etapą modelis apmokomas dar kartą, šį kart su sužymėtais (angl. *labeled*) duomenimis, siekiant iš sekos išgauti tam tikrą žodį. Tam pridedamas papildomas sluoksnis, gebantis paversti GPT modelio sugeneruotą žodį į tą, kuris buvo pateiktas kaip žymeklis.

Kadangi modelis išmokytas sugeneruoti sekantį žodį tam tikrai žodžių sekai, norint toliau išmokyti spręsti tam tikrus uždavinius, pvz., gebėti atsakyti į klausimus, įvestis turi būti šiek tiek pakeista, kad tiktų modeliui. Tais atvejais, kai įvesties duomenys turi tam tikrą struktūrą, įvestis yra paverčiama į paprastą žodžių seką (pvz., klausimus sujungiant su atsakymu į vieną bloką). Taip išvengiama modifikacijų pačiam modeliui.

Modelis, išmokytas su tekstu iš virš 7000 įvairaus žanro knygų anglų kalba BooksCorpus rinkiniu, vėliau sėkmingai pritaikytas tokiom užduotimis kaip išvadų darymas natūralioje kalboje (turint du sakinius, bandoma atspėti sąryšį tarp jų – pasekmė, prieštara ar neutralumas), atsakymas į klausimus, semantinis panašumas ir kt.

1.3.2. Kiti iš anksto apmokytų generatyvių transformerių modeliai

2019 m. ši generalizavimo idėja buvo išplėtota dar toliau. Vietoje to, kad modelio parametrai būtų atskirai pritaikyti kiekvienai užduočiai, pati užduotis gali būti paduodama kaip įvestis [RWC+19]. išmokius modelį vieną kartą daugiau parametrų nereikia keisti – užtenka paduodame duomenų tekste pasakyti, ko norima, pvz., „išversk <*tekstas*> iš <*kalba1*> į <*kalba2*>“.

Šiam sumanymui įgyvendinti reikia milžiniško kiekio įvairių duomenų, kuriuose glūdėtų pakankamai informacijos apie pačius natūralios kalbos uždavinius, kad modelis galėtų juos atlikti nekeičiant parametrų. Pavyzdžiui, jei tekste, pagal kurį apmokomas modelis, slypi tokie sakiniai kaip „einu į mokyklą, arba, išvertus į anglų kalbą – I’m going to school“, paprašius išversti sakinį „einu į mokyklą“ į anglų kalbą modelis turėtų nesunkiai su tuo susidoroti. Tam buvo sukurti interneto naršytojai (angl. *web scrapers*), renkantys įvairiausią tekstą iš Reddit socialinės platformos.

Gautas GPT-2 modelis, kelis kartus didesnis už GPT savo parametrų skaičiumi, sėkmingai buvo panaudotas sprendžiant tokias užduotis kaip išmesto žodžio iš sakinio atspėjimas, teksto suvokimas, vertimas, santraukų sudarymas ir pan. Įdomu tai, jog modelis sugebėjo atsakyti į klausimus, kurių nebuvo tekste, pagal kurį modelis išmokytas.

Vėliau buvo pristatytas GPT-3 modelis, turintis dar daugiau parametrų ir apmokytas su dar didesniu kiekiu duomenų [BMR+20]. Modelis buvo toks efektyvus generuojant tekstą, jog susilaukė itin daug viešo susidomėjimo bei nuogąstavimų, jog dirbtinio intelekto sukurtas tekstas darosi neatskiriamas nuo žmogaus kurto teksto [She20]. Dėl šios priežasties apmokytas modelis nebuvo išleistas viešai, tačiau išleistas API, kurį gali naudoti įvairios verslo aplikacijos [3].

1.4. Dialogo būsenos sekimas transformerių tinklais

Šiame poskyryje aprašomi keli scenarijai, kai transformeriai bei jais paremti tinklai yra naudojami spręsti dialogo būsenos sekimo uždavinį. Pagrinde aptariami du metodai: pirmasis yra tinkamas tada, kai visos reikšmės ontologijoje yra žinomos. Tuomet generuojami kandidatai, bei vykdomas klasifikavimo uždavinys, kur klasė nurodo, ar kandidatas atitinka būseną, ar ne; antrasis metodas tinkamas naudoti, kai ontologija yra dalinai žinoma, ir kandidatų generuoti neįmanoma. Tuomet būsenos reikšmės ieškoma tekste vykdant klasifikavimo uždavinį, šį kartą su dvejom klasėm – pirmoji klasė parodo tikimybę, kad žodis yra būsenos reikšmės pradžia, antroji – tikimybę, kad žodis yra būsenos reikšmės pabaiga.

1.4.1. Dialogo būsenos sekimas generuojant kandidatus iš ontologijos

Dažnai sprendžiant DST uždavinį pateikta fiksuota ontologija. Tokiais atvejais dialogo sistema yra ganėtinai ribota, tačiau pasižyminti didelio tikslumo spėjimais. Tai gali būti svarbu siekiant išvengti klaidų tam tikrais atvejais – pavyzdžiui, rezervuojant skrydžius. Tuomet pravartu turėti ontologiją, kuri nusako, kokius žodžius yra įmanoma išgauti iš pokalbio. Populiariuose duomenų rinkiniuose skirtuose su dialogu susijusių uždavinių sprendimams tikrinti, pvz., [HTW14b] [BWT+18], visiems dialogams yra pateikiamos ontologijos.

[LTB+20] sprendžiamas toks uždavinys, kai ontologija (t.y. visos galimos būsenos reikšmės) yra žinoma iš anksto. Uždavinys sprendžiamas pasinaudojant BERT modeliu – pasirinkimas motyvuotas tuo, jog BERT modelis padeda uždavinį žymiai supaprastinti. Pagrindinis keliamas tikslas modeliui buvo paprastumas, t.y. kuo mažesnis parametrų skaičius, kuris nesikeičia didėjant ontologijai.

Autorių teigimu, yra du pagrindiniai modelio pranašumai: pirmasis – dydis, arba paprastumas. Anot autorių, modelis užima 8 kartus mažiau vietos nei pilnas BERT modelis bei veikia 7 kartus greičiau, paaukojant itin mažai tikslumo. Tai pasiekta naudojantis žinių distiliavimo metodu, kai mažesnis modelis yra apmokamas atkartojant didesnio modelio išvestį tai pačiai įeičiai. Antrasis modelio pranašumas – gebėjimas spręsti DST problemą net tada, kai ontologija keičiasi dialogo raidoje, t.y. yra dinamiška.

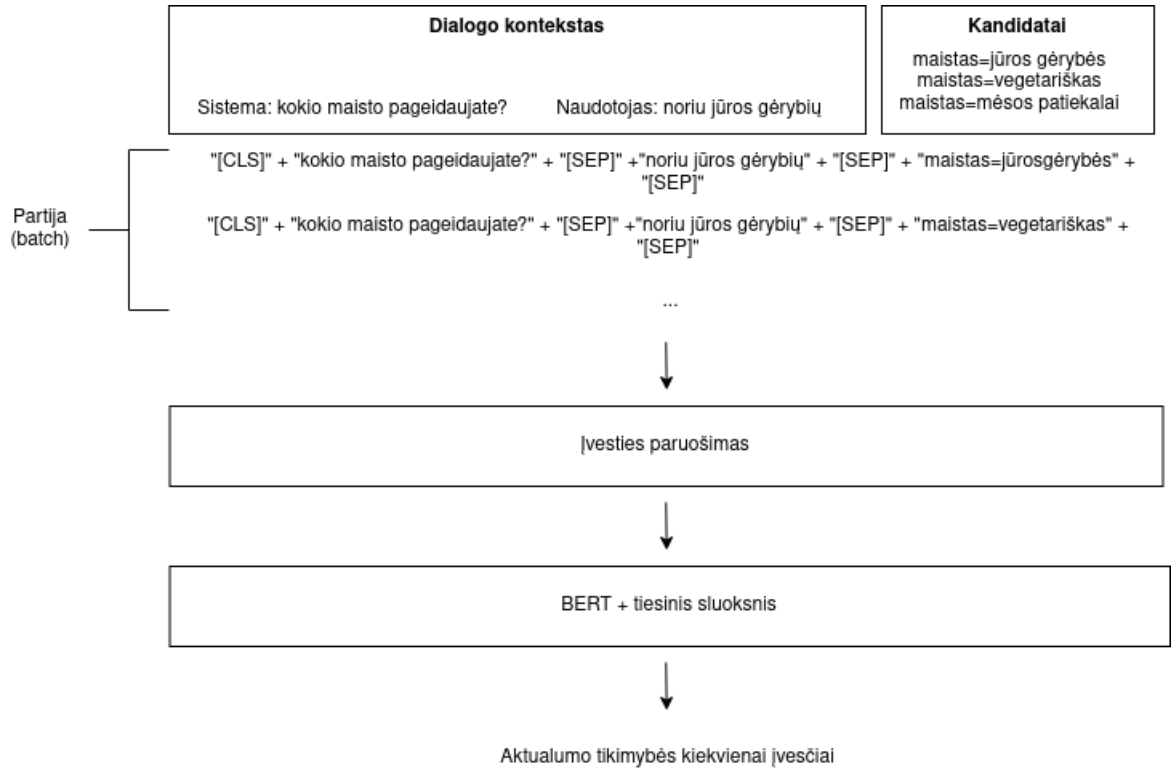
[3] https://openai.com/blog/gpt-3-apps/

Sprendimas yra panašus į sakinių porų klasifikavimo problemą (įeitis – sakinių pora, tikslas – tam tikra klasė, pvz., sakinių panašumas). Turint dialogo kontekstą C bei kandidato kintamojo ir jo reikšmės porą $(s_n, v_m) \in O$, modelis pateikia atsakymą, ar s_n ir v_m yra dialogo kontekste C.

Dialogo kontekstas C yra sukonstruojamas paėmus sistemos pasisakymą praeitame žingsnyje (kuris gali būti lygus tuščiam žodžiui „“, jei $t = 0$) bei naudotojo pasisakymą dabartiniame žingsnyje. Sukonstruojama įeitis I_i sujungus C, s_n ir v_m į vieną sakinį: $I_i = concat(C, s_n, v_m)$. Šis sakinys yra papildomai apdorojamas prieš pateikiant jį modeliui. Verta paminėti, jog net ir pasikeitus ontologijai, modelio nereikės apmokyti iš naujo – modelis gali būti tiesiogiai panaudotas naujiems kintamiesiems bei jų reikšmėms, kurių nebuvo apmokymo aibėje. Modelis pateikia tikimybę, jog tam tikras kandidatas kintamasis ir jo reikšmė yra paduotame dialogo kontekste.

BERT modelis naudojamas duomenis paversti į įterpinius (angl. *embeddings*). Prieš įeitį pateikiant BERT enkoderiui, ji yra paverčiama į vektorių atliekant kelis žingsnius: 1) įeitis suskaidoma į žetonų eilę (angl. *tokens*) 2) žetonų eilė paverčiama į skaitinį vektorių. Enkoderio išvestis – vektorius $(h1, h2, ..., h_m)_i$ kiekvienai įeičiai I_i, kur m yra maksimalus įeities I_i dydis (visų įeičių vektorių dydžiai yra suvienodinami pasinaudojus apkemšimo (angl. *padding*) principu, t.y. į mažesnius vektorius pridedamos 0-inės reikšmės tol, kol jų dydžiai sutampa su ilgiausio vektoriaus). Šių vektorių pirmoji reikšmė h_1 yra praleidžiama per tiesinį sluoksnį $y = \sigma(Wh_1 + b)$, W ir b yra modelio parametrai.

Rezultatas – y reikšmė kiekvienam įeities elementui I_i, nurodanti tikimybę, jog kandidatas kintamasis ir jo reikšmė yra duotajame dialogo kontekste (kuriame glūdi naudotojo pasisakymas). Jei ši reikšmė didesnė, nei 0,5, būsena yra atnaujinama naujomis reikšmėmis.



8 pav. DST sprendimo BERT modeliu iliustracija

Modelis autorių buvo patikrintas su Woz 2.0 duomenų aibe. Autorių teigimu, modelis savo tikslumu pranoko daugelį kitų žinomų modelių, įskaitant ir modelius paremtus BERT. Turbūt nenuostabu, jog jungtinio tikslo (angl. *joint goal*) rezultatai pranoko kitą šiame darbe nagrinėjamą sprendimą [CL19], kuriame galimos kintamųjų reikšmės nėra apibrėžtos ontologijoje, tačiau daroma prielaida, jog egzistuoja tekste.

2 lentelė. Netinkamas kintamųjų atpažinimas, galintis įvykti, kai naudojamos iš anksto apibrėžtos ontologijos reikšmės

Naudotojo pasisakymas	Kintamųjų ir jų reikšmių poros
hello, I'm looking for a restaurant with fair prices in the area **north**	(area, north), (price range, moderate)
hello, I'm looking for a restaurant with fair prices in the **oldtown Vilnius** area	(price range, moderate)

1.4.2. Dialogo būsenos sekimas, kai būsenos reikšmių ieškoma tekste

Darbe [CL19] taip pat yra nagrinėjama, kaip panaudoti BERT sprendžiant DST uždavinį. Tačiau, skirtingai nei [LTB+20], kintamųjų reikšmės gali būti iš anksto nežinomos – verčiau, daroma prielaida, jog visos reikšmės yra tekste. Tokį pasirinkimą galima pagrįsti tuo, jog dauguma pasisakymų dialoguose, kai naudotojas prašo sistemos atlikti tam tikras užduotis, išreikštinai mini kintamojo reikšmę. Pavyzdžiui, naudotojui pasakius „noriu užsirezervuoti vietą rytoj 11 val.“, akivaizdu, jog laiko reikšmė yra „rytoj 11 val.“.

Kai kintamųjų reikšmių aibės dydis yra baigtinis, vienas iš paprasčiausių sprendimų yra generuoti kandidatus bei tikrinti kiekvieno iš jų egzistavimą tekste. Tačiau jeigu kintamųjų reikšmių aibės dydis yra praktiškai nesuskaičiuojamas (pvz., laikas, įskaitant metus, mėnesius, dienas, valandas, minutes, sekundes, milisekundes ir t.t.), adresai ir kt., sudaryti kandidatus tampa itin nepraktiška. Tokiu atveju būtina naudotis kita turima informacija idant rasti tinkamą reikšmę.

Šio sprendimo idėja remiasi tokiu principu: BERT modeliui pateikiama dialogo informacija (kaip ir [LTB+20], ji yra sistemos pasisakymas praeitame žingsnyje, bei naudotojo pasisakymas dabartiniame žingsnyje), gaunamas vektorius, reprezentuojantis kontekstinę informaciją. Šis vektorius (susidedantis iš sakinių lygio ir sakinio dalių lygio informacijos) toliau yra perduodamas į tiesinius sluoksnius, skirtus klasifikuoti, ar tekste slypi informacija apie dialogo kintamųjų reikšmes. Kiekvienas iš šių tiesinių sluoksnių yra apmokytas skirtingam dialogo kintamajam.

Architektūroje po BERT naudojami du skirtingo tiesiniai sluoksniai: pirmasis apdoroja visą sakinio kontekstinę informaciją, užkoduotą BERT išvesties vektoriuje t_0, ir klasifikuoja, ar tekste yra reikalinga informacija. Jei ji yra, išvestis, užkoduojanti sakinio dalių (žetonų, angl. *tokens*) lygio informaciją (vektoriuose $t_1, t_2, ..., t_n$, kur n – sakinio dalių (žetonų) skaičius įeityje) perduodama į tolimesnius tiesinius sluoksnius, kurie nustato, kurie įeities elementai nusako dialogo kintamąjį ir jo reikšmę – tiksliau jų vietą įeityje (pradžios indeksą ir pabaigos). Kaip ir minėta anksčiau, visi parametrai šiems sluoksniams yra parenkami kiekvienam dialogo kintamajui iš ontologijos atskirai.

Šis modelis turi nemažai panašumų su [LTB+20]: abiejuose sprendimuose naudojami BERT enkoderiai ir tiesiniai sluoksniai po jo. Abi architektūros būseną atnaujina tokiu pat būdu (perrašant senas reikšmes). Taip pat panašiai konstruojamas dialogo kontekstas iš sistemos ir naudotojo pasisakymų, tik kuriant įeitį modeliui, BERT-DST nėra naudojami kandidatai.

1.4.3. Kiti dialogo būsenos sekimo metodai

1.4.3.1. Architektūra „iš sekos į seką"

Itin lankstus metodas yra enkoderiui pateikti ne tik dialogo kontekstą, bet kartu ir visą ontologiją – šis sprendimas yra pasiūlytas darbe [FWL20], kurių sprendimas įgyvendintas sekos-į-sekos (angl. *seq-to-seq*) architektūra. Dialogo istorija bei ontologija paverčiami į šabloninius vektorius, kuriems apjungti yra ištreniruojamas dėmesio sluoksnis (angl. *attender*). Sukuriamos pasisakymų-pagal-schemos-dėmesį reprezentacijos (angl. *schema-attended utterance representations*) E_a ir schemos-pagal-pasisakymo-dėmesį reprezentacijos (angl. *utterance-attended schema representations*) D_a. Abu kintamieji yra vektoriai. Tuomet šie vektoriai naudojami išgauti būseną naudojantis LSTM dekoderiu kartu su rodyklės (angl. *pointer*) [VFJ15] ir dėmesio [BCB14b] mechanizmais.

Nepaisant to, jog [FWL20] pasiūlyta architektūra dialogo būsenai atpažinti naudojasi ir dialogo tekstu, ir ontologija, skirtingai nei anksčiau minėti darbai, įgyvendinta sistema yra kompleksiška: naudojami enkoderiai – du BERT modeliai kurti dialogo ir ontologijos šablonėms reprezentacijoms, dekoderiai – LSTM kartu su rodyklės ir dėmesio mechanizmais, bei juos apjungiantis dėmesio tinklas. To pasekoje sistema tampa kaip juoda dėžė – darosi sunku suprasti, kokios transformacijos vyksta jos viduje, aptikti sistemos silpnąsias bei stipriąsias vietas ir rasti būdų ją patobulinti.

Visgi šis sprendimas lenkia aukščiau aprašytus sprendimus. Pavyzdžiui, naudojantis WOZ 2.0 duomenų rinkiniu, pasiektas 0,912 jungtinio siekio tikslumas, kai tuo tarpu BERT-DST pasiektas 0,877, o tikrinant DSTC2 duomenų rinkiniu, pasiektas 0,850 jungtinio siekio tikslumas – žymiai didesnis už BERT-DST pasiektą 0,693. Autoriai šiuos puikius rezultatus priskiria generalizuotam architektūros dizainui, gebančiam susidoroti su daug skirtingų atvejų.

1.4.3.2. GPT architektūros

Nepaisant to, jog GPT modeliai yra dažniausiai naudojami teksto generavimui, jie kartais yra naudojami ir dialogo būsenos sekimo užduočiai spręsti, pvz., [YLQ21] ir [LTB21]. GPT modelis yra apmokomas generuoti dialogo būseną iš kontekstinės informacijos, t.y. dialogo istorijos ir egzistuojančių būsenų.

GPT metodai nėra itin plačiai išnagrinėti literatūroje. Tačiau architektūrų, naudojančių GPT rezultatų tikslumas sprendžiant DST neatsilieka nuo BERT architektūrų. Pavyzdžiui, [YLQ21] tikslumo metrikos testuojant su MultiWOZ 2.1 duomenų rinkiniu siekia 56,20 % jungtinio tikslumo (*joint accuracy*), o [LTB21] su MultiWOZ 2.0 duomenų rinkiniu pasiekia 54.86 % jungtinio tikslumą. Šiame darbe vienintelė nagrinėta BERT naudojanti architektūra [FWL20], kuri tikrinta su tais pačiais duomenų rinkiniais (MultiWOZ 2.1), pasiekia 56,1 % Tačiau enkoderiais paremtos architektūros išlieka tarp lyderių, pvz., ROBERTA modeliu paremta architektūra [YZP+20] pasiekė 60,48 % jungtinį tikslumą su MultiWOZ 2.1 duomenų rinkiniu.

1.4.3.3. ChatGPT

Pastaruoju metu ypač daug susidomėjimo susilaukė ChatGPT [Ope22], GPT architektūra paremtas modelis, gebantis generuoti itin realistiškus atsakymus į naudotojo užklausas. ChatGPT

atsakymai yra tiek detalūs, jog panašu, kad jis gali pakeisti kitus specializuotus modelius, skirtus tam tikriems uždaviniams spręsti. ChatGPT gali atsakyti į klausimus, generuoti istorijas, apibendrinti tekstą, atpažinti sentimentus, rašyti kodą dauguma programavimo kalbų ir kt.

Kyla natūralus klausimas – ar ChatGPT taip pat gali išspręsti dialogo būsenos sekimo problemą? Panašu, jog literatūros, atsakančios į šį klausimą, kol kas nėra. Atlikti eksperimentus su ChatGPT kurį laiką buvo sudėtinga, kadangi modeliai yra komerciniai, ir vienintelis būdas naudoti ChatGPT tai per naudotojo sąsają. Tačiau OpenAI išleidus API, eksperimentai supaprastėjo. Apie juos daugiau 3.3.6 poskyryje.

2. Siūlomos architektūros

Šiame skyriuje glaustai išdėstytos siūlomos naujovės, siekiant pagerinti rezultatus dialogo būsenai sekti bei supaprastinti tam tikras architektūras.

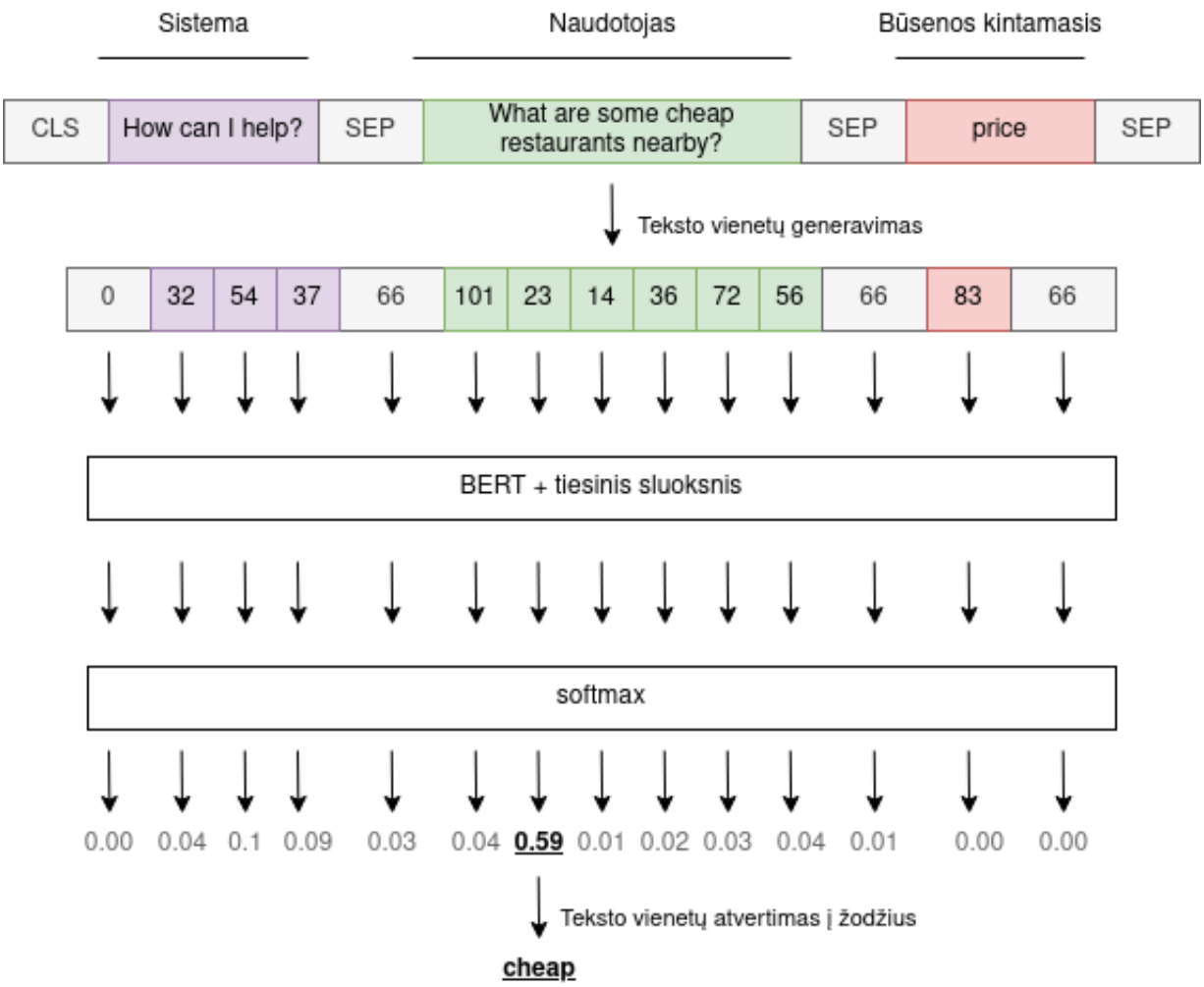
2.1. Paprastesnė būsenos radimo tekste architektūra

Šio darbo metu apibrėžta ir įgyvendinta architektūra, supaprastinanti sprendimą, pateiktą [CL19] bei aprašytą poskyryje 1.4.2. Šis sprendimas leidžia rasti dialogo būsenos reikšmę tekste. Tačiau kad tą pasiekti, yra apmokama s skirtingų klasifikavimo sluoksnių, kur $s \in \mathcal{S}$ nurodo galimus kintamuosius, kurie yra apibūdinti ontologijoje.

Skirtingi klasifikavimo sluoksniai reiškia, jog sprendimas nėra lankstus: atsiradus papildomoms reikšmėms ontologijoje reikia keisti visą architektūrą ir iš naujo permokyti visus modelius. Taip pat apmokymo procedūra tampa ne triviali – net ir naudojantis žinių pernešimu, n klasifikavimo sluoksnių reiškia, jog modelį teks apmokyti n kartų (kiekvienam sluoksniui po kartą).

Kadangi sprendimas daro prielaidą, kad ontologijos kintamieji yra žinomi (reikšmės – nebūtinai), autorius siūlo šią informaciją integruoti į įeities tekstą. Tai panašus būdas į kandidatų generavimą 1.4.1, su skirtumu, jog nėra generuojamos kandidatų reikšmės. Taigi, vietoje poros s, v į dialogo kontekstą įeina tik s.

Šis metodas teikia tuos pačius privalumus, kaip ir 1.4.2 poskyryje aprašytas metodas, tačiau tuo pačiu leidžia vienu ypu apmokyti visą modelį kartu su klasifikavimo sluoksniu neribotam skaičiui ontologijos kintamųjų s.



9 pav. Siūloma nauja architektūra DST uždaviniui, ieškant būsenos tekste, spręsti

2.1.1. Teksto segmentų užmaskavimas

Kaip matyti 9 paveikslėlyje, nors įeities tekstas turi keletą segmentų, atsakymas į klausimą „kokia yra dialogo būsena“ visada slepiasi naudotojo pasisakyme. Tačiau netaikant jokių apribojimų apmokymo bei prognozių metu, modelis gali neteisingai suklasifikuoti žodžius iš kitų segmentų.

Akivaizdu, kad privertus modelį klasifikuoti tik žodžius iš segmento, kuriame slypi naudotojo pasisakymas, turi būti pasiekiamas geresnis bendras modelio tikslumas. Kad šią problemą išspręsti, siūlomi du būdai:

1. Apmokymo metu smarkiau bausti modelį už klaidas, kai spėjimas atliekamas už naudotojo pasisakymo ribų;

2. Prognozės metu ignoruoti visas prognozes, darytas už naudotojo pasisakymo ribų.

Pirmąją sąlygą tenkinti galima sukuriant svorių sąrašą, kuris yra taikomas skaičiuojant paklaidos reikšmę klasifikavimo metu. Tuomet nuostolių funkcijos reikšmė l tampa tokia:

$$l_n = -w_n \cdot log \frac{exp(x_{n,y_n})}{\sum_{c=1}^{C} exp(x_{n,c})} \tag{12}$$

Čia n reiškia n-tojo žodžio indeksą, x_n – modelio spėjimą (tikimybė, kad žodis atitinka būseną). Ši funkcija skaičiuoja reikšmes kiekvienam spėjimui, todėl c (kas paprastai reiškia klasę) žymi visų kitų žodžių spėjimus. log ir exp atitinkamai yra logaritminė bei eksponentinė funkcijos. Galų gale konfigūruojamas w parametras šią paklaidos reikšmę gali padidinti.

Antrąjį būdą pasiekti dar paprasčiau – kadangi spėjimas grąžina tikimybes, kad kuris nors žodis yra būsenoje, naudojamasi binariniu sąrašu (jo reikšmės 1 ir 0), dar vadinamu maske. Atlikus prognozę bei gavus tikimybių sąrašą, atliekama AND operacija, arba kitaip – sudauginami sąrašų i-tieji elementai ir sukuriamas naujas sąrašas, kuriame tikimybės už naudotojo pasisakymo ribų yra 0.

Tam, kad įgyvendinti abu šiuos reikalavimus reikalinga turėti sąrašą, žymintį skirtingus sakinio segmentus. Tai nėra sudėtinga padaryti, kadangi visuose duomenyse naudotojo ir sistemos pasisakymai yra atskirti. Tolimesniame skyriuje detaliau aprašyta, kaip šie duomenys yra paruošiami modeliui apmokyti. To tarpu poskyryje 3.3.3 aprašyti bandymai, parodantys, jog ši metodika pagerina modelio rezultatus.

2.1.2. Duomenų paruošimas

DSTC2 bei WOZ 2.0 duomenų rinkiniai neturi informacijos apie tai, kuris iš žodžių tekste atitinka būseną, tuo tarpu Sim-R ir Sim-M duomenų rinkiniai – turi, tačiau dėl nedidelės duomenų aibės, šių duomenų neužtenka tinkamai apmokyti modelį. Taigi, siekiant pasiekti kuo tikslesnius rezultatus, siūloma modifikuoti DSTC2 bei WOZ 2.0 duomenų rinkinius, pridedant papildomą informaciją, kur būsenos kintamojo reikšmė yra naudotojo pasakytame tekste (t.y. žodžio indekso pradžią ir pabaigą).

Kad tą pasiekti buvo naudota modifikuota apytikslės atikties (angl. *fuzzy match*) algoritmo versija. Algoritmas priema dialogo kontekstą (teksto formatu) bei ieškomą būsenos reikšmę. Dialogo kontekstas paverčiamas į visų galimų n-gramų sąrašą, kur $1 \leq n \leq words\_count(sentence)$, čia $words\_count$ žymi žodžių sakinyje skaičių. Pavyzdžiui, sakinys „dar ne vakaras“ bus transformuotas į sąrašą „dar“, „ne“, „vakaras“, „dar ne“, „ne vakaras“, „dar ne vakaras“. Tuomet kiekviena n-grama tikrinama su būsenos reikšme. N-gramoms priskiriama tikimybė, jog ji atitinka reikšmę; galų gale paimamas n-gramos su didžiausia tikimybe pradžios ir pabaigos indeksas. DSTC2 bei WOZ 2.0 duomenų rinkiniai yra papildomi šiais indeksais.

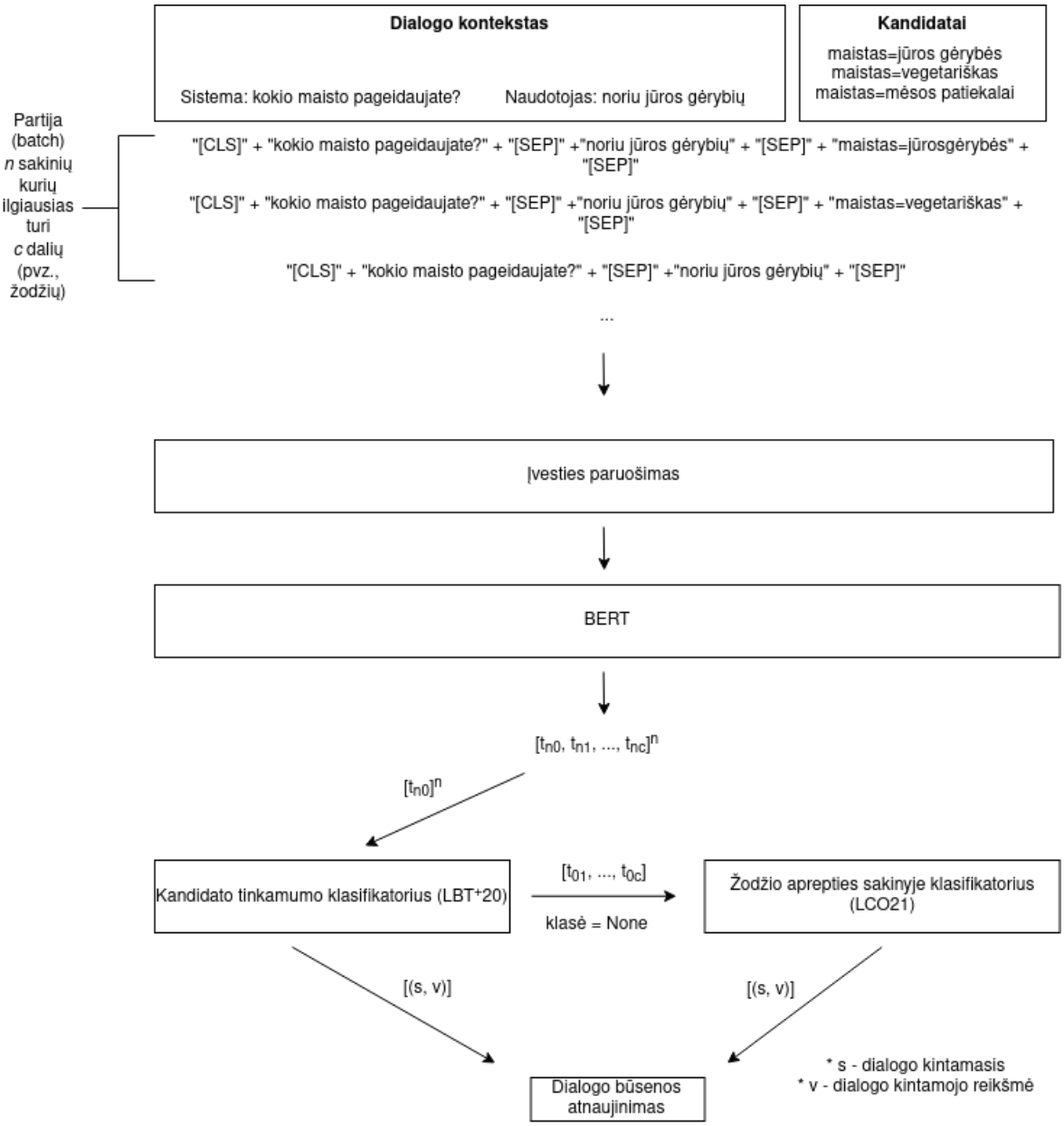
Negana to, kaip aprašyta 2.1.1 poskyryje, išsaugomas sąrašas (maskė), nusakantis, kurie žetonai įterpinyje priklauso naudotojui, kurie – ne.

Taigi, apmokymui BERT pateikiami žodžių įterpiniai bei žymuo 1, jei tas žodis atitinka būsenos reikšmę, bei 0, jei ne. Taikant funkciją CrossEntropyLoss naudojama aukščiau aprašyta maskė, siekiant atskirti, kurie žodžiai yra naudotojo, kurie ne. Toliau apmokymas vykdomas kaip ir kituose modeliuose.

2.2. Hibridinė architektūra

Atliekant bandymus su [LTB+20] bei [CL19], pastebėta, jog ten, kur vienas modelis suklumpa, tikslesnius rezultatus išgauna kitas. Remiantis tuo iškelta hipotezė, jog sukombinavus abu šiuos modelius į vieną architektūrą būtų gaunamas tikslesnes prognozes darantis modelis.

Ši architektūra būtų sudaryta iš dviejų individualiai apmokytų modelių, kurie aprašyti ankstesniuose skyreliuose. Pirmasis modelis, kaip ir [LTB+20], naudotųsi iš anksto žinomos ontologijos privalumais (generuojant kandidatus), t.y. dideliu tikslumu bei tikslių dialogo kintamųjų reikšmių atpažinimu, tačiau taip pat galėtų susidoroti su situacijomis, kai kandidatų generuoti nepavyksta, nes reikšmės nėra kategorinės. Jeigu tinkamo kandidato kintamojo ir reikšmės poros nepavyko rasti ontologijoje – jis ieškomas tekste, panašiu į saugumo nesėkmei (angl. *fail-safe*) principu. Bendrai, šis modelis turėtų susidoroti su daugiau atvejų, nei bet kuris iš prieš tai buvusių modelių.



10 pav. Siūlomo hibridinio modelio DST uždaviniui spręsti architektūra

Įeitis BERT modeliui paruošiama taip pat, kaip [LTB+20] – remiantis kandidatų generavimu, su vienu skirtumu: paruošiamas vienas įeities sakinys, kuriame nėra kandidatų. Tai reikalinga tam, jog kandidato tinkamumo klasifikatoriui neradus tinkamos reikšmės, būtų galima pasinaudoti žodžio aprepties sakinyje klasifikatoriumi. Jei įeities sakinyje bus kandidato kintamojo ir jo reikšmės pora, tai atsispindės ir BERT enkoderio sukurtame šablonų vektoriuje, tad būtent tie kandidatai ir bus randami klasifikatoriaus. Siekiant išvengti tokios situacijos, kartu su kitais paliekamsa ir originalus sakinys (sukurtas tik iš dialogo konteksto).

Su šiomis siūlomomis architektūromis atlikti bandymai, detaliau aprašyti 3.3.3 bei 3.3.4 poskyriuose.

Šio sprendimo savybėms keliami tokie tikslai:

1. Modelio tikslumas pagal jungtinio tikslo kriterijų naudojantis Woz 2.0 duomenų rinkiniu [WVM+16] turėtų būti nemažesnis nei [LTB+20], kadangi kandidatų reikšmių tikrinimas turėtų veikti taip pat. Skirtumai atsiranda tuomet, kai reikšmės nerandamos naudojantis klasifikatoriu, įgyvendintu pagal [LTB+20] – hibridinis modelis tuomet tuomet naudojasi [CL19] sprendimu, bandant rasti papildomus atvejus.

2. Architektūra turi likti kiek galima labiau paprasta, remiantis [LTB+20] darbe pateiktais apibrėžimais. Idealiu atveju, komponentų (t.y. skirtingų modelių) neturėtų būti daugiau, nei BERT ir keli papildomi tiesiniai sluoksniai, taip pat ir pradinis parametrų skaičius neturėtų būti didesnis nei aukščiau aprašytų darbų, bei neturėtų augti plečiant ontologiją. Norint tą pasiekti reikalingi patobulinimai žodžių aprepties sakinyje klasifikatoriuje, įgyvendintame pagal [CL19], kur visų tiesinių klasifikatorių parametrų skaičius auga, augant ontologijai.

3. Laiko trukmė, reikalinga modeliui apskaičiuoti rezultatus, neturėtų būti ženkliai didesnė už [LTB+20]. Minėtas darbas kenčia dėl to, kad augant ontologijai, reikalinga sugeneruoti ir patikrinti vis daugiau kandidatų. Viena iš galimų krypčių yra į ontologiją žiūrėti kaip į įeitį, iš kurios kuriamos šabloninės reprezentacijos, kaip daroma darbe [FWL20].

2.3. Siūlomi kriterijai bei metrikos skirtingiems modeliams palyginti

Kaip matyti iš tiriamosios dalies, kiekvienai pasiūlytai architektūrai keliami skirtingi tikslai – tai gali būti paprastumas, arba parametrų skaičiaus sumažinimas, tikslumo padidinimas ar pan. Tad tam, kad būtų susidaryti holistinį vaizdą apie tai, kokios yra kiekvieno būdo silpnybės bei stiprybės, siūloma sudaryti kriterijus, pagal kuriuos galima būtų lyginti modelius ir daryti išvadas.

Šiame darbe modeliai yra lyginami pagal šiuos kriterijus:

1. Bendra apmokymo trukmė,

2. Modelio dydis,

3. Jungtinio tikslo (angl. *joint goal*) tikslumas naudojantis (bet neapsiribojant) šiais duomenų rinkiniais: MultiWOZ 2.2, DSTC2, DSTC3, Sim-M, Sim-R ir pan.

Siekiant įsitikinti modelių efektyvumu produkcinėje aplinkoje, kur yra itin svarbus rezultatų apskaičiavimo kiekis ar resursų panaudojimas, siūloma modelius lyginti dar ir pagal šiuos kriterijus:

1. Bendra vykdymo trukmė,
2. Vykdymo trukmė su vienu įrašu,
3. Vykdymo ir apmokymo trukmės priklausomybė nuo įeities ir ontologijos dydžio (bendra ir vienam įrašui),
4. Naudojama vaizdo plokštės atmintis užkraunant modelį,
5. Naudojama vaizdo plokštės atmintis apmokant,
6. Naudojama vaizdo plokštės atmintis vykdant prognozes,
7. Architektūros paprastumas, kuris gali būti apibrėžtas 1) apmokymo parametrų skaičiumi, (svorių bei slenksčių skaičius, jei kalbama apie neuroninius tinklus, 2) naudojamų komponentų skaičius bei 3) vidinių sluoksnių kiekis.

Žinoma, šitie kriterijai nėra fiksuoti, tikimasi, jog tolimesniuose darbuose jie bus plėtojami, pvz., randant būdų įvertinti ne tik tikslumą, bet ir, pavyzdžiui, kokios yra dialogų, su kuriais modelis geba susitvarkyti lengvai, savybės.

3. Tiriamoji dalis

3.1. Įgyvendinimas

Šiame poskyryje pristatoma, kaip buvo įgyvendinta sistema, leidžianti atlikti bandymus. Kodas buvo rašomas siekiant įgalinti lengvus modelių, įvairių parametrų bei įvairių procesų (modelio apmokymo, rezultatų prognozavimo) veikimo pakeitimus įvairiems bandymams. Sukurtas kodas leidžia apdoroti ir paruošti įvairius duomenis skirtingoms DST užduotims, parinkti skirtingus modelius, keisti modelių parametrus (dydį ir panašiai), parinkti tokius parametrus kaip atsitiktinio praretinimo (angl. *dropout*) ar perdėto atsitiktinio perrinkimo (angl. *random oversampling*) stiprumą, įgalina vykdyti žinių pernešimą (angl. *knowledge transfer*); atlikti bandymai apmokant modelį ar kuriant prognozes yra išsaugojami kartu su parametrais bei papildoma informacija, pvz., kiek užtruko apmokyti; galiausiai sukurta programa leidžia vizualiai palyginti atliktų bandymų rezultatus.

3.1.1. Naudota įranga

Atlikti eksperimentus naudotasi Python programavimo kalbos 3.9.12 versija. Skaičiavimai atliekami su vaizdo plokštės pagalba naudojantis CUDA bibliotekos 11.7 versija. Operacinė sistema – Linux Mint 21 (Vanessa). Kompiuterio aparatūra: vaizdo plokštė – GeForce RTX 3070 8 GB, procesorius – AMD Ryzen 7 3700X, viso atminties – 32 GB. Karkasas, naudojamas apmokyti modelį bei vykdyti prognozes, buvo PyTorch [4]. Nupiešti grafikus naudotasi Python kalbos mathplotlib biblioteka. Iš anksto apmokytus modelius atsisiųsti naudota HuggingFace platforma.

3.1.2. Sistemos įgyvendinimas

Darbo metu buvo įgyvendinta platforma, leidžianti paruošti duomenis, modelius, juos apmokyti bei įvertinti jų tikslumą. Naudojantis įgyvendinta platforma buvo atliekami įvairūs eksperimentai, bandant palyginti kelis skirtingus dialogo būsenos radimo užduoties sprendimo būdus, bei rasti būdų, kaip juos pagerinti.

Duomenų paruošimas. DSTC2 bei WOZ 2.0 duomenys (kartu su ontologija) yra nuskaitomi JSON formatu. Kadangi sprendžiamos kelios skirtingos problemos, duomenys taip pat turi būti paruošti skirtingu būdu. Dialogo būsenos sekimo problemai spręsti, kai tikrinama, ar kandidatai atitinka būseną, duomenys yra paruošiami taip pat, kaip ir [LTB+20]: įvesties tekstas konvertuojamas į BERT įterpinius ir žymenį 1 ar 0. Tuo tarpu modeliams, kurie būsenos ieško tekste, duomenų ruošimas yra kiek sudėtingesnis. Jis aprašytas 2.1 poskyryje.

Modelio apmokymas. Iš pradžių užkraunami norimi apmokyti modeliai. Jei norima tęsti apmokyti jau apmokytus modelius, kurie yra išsaugoti atliekant bandymus, naudojama PyTorch *load()* bei *load_state_dict()* funkcijos. Kitu atveju iš anksto apmokyti modeliai BERT {https://huggingface.co/bert-base-uncased} bei RoBERTa [5] atsisiunčiami iš HuggingFace platformos. Sukuriamas papildomas tiesinis sluoksnis naudojantis PyTorch funkcija *nn.Linear*. Tiesinis sluoksnis bei

[4] https://pytorch.org/

[5] https://huggingface.co/roberta-base

BERT arba RoBERTa yra užkraunami į vaizdo plokštę PyTorch pagalba. Kaip elgtis kai modelis yra per didelis užkrauti į vaizdo plokštę aprašyta tolimesnėse pastraipose.

Paruošiami apmokymo ir validavimo duomenys, jie yra sumaišomi, taip pat pagal parametrų nustatymus vykdomas perdėtas perrinkimas, aprašytas 3.3.5.3 poskyryje, padidinant teigiamų pavyzdžių kiekį. Nustatomas epizodų skaičius bei duomenų rinkinio (*batch*) dydis. Vieno rinkinio (batch) duomenys yra transformuojami į BERT tinkantį vektorinį formatą bei apvelkami į *torch.Tensor()* klasę. Gaunami įterpiniai, vadinami *pooled_outputs*, kurie paduodami į tiesinį sluoksnį. Gautiems rezultatams įvykdoma PyTorch *CrossEntropyLoss()* funkcija, kuri savyje įvykdo dar ir *softmax* transformaciją bei apskaičiuoja paklaidą.

Gavus paklaidos rezultatus, kviečiama *backward()* funkcija, vykdanti sklidimo atgal metodą ir apskaičiuojanti gradientų reikšmes, tuo tarpu *optimizer.step()* funkcija įvykdo gradientinį nusileidimą ir atnaujina parametrų reikšmes. Parametrų reikšmių atnaujinimui naudojamas ADAM optimizatorius.

Apdorojus duomenų rinkinio dalį (batch), skaičiuojamas validacijos duomenų rinkinio paklaida (arba kitaip nuostolių reikšmės) labai panašiu būdu, tik nekviečiant *backward()* bei *optimizer.step()* funkcijų. Taip pat apskaičiuojamos F2 reikšmės, lyginant validacijos duomenų rinkinio rezultatus su tikraisiais rezultatais.

Apmokius išsaugomi paklaidos (nuostolių) funkcijos reikšmės, F2 reikšmės, modelio apmokymo trukmė, eksperimento parametrai JSON formatu bei patys modeliai (modeliai išsaugomi naudojantis PyTorch funkcija *save()* bei *state_dict()*) [6].

Modelių rezultatų lyginimas. Norint palyginti rezultatus, modeliai yra užkraunami tuo pačiu principu, kaip aprašyta anksčiau. Tuomet kuriamos modelių prognozės. Vienintelis skirtumas, gavus tikimybes, jos yra išverčiamos į būseną. Kandidatų generavimo metodo metu iteruojama per galimas būsenas bei ieškoma tos, kurių tikimybė didesnė nei nustatyta (pradine reikšmė $p = 0.5$), būsenos radimo tekste metu tas pats atliekama su žodžiais tekste.

Gautų būsenų aibė yra lyginama dvejopai: pirma – su tikrųjų būsenų aibėmis. Jei jos sutampa, priskiriamas 1 taškas, jei ne – 0. Tuomet išvedama vidutinė reikšmė, kuri nurodo, kokia dalis būsenos reikšmių buvo atspėta.

Didelio modelio užkrovimas. Kartais nutinka taip, jog modelis yra per didelis, kad jį būtų galima užkrauti į vaizdo plokštę. HuggingFace biblioteka pavadinimu Accelerate [7] leidžia naudojantis komanda *with init_empty_weights()* modelio dalis perkelinėti tarp disko, atminties bei vaizdo plokštės. Modelis yra išskirstomas į kelias dalis, vaizdo plokštėje paliekami tik tos dalys, kurios yra naudojamos apdorojimo metu, kai prireikia kitų dalių įvyksta perkėlimas iš atminties į vaizdo plokštę arba, jei trūksta atminties, iš disko į vaizdo plokštę.

Taip pat teikiamas dar vienas būdas kontroliuoti, kurie modelio sluoksniai laikomi atmintyje, vaizdo plokštėje ar diske naudojantis *model.hf_device_map* konfigūracija. Atliekant bandymus su modeliu, optimaliausia, jog visas modelis būtų laikomas vaizdo plokštėje, jei modelis laikomas diske, modelio prognozių apskaičiavimo laikas gali sulėtėti šimtus ar tūkstančius kartų.

[6] Daugiau informacijos apie modelių išsaugojimą https://pytorch.org/tutorials/beginner/saving_loading_models.html

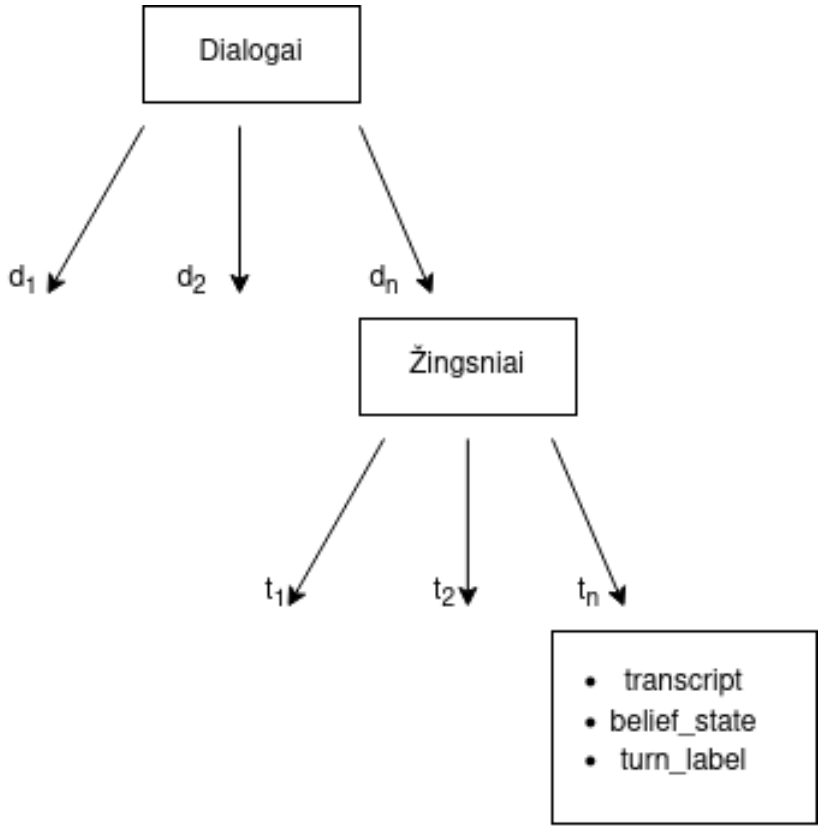
[7] https://huggingface.co/docs/accelerate/usage_guides/big_modeling

3.2. Duomenys

3.2.1. Duomenų rinkinys WOZ 2.0

Modeliui apmokyti naudojami WOZ 2.0 duomenys pagal [WVM+16] [8]. Šie duomenys yra surinkti naudojantis sutelktinių išteklių panaudojimo (angl. *crowdsourcing*) principu. Sunku pasakyti, kaip atsirado Wizard-of-Oz pavadinimas, tačiau originaliame straipsnyje duomenų rinkimo metodika pavadinama OZ [Kel84].

Duomenys yra JSON formatu. Jų turinys yra apie 1200 dialogų tarp sistemos ir naudotojo. Kiekvienas dialogas yra skirstomas į žingsnius (angl. *turns*), kiekvienas jų reprezentuoja naują sistemos ir naudotojo pasisakymą. Kiekvienam žingsniui yra pateikiama tikra dialogo būsena. Supaprastinta duomenų struktūra pateikiama 11 paveikslėlyje.



11 pav. Woz 2.0 struktūra

Kaip matome, duomenų aibė susideda iš daug dialogų, kiekvienas iš dialogų taip pat gali turėti keletą žingsnių. Žingsnyje laikoma informacija apie tai, ką pasakė naudotojas ir sistema bei tuo metu esanti būsena. Toliau pateikiamas vieno dialogo žingsnio (*turn*) pavyzdys JSON formatu. Svarbiausi JSON raktai – *transcript* (naudotojo pasisakymas), *system_transcript* (sistemos pasisakymas, šiame pavyzdyje – tuščias), *turn_label* (šiame žingsnyje išgauta), *belief_state* (visa dialogo būsena).

[8] Duomenys šiuo metu pasiekiami tik per atviro kodo programines įrangas, naudojančias šiuos duomenis, pvz., https://github.com/laituan245/BERT-Dialog-State-Tracking

```
{
        'turn_id': 0,
        'transcript': ['Are','there','any','eritrean','
          restaurants','in','town','?'],
        'turn_label': [['food', 'eritrean']],
        'belief_state': [{'slots': [['food', 'eritrean']], 'act':
           'inform'}],
        'system_acts': [],
        'system_transcript': '',
        'num': {'system_acts': [[0, 16, 1]],
        'transcript': [0, 106, 48, 120, 301, 141, 9, 14, 33, 1]}
}
```

1200 dialogų yra išskaidoma į žingsnius (vienas žingsnis – sistemos sakinys bei naudotojo užklausa), taip iš viso gaunant apie 5000 dialogo žingsnių. Toliau kiekvienas iš šių įrašų konvertuojamas į keletą naujų, kaip aprašyta poskyriuose 1.4.1 ir 1.4.2. Rezultate iš vieno įrašo gaunant n teigiamų pavyzdžių bei m neigiamų. Taip iš viso įrašų skaičius praplatinamas 100 kartų, rezultate turint duomenų aibę su 500 tūkstančių pavyzdžių tinkamų kandidatų radimui, ir apie 10 tūkstančių – būsenos tekste radimui.

Pagal gerąsias apmokinimo praktikas (žr. 1.2.3), dialogų aibė yra išskiriama į tris dalis: apmokymo, testavimo ir validacijos. Kadangi nėra griežtos taisyklės, kokie turėtų būti dydžių santykiai, šiame darbe duomenys išskiriami 8:1:1 santykiu, siekiant kuo daugiau duomenų palikti apmokymui.

Toliau duomenys yra transformuojami į formatą, tinkamą modelio apmokymui. Sistemos bei naudotojo sakiniai yra apjungiami (jie yra atskirti specialiu jungtuku) sukuriant dialogo kontekstą. Iteruojant per kintamuosius ir jų reikšmes iš ontologijos, visa ši informacija – kandidatas kintamasis, reikšmė, dialogo kontekstas yra apjungiama į vieną. Ši sąjunga yra paverčiama į unikalius teksto simbolius. Teksto simboliai paverčiami į skaitines reikšmes, taip pat išsaugomas maskavimo masyvas, atskiriantis skirtingas sakinių dalis (tačiau naudojant RoBERT modelį, to nereikia, žr. 3.3.5.5). Taip pat sukuriamas žymuo, nurodantis, ar kandidatas kintamasis ir jo reikšmė atitinka tikrąją žingsnio būseną: 1 jei taip, 0 jei ne.

Galutiniai duomenys, pateikiami BERT modeliui, atrodo taip, kaip parodyta 3 lentelėje.

3 lentelė. Galutiniai duomenys BERT modeliui po apdorojimo

Įeities vektorius	**Žymuo**
[0,1437,1437,2,38,524,546,...,1437,2]	1
[0,20,20088,2141,16,41,...,1437,2]	1
[0,38,1017,28,1372,7,...,1437,2]	0
...	...

Tikslus formatas (pvz., įvesties ilgis) gali skirti priklausomai nuo modelio, tačiau – ypač klasifikavimo užduočiai vykdyti – dauguma BERT bei apskirtai NLP architektūrų veikia panašiu prin-

cipu.

Šitaip, iš Woz 2.0 duomenų galima sugeneruoti virš 300 tūkst. įrašų modelio apmokymui. Verta paminėti, jog šitaip sugeneruotoje duomenų aibėje neigiamų įrašų yra 100 kartų daugiau, nei teigiamų, tad tam, kad tinkamai apmokyti modelį, verta pasitelkti įvairias technikas, sumažinančias skirtumą tarp teigiamų ir neigiamų pavyzdžių.

Taip pat duomenys turi ontologiją – arba schemą – apibūdinančią galimas reikšmes (taip pat iš jos generuojami kandidatai). Ontologijoje apibūdinti šie kintamieji: *area*, *food*, *price range*, *request*. Kiekvienas kintamasis turi leistinas reikšmes, pavyzdžiui, *food* gali būti *austrian*, *british*, *creative*, *seafood* ir panašiai. Toliau pateikiamoje 4 lentelėje apibūdinama ontologija.

4 lentelė. WOZ 2.0 ontologija

Būsenos kintamasis	Unikalių galimų reikšmių kiekis
area	7
food	76
price range	4
request	7

3.2.2. Duomenų rinkinys DSTC2

DSTC duomenys [9] sukurti dialogo būsenos sekimo iššūkiui (angl. *Dialog State Tracking Challenge*) [HTW14a; HTW4b; WRR+13]. Kaip ir WOZ 2.0, šie duomenys surinkti sutelktinių išteklių panaudojimo principu iš realių dialogų. Dialogų temos – informacijos apie restoranus, įskaitant kavines ir barus, bei informacija apie turizmą.

Duomenys savo struktūra yra labai panašūs į WOZ 2.0 duomenis, tad reikia labai mažai modifikavimo norint juos naudoti su tais pačiais modeliais, kurie yra apmokyti su WOZ 2.0 duomenų rinkiniu. Dialogų viso yra apie 3200, pavertus į dialogo žingsnius – 25500 įrašų, o kiekvieną jų konvertavus į pavyzdžius modeliui apmokyti ir įvertinti (aukščiau aprašytu būdu) gaunama apie 3 milijonus pavyzdžių (jie atrodo labai panašiai į tuos, pavaizduotus aukščiau esančioje 3 lentelėje). Toliau pateikiamas DSTC vieno dialogo žingsnio JSON formatu pavyzdys:

```
{
        'turn_id': 9,
        'transcript': 'could i get the phone number and price
          range',
        'turn_label': [['request', 'phone'], ['request', 'price
          range']],
        'system_acts': [],
        'system_transcript': 'cotto is in the moderate price
          range',
        'num': {}
}
```

[9] https://github.com/matthen/dstc

Struktūra ir pavadinimų reikšmės beveik atitinka WOZ 2.0 duomenų struktūrą bei reikšmes. Vienas iš skirtumų, kuriuos svarbu paminėti – DSTC2 duomenų rinkinyje nėra informacijos apie *belief_state*, todėl tikrinti tikslumą galima tik naudojantis *turn_label* (to žingsnio būseną), kas reiškia, jog nėra tikrinamas modelių gebėjimas būseną saugoti kelis žingsnius ir ją tinkamai atnaujinti. Kitas būdas vertinti modelį yra naudojantis WOZ 2.0 duomenų rinkiniu, kadangi duomenų rinkiniai yra gan panašūs struktūra ir turiniu. Dėl šios priežasties, taip pat ir tam, jog būtų galima lengvai apmokyti tuos pačius modelius skirtingais duomenų rinkiniais, šiame darbe DSTC2 duomenų rinkiniui taikoma WOZ 2.0 ontologija, aprašyta 4 lentelėje.

Taip pat svarbu paminėti, jog modelio tekste randama būsena nebūtinai sintaksiškai sutampa su duomenų aibėje apibrėžta tikrąja būsena. Kaip jau minėta, WOZ ir DSTC duomenų rinkiniuose, būsenos yra aprašytos ontologijoje. Tad jei ontologijoje apibūdintos reikšmės yra gan ribotos, o tuo tarpu pačiame tekste – ne, būtina naudoti įvairias technikas (pavyzdžiui, žodžių suvienodinimą), kad pasiekti geriausius rezultatus.

3.3. Atlikti bandymai ir analizė

Šiame poskyryje aprašomi atlikti pradiniai bandymai su [LTB+20] ir [CL19]. Bandymų esmė yra geriau suprasti modelius, išnagrinėti jų savybes bei bandyti atkartoti rezultatus, pateiktus literatūroje. Verta paminėti, jog beveik visi bandymai atlikti apmokius modelius su kiek sumažinta duomenų aibe dėl laiko stokos. Tačiau šiame darbe daroma prielaida, jog tokie patys rezultatai būtų gaunami ir praplečiant duomenų aibės dydį.

3.3.1. Bandymai su kandidatų generavimą naudojančia architektūra

Darbo metu buvo atlikti keli eksperimentai, kurių tikslas buvo geriau suprasti santykį tarp įeities bei ontologijos dydžio ir naudojamos vaizdo plokštės atminties bei vykdymo laiko. Šie rezultatai suteikia geresnį supratimą, kaip modelis reaguoja į įvairius kintamuosius.

Eksperimentai buvo atliekami šitaip: eksperimento metu vykdomi keli modelio funkcijos *.predict()* iškvietimai. Kiekvienam vykdymui yra apibrėžiama viena įeitis, susidedanti iš naudotojo pasisakymo. Prieš vykdant yra išsaugomas dabartinis sistemos laikas t_0, baigiant vykdyti – t_1, tuomet trukmė randama suskaičiavus $t_1 - t_0$. Taigi, šiame skyrelyje žodis „vykdymas“ atitinka vieną funkcijos *.predict()* iškvietimą. Eksperimentai vykdyti Jupyter aplinkoje [10].

Rezultatų, pateiktų literatūroje, atkartojimas. Pirmojo eksperimento metu buvo bandyta atkartoti autorių skelbiamą rezultatų tikslumą. Autorių teigimu, modelis turėtų pasiekti iki 90,4 % jungtinio siekio tikslumą naudojantis WOZ 2.0 duomenų rinkiniu. Duomenų rinkinys tikrinimui turi 1646 įrašų. Viso rinkinio tikrinimas užtruko 3 minutes ir 53 sekundes, gautas 84,7 % jungtinis siekio tikslumas.

Atminties naudojimas bei skaičiavimų trukmė. Modelio veikimas buvo patikrintas didinant įeities teksto ilgį. Rezultatai pateikiami 5 lentelėje. Reikia paminėti, jog pradinė išnaudojama

[10] https://jupyter.org/

vaizdo plokštės atmintis tik įjungus programą būdavo apie 570 MB, užkrovus modelį – apie 1270 MB.

5 lentelė. Modelio veikimas didinant įeities dydį

Įvesties dydis (B)	Užimama vaizdo plokštės atmintis (MB)	Vykdymo trukmė pirmą kartą (ms)	Vykdymo trukmė tolimesnius kartus (ms)
110	2591	670	100
143	2913	680	122
189	3607	718	190
275	5425	934	255
350	6615	1018	380
414	7827	947	459

Iš šių rezultatų matyti, jog įeities užkrovimas pirmą kartą užtrukdavo apie 500 milisekundžių (beveik nepriklausomai nuo įeities dydžio, tačiau tai reikėtų pagrįsti tolimesniais eksperimentais). Vykdymo laikas jau užkrovus į atmintį auga tiesiškai proporcingai įeities dydžiui. Panašu, jog naudojama atmintis auga apytiksliai tiesiškai proporcingai įeities dydžiui. Atliekant eksperimentus, autoriaus vaizdo plokštei pritrūko atminties didinti įeities dydį daugiau nei apie 420 baitų. Tai – galima spraga, kadangi 420 baitų yra visai nedaug teksto (priklausomai nuo koduotės). Eksperimentų metu, į 420 baitų buvo sutalpinama apie 34 žodžius (naudojantis Python programavimo kalba). Autoriaus nuomone tai, kad 34 žodžių sakinys reikalauja daugiau atminties, nei turi moderni žaidimams skirta vaizdo plokštė, parodo, jog kol kas tokius modelius integruoti į mažai resursų turinčias sistemas yra mažai vilčių.

Skaičiavimų trukmės priklausomybė nuo ontologijos dydžio. Kiekvienai reikšmei iš ontologijos konstruojamas kandidatas, iš kurio gaminama įeitis BERT. Siekiant įgyvendinti sprendimą, tinkantį apdoroti milžiniškus dialogus, kuriose kalbama įvairiomis temomis ir pan., kyla natūralus klausimas – ar šis sprendimas lieka tinkamu augant ontologijai? Rezultatai pateikiami kitoje lentelėje. Ontologijos dydis skaičiuojamas suskaičiavus visas poras iš aibės $O \subseteq \mathcal{S} \times \mathcal{V}$, kur aibė O yra sudaryta iš autorių naudojamo Woz 2.0 [WVM+16] duomenų rinkinio.

Įvesties dydis visais atvejais buvo to paties dydžio – 189 baitų. Iš lentelės 5 žinoma, jog vykdymo laikas su baziniu ontologijos dydžiu (105 reikšmių) yra apie 190 ms. Taip buvo lyginama užimama vaizdo plokštės atmintis.

6 lentelė. Modelio veikimas didinant ontologiją

Ontologijos dydis (reikšmių vnt.)	Užimama vaizdo plokštės atmintis (MB)	Vykdymo trukmė (neskaitant pirmojo karto) (ms)
105	3690	168
181	3863	326
225	4005	442
333	3986	594
485	4095	832
785	4118	1453

Iš rezultatų galime matyti, jog nors ir naudojama atmintis didėja nežymiai, tačiau vykdymo laikas didėja sparčiau, nei tiesiškai proporcingai: ontologijai padidėjus beveik 7 kartus, vykdymo trukmė pailgėjo daugiau nei 8 kartus. Šis rezultatas nekelia nuostabos, tačiau patvirtina, jog skirtingų kandidatų generavimas nėra itin praktiškas sprendimas.

Toliau pateikiami keli autoriaus pastebėjimai, apibendrinantys sprendimą:

1. Iš karto atnaujinti būseną, perrašant senus kintamuosius nėra pats geriausias variantas – taip gali būti prarandama naudinga informacija, o nauja gali būti klaidinga, pvz., ne taip interpretuotas pasisakymas ar naudotojas persigalvojo kelis kartus. Kyla klausimas, kaip tokiu atveju atgauti informaciją.

2. Jeigu ontologijoje yra s kintamųjų bei v reikšmių, bus sukurta $s \times v$ skirtingų kandidatų, tuo pačiu ir įeičių BERT modeliui. Pridėjus papildomus kintamuosius ir jų reikšmes į ontologijas, sistemos veikimas sulėtėja, kaip matyti iš atliktų eksperimentų. Idėjų pasisemti galima iš darbo [FWL20], kuriame ontologija yra paverčiama į šabloninius vektorius ir naudojama kartu su dėmesio sluoksniu. Autorių teigimu, jų sprendimas yra tinkamas plėtojimui (angl. *scalable*).

3. Bene didžiausia šio sprendimo problema yra ta, jog modelio gebėjimas atrasti kintamuosius ir jų reikšmes iš teksto yra labai ribotas. Jei kintamojo reikšmės nėra kategorinės, tuomet ir tinkamo kandidato įeities sudaryti nebus neįmanoma – tuomet gali būti neatpažinta reikalinga informacija (iliustruota 2 lentelėje).

4. Šis sprendimas buvo patikrintas su mažai duomenų (tik Woz 2.0 rinkiniu [WVM+16]). Kiti panašūs sprendimai, pvz., [CL19] yra patikrinti su Sim-M, Sim-R, DSTC2 duomenų rinkiniais. Taip pat šiuo metu rinkiniai ganėtinai pasenę, pavyzdžiui, egzistuoja MultiWOZ 3.0 versija, taip pat yra tolimesnių DSTC versijų.

5. Nėra minima, ar naudojamas enkoderis, paremtas BERT modeliu, yra specifiškai apmokytas spręsti DST problemas. Turint omenyje, jog gaunamas tikslumas lenkia daug kitų sprendimų, pvz., [CL19], atrodytų, jog ši gana svarbi detalė yra praleista. Jeigu iš tiesų modelis nėra papildomai apmokytas, išsiaiškinti, kada BERT ir jais paremtų enkoderių apmokymas specifiniams uždaviniams spręsti yra naudingas, o kada – nebūtinas, galėtų būti tolimesnių tyrimų kryptis.

6. Eksperimentai parodė, jog modelio veikimas reikalauja nemažai atminties. Net naudojantis 8 GB turinčia vaizdo plokšte, modelis sekant dialogo būseną veikdavo tik su iki 35 žodžių turinčiu tekstu.

3.3.2. Bandymai su būsenos radimą tekste atliekančia architektūra

Šio darbo metu buvo atlikti keli eksperimentai, siekiant nuodugniau išnagrinėti BERT bei būsenos radimo tekste veikimą. Modelis buvo apmokomas su Woz 2.0 bei DSTC 2 duomenų rinkiniais,

tikrinamas jo vykdymo laikas bei tikslumas: jungtinių tikslų tikslumas (angl. *joint goal accuracy*) bei vidutinis tikslų tikslumas (angl. *average goal accuracy*).

Deja, tačiau atlikti tokių pačių eksperimentų, kaip kad 2.1.1. skyrelyje, nepavyko dėl laiko stokos – šiai architektūrai įgyvendinti naudotas visiškai kitoks karkasas (Tensorflow, kai tuo tarpu anksčiau nagrinėtas darbas naudojo PyTorch), taip pat dėl kitų apmokymo ir modelio įvertinimo kodo skirtumų; tad modelių tarpusavio palyginimas nėra visiškai tikslus.

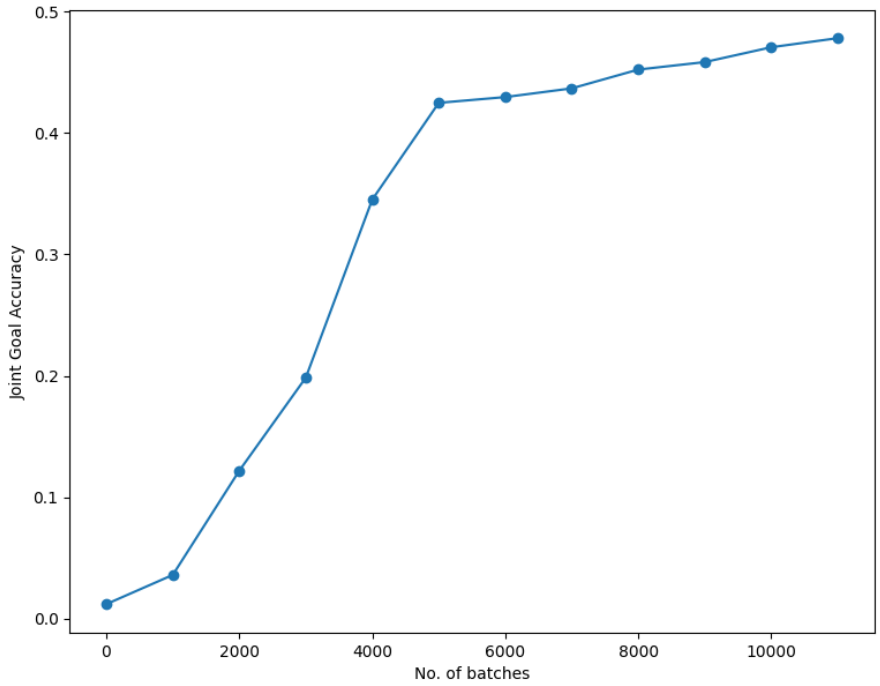
Rezultatų, pateiktų literatūroje, atkartojimas. Pirmojo eksperimento metu modelis buvo apmokomas naudojantis autorių pateiktu kodu bei tikrinami prognozių rezultatai su šiais duomenų rinkiniais. BERT-DST autorių teigimu, modelis geba pasiekti apie 69,3 % jungtinio siekio tikslumą su DSTC2 duomenų rinkiniu, bei 87,7 % su WOZ 2.0 duomenų rinkiniu. 7 lentelėje pateikiama minimali naudotų duomenų rinkinių statistika.

7 lentelė. DSTC2 ir WOZ 2.0 duomenų rinkinių statistika

Pavadinimas	Įrašų skaičius (apmokymo rinkinys)	Įrašų skaičius (testavimo rinkinys)
DSTC2	11677	9890
WOZ 2.0	2536	1646

Modelis buvo apmokomas su DSTC2 duomenų rinkiniu apie 30 minučių, per tą laiką spėta modelį apmokyti su apie 12000 partijų, arba 96000 įrašų (kiekvienoje partijoje 8 įrašai). Taigi, apmokymo aibė panaudota apie 8-9 kartus, t.y. apmokymas turėjo 8-9 epochas. Įdomu tai, jog BERT-DST autoriai modelį apmokė su 100 epochų (tai nepaminėta darbe), kas, autoriaus nuomone, indikuoja stiprų modelio permokymą.

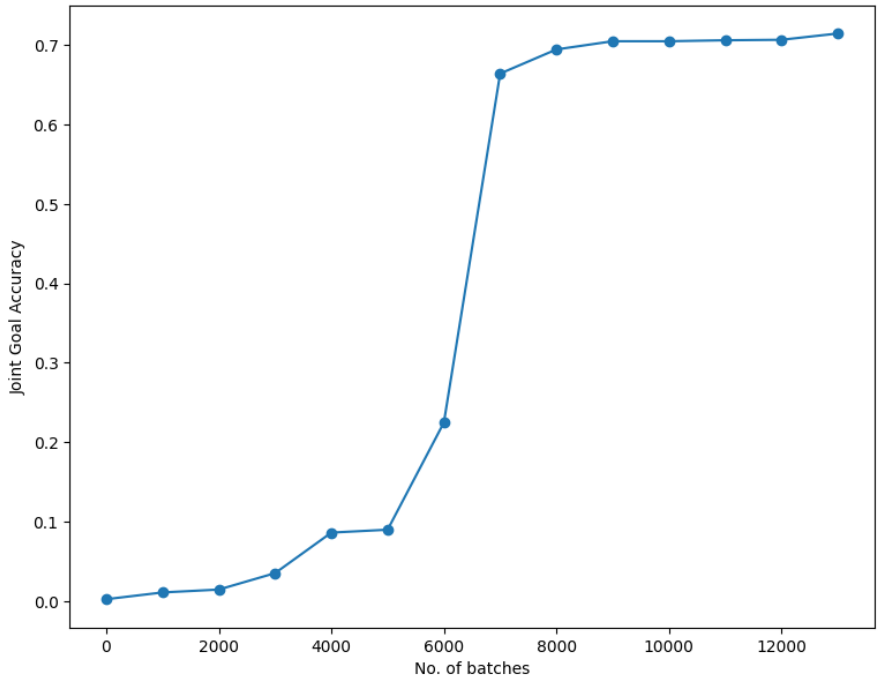
Apmokymo metu kas kiekvienas 1000 partijų yra išsaugomi modelio parametrai. Tikrinimo metu tikrinami visi išsaugoti parametrai su testavimo duomenų aibe. Tikrinimo metu pasiektas didžiausias 47,8 % jungtinio siekio tikslumas, vidutinis siekio tikslumas – apie 77 %, kas reiškia, jog modelis parenka parenka teisingas kintamųjų ir jų reikšmių poras (s, v) 77 kartų iš 100. Rezultatų tikrinimo grafikas matyti 12 paveikslėlyje.



12 pav. Modelio rezultatai, apmokius su DSTC2 duomenų rinkiniu

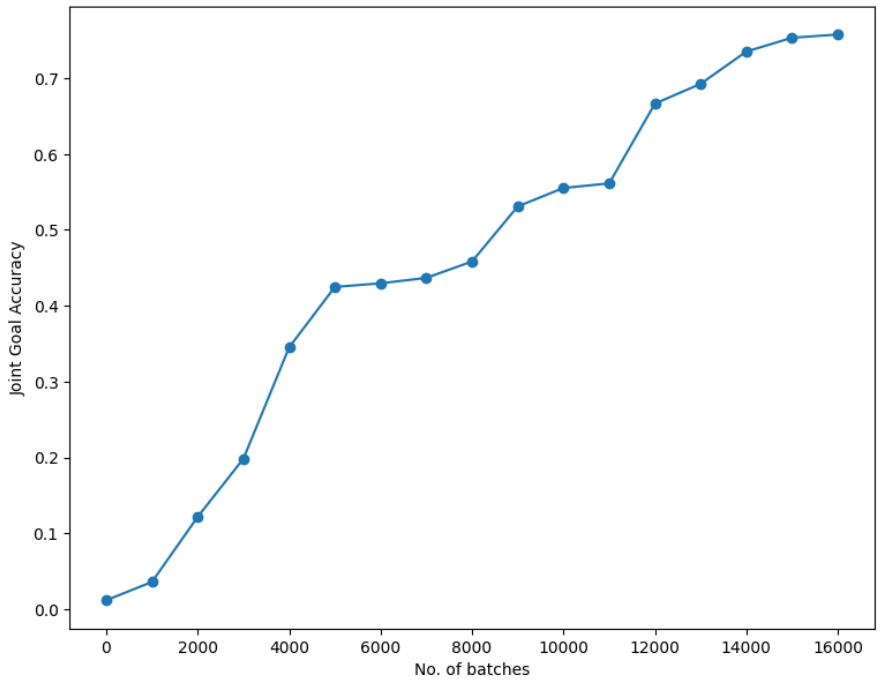
Tai, žinoma, yra žymiai mažesnis nei autorių išsakytas 69,3 % tikslumas, tačiau tai greičiausiai dėl to, jog autoriai modelį mokino virš 10 kartų ilgiau. Iš grafiko atrodo, jog tikslumo augimas stipriai sumažėjo po 5000 partijų, pasiekta apie papildomus 5 procentus tikslumo per kitas 5000 partijų. Sprendžiant iš grafiko, tikėtina, jog tikslumas gali augti ir daugiau.

Toliau modelis buvo apmokomas ir tikrinamas su Woz 2.0 duomenų aibe. Rezultatų tikrinimo grafikas matyti 13 paveikslėlyje. Iš šio grafiko matyti, jog augimas stipriai sulėtėjo ties 8000-9000 partijų rėžiu. Didžiausias pasiektas jungtinio siekio tikslumas – 71,4 % (autorių skelbtas tikslumas – 87,7 %). Tačiau vėl labai svarbu paminėti, jog autoriai modelį apmokė su 100 epochų, kai šiame darbe modelis buvo apmokytas su iki 50 epochų.



13 pav. Modelio rezultatai, apmokius su WOZ 2.0 duomenų rinkiniu

Galiausiai buvo pabandyta apmokyti modelį iš pradžių su DSTC2 duomenų rinkiniu, vėliau, nuo tam tikro taško (11 000-osios partijos) apmokyti su Woz 2.0 rinkiniu. Įdomu tai, jog šis hibridinis apmokymo būdas suteikė tikslesnius rezultatus (75,7 proc), tikrinant su Woz 2.0 testavimo duomenų aibe, nei su tais pačiais duomenimis tikrinant modelį, apmokytą tik Woz 2.0 duomenimis. Grafikas matyti 14 paveikslėlyje.



14 pav. Modelio rezultatai, apmokius su DSTC2 ir su WOZ 2.0 duomenų rinkiniais

Tai galima paaiškinti tuo, jog DSTC2 duomenų rinkinys turi 5 kartus daugiau duomenų, nei WOZ 2.0, kitaip tariant, kuo daugiau atlikta partijų, tuo naudingos informacijos, esančios WOZ

2.0 rinkinyje, mažėja, lyginant su DSTC2. Nepaisant to, šie rezultatai indikuoja, jog sėkmingas modelis yra tas, kuris yra apmokytas iš skirtingų duomenų rinkinių.

Atminties naudojimas bei apmokymo ir skaičiavimų trukmė. Pirmieji eksperimentai buvo vykdomi naudojantis procesoriumi, kadangi nepavyko integruoti vaizdo plokštės pagalbos kartu su tensorflow karkasu. Apmokant su procesoriumi, apmokymo greitis tesiekė 2,3 pavyzdžių per sekundę (čia dialogo pavyzdys yra vienas naudotojo bei sistemos pasisakymas). Integravus vaizdo plokštę pasiektas net apie 47 pavyzdžių per sekundę greitis.

1 partiją sudaro 8 įrašai, tad apmokyti su 16000 partijų užtrukdavo apie 45 min. Tuo tarpu patikrinti modelį su visa Woz 2.0 testine duomenų aibe užtruko apie 11 sekundžių (apie 150 įrašų per sekundę). Šioje vietoje verta paminėti, jog BERT-DST vykdymas yra žymiai spartesnis, nei [LTB+20]: eksperimentai parodė, jog minėto modelio vykdymo greitis yra iki 10 įrašų per sekundę.

Nors pamatyti, kiek atminties užima kiekviena prognozė atskirai po vieną, nepavyko, tačiau patikrinta, kiek atminties užima pats modelis. Vos užkrovus, naudojama vaizdo plokštės atmintis pakyla nuo 530 MB iki 6870 MB, kas yra žymiai daugiau, nei anksčiau aprašyto modelio, užimančio apie 1270 MB. Šis skirtumas, matyt, kyla dėl labai skirtingų modelio įgyvendinimų (praeitas modelis naudoja PyTorch, šis – Tensorflow), tačiau verta panagrinėti, kodėl taip iš tikrųjų yra. Taip pat eksperimentai parodė, jog partijų dydis gali turėti įtakos, kiek modelis užima atminties: jei partijos dydis yra 16, gaunama *out of memory* išimtis (naudojant 8 GB atminties turinčią vaizdo plokštę), tačiau su bet kokiu kitu partijos dydžiu, mažesniu nei 8, išnaudojama beveik tiek pat atminties.

Toliau pateikiami keli autoriaus pastebėjimai, apibendrinantys sprendimą:

1. Kiekvienas dialogo kintamasis, kurio reikšmes siekiama atrasti, reikalauja tiesinio sluoksnio su skirtingais parametrais. Tai reiškia, kad modelio bendras parametrų skaičius gali sparčiai išaugti turint didelę ontologiją, taip pat – laikas, reikalingas modelį apmokyti, taip pat – reikalingi didesni resursai. Viena iš tolimesnių darbų krypčių gali būti sprendimų, kaip tai optimizuoti, ieškojimas.

2. Prielaida, jog dialogo kintamųjų reikšmės slypi tekste, nėra visada validi. Kartais pasisakymas apie objektą šneka netiesiogiai (darbe pateiktas pavyzdys: *my wife thinks she likes international, but I don't want to take out a loan*. Net jei ir BERT modeliui pašnekovo intenciją pavyktų suprasti, šis noras pažodžiui išimtas iš konteksto ir išsaugotas būsenoje neturėtų jokios prasmės).

3. Nors šis sprendimas leidžia tokias reikšmes, kurios nebūtinai yra apibrėžtos ontologijoje, priklausomybės nuo ontologijos neišvengiama – t.y. žinios apie galimus kintamuosius yra vis tiek reikalingos. Norint pasiekti pilną dinamiškumą, priklausomybės nuo ontologijos galėtų būti atsisakyta – tuomet nebūtų jokių apribojimų dialogo būsenos apskaičiavime.

Kaip ir programavimo kalbose plačiai naudojama *enum* duomenų struktūra, kategorinės reikšmės dažnai turi tik kelis užrašymo būdus, kuriuos svarbu būsenoje išsaugoti teisingai pažodžiui (pavyzdžiui, skrydžio klasę perkant lėktuvo bilietą), kadangi ta informacija gali būti toliau siunčiama į kitas sistemas naudojantis įvairiomis sąsajomis, pvz., REST API. Remiantis tuo, siūloma šiuos

du būdus apjungti į vieną – tai yra, kai įmanoma, naudotis kategorinių reikšmių radimu generuojant kandidatus, o kada tai neveikia – naudotis reikšmės ištraukimo iš pačio teksto metodu. Kadangi du aprašyti metodai yra gan panašūs ir abu savo pagrinde naudoja BERT modelį, juos apjungti nėra itin sudėtinga.

3.3.3. Bandymai su šiame darbe siūloma supaprastina būsenos radimą tekste atliekančia architektūra

Šiame poskyryje aprašomi atlikti bandymai, siekiant patikrinti, kokią įtaką rezultatams daro architektūros patobulinimai, aprašyti 2.1 poskyryje.

Bandymai su klasifikavimo sluoksnių supaprastinimu. Pirmiausia patikrinta, kokią įtaką architektūros supaprastinimas iš kelių klasifikavimo sluoksnių į vieną, reikalingą informaciją užkoduojant įeityje, daro rezultatų tikslumui.

Šiam bandymui atlikti buvo sukurtas supaprastintas modelis, veikiantis pagal [CL19], tik apmokytas su mažiau duomenų. Apmokymas vyko taip: pirmiausia apmokomas BERT modelis su tiesiniu klasifikavimo sluoksniu ant duomenų, kurių būsenos kintamasis yra *area*. Tuomet šis modelis apmokomas su kitu, skirtingu tiesiniu klasifikavimo sluoksniu su kitais duomenimis, kurių būsenos kintamasis yra *food*. Rezultate turimas vienas BERT modelis ir kelis klasifikavimo sluoksnius (autorius patarė naudoti vieną BERT modelį, kadangi dėl žinių dalijimosi tarp skirtingų kintamųjų gaunami geresni rezultatai).

Turint supaprastintą [CL19] architektūra paremtą modelį, su tais pačiais duomenimis ir parametrais apmokomas kitas modelis, tačiau šį kartą naudojamas vienas klasifikavimo tinklas, o apmokoma su duomenimis iš $area$ ir $food$ vienu metu. Informacija apie tai, koks kintamasis, užkoduojama į įeities vektorių (vietoje kelių klasifikavimo sluoksnių). Rezultate gaunamas naujas modelis.

Literatūroje aprašyto modelio kopijos ir naujo siūlomo – rezultatai palyginami, atliekant būsenos radimo problemos uždavinį su tik *food* duomenimis, taip pat su tik *area* duomenimis, ir su *area* bei *food* duomenimis. Lentelėje 8 pateikiami rezultatai.

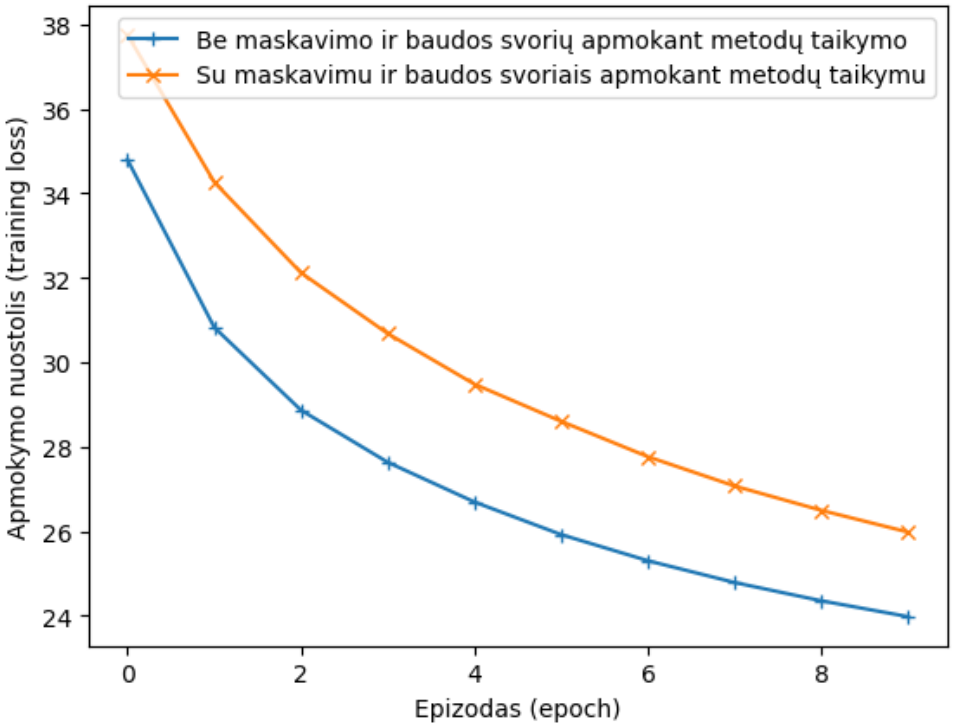
8 lentelė. Modelių, apmokytu senuoju (pagal [CL19]) būdu bei naujuoju palyginimas

Modelio apmokymas	**Duomenų tipas**	**Būsenos spėjimo tikslumas (proc.)**
Senuoju būdu	area	89,1
Nauju būdu	area	91,9
Senuoju būdu	food	80,4
Nauju būdu	food	82,6
Senuoju būdu	area, food	73,37
Nauju būdu	area, food	76,02

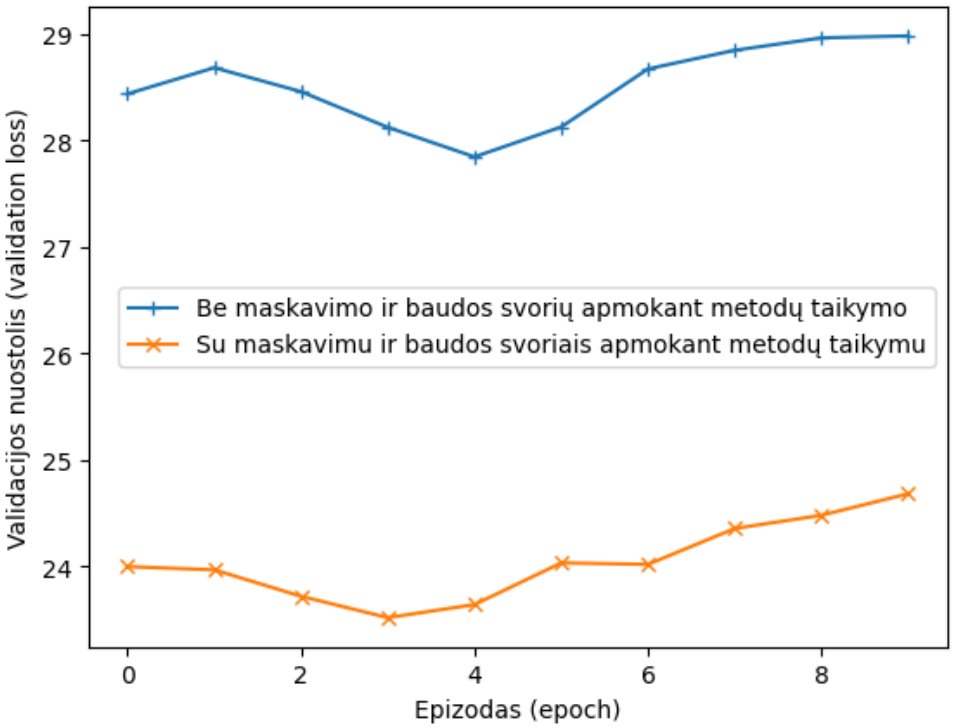
Kaip matyti iš rezultatų 8 lentelėje, naujasis apmokymo būdas ne tik supaprastina architektūrą, bet ir pagerina rezultatus. Verta paminėti, jog tai nėra tiesioginis palyginimas su [CL19] rezultatais – verčiau modelio, kuris yra sukurtas pagal autorių aprašytą architektūrą (ir apmokytas su

mažiau duomenų) bei modelio, apmokyto pagal šiame darbe pasiūlytą architektūrą. Siekiant daugiau užtikrinimo, tolimesniuose darbuose siūloma apmokytą naująjį modelį palyginti tiesiogiai su egzistuojančiu išleistu modeliu.

Maskavimo metodikų patikrinimas. Taip pat patikrinta, kaip 2.1.1 poskyryje aprašytos maskavimo metodikos padeda pagerinti rezultatus, kaip matyti 15 bei 16 paveikslėliuose.

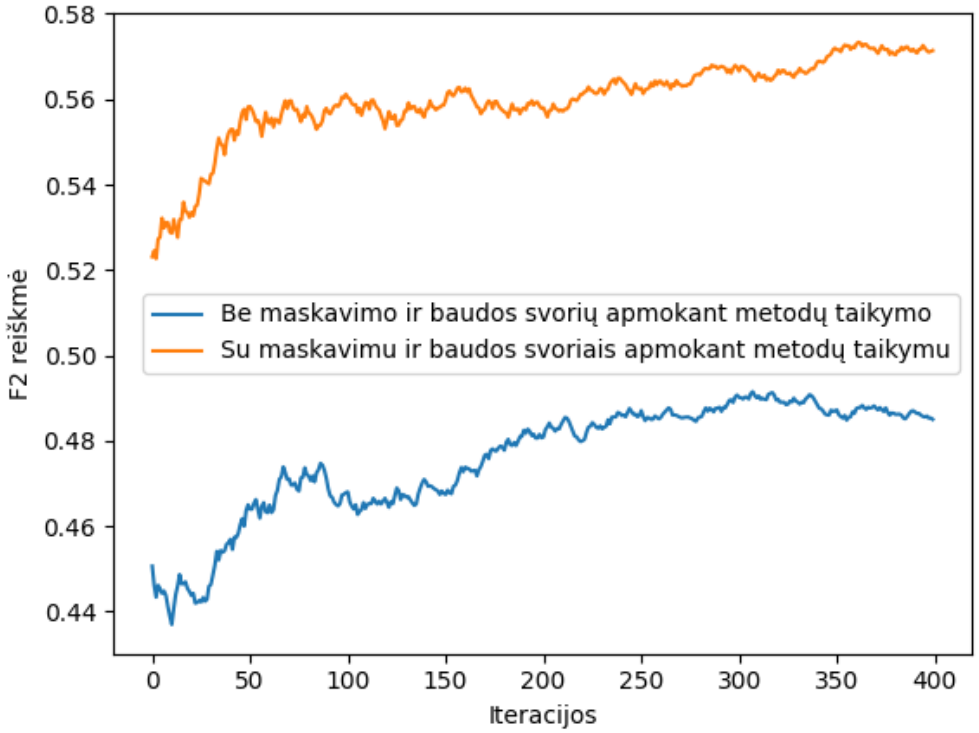


15 pav. Apmokymo nuostolių reikšmių palyginimas su maskavimu ir baudos svoriais apmokant metodų taikymu, bei be jų



16 pav. Validacijos nuostolių reikšmių palyginimas su maskavimu ir baudos svoriais apmokant metodų taikymu, bei be jų

Iš šio grafikų matome, jog apmokymo nuostolių funkcijos reikšmės padidėja naudojant aprašytas metodikas, kas nėra nuostabu, kadangi pakeičiama nuostolių kriterijų funkcija. Tačiau validacijos paklaidos reikšmės gan stipriai nukrenta. Taikomų metodų privalumas akivaizdus tikrinant F2 reikšmių grafiką, kaip matyti 17 paveikslėlyje.



17 pav. F2 funkcijos reikšmių palyginimas su maskavimu ir baudos svoriais apmokant metodų taikymu, bei be jų

Per tą patį apmokymo laiką taikomi metodai padeda pasiekti beveik dešimčia procentų geresnį rezultatą.

3.3.4. Bandymai su šiame darbe siūloma hibridine architektūra

Čia aprašomi atlikti bandymai, siekiant patikrinti, kokią įtaką rezultatams daro architektūros patobulinimai, aprašyti 2.2 poskyryje.

Tam, kad patikrinti architektūros pranašumą, apmokyti du modeliai – kandidatų generavimo ir būsenos radimo tekste. Jie buvo apmokyti skirtingais parametrais bei duomenimis, kadangi norėta įsitikinti, jog hibridinė architektūra teikia pranašumų nepaisant to, kaip apmokyti modeliai. Tikrinta su WOZ 2.0 duomenimis.

Vietoje to, kad modeliai būtų tikrinami vienu kartu, išgaunami modelių spėjimai nepriklausomai vienas nuo kito. Abu modeliai sugeneravo po 830 būsenos spėjimų. Tikrinamas kandidatų modelio spėjimų tikslumas (tikrinant žingsnio užklausos tikslumą) – 63,9 %, būsenos radimo tekste – 59,5 %.

Toliau taikomas šis algoritmas: kuriamas naujas spėjimų sąrašas. Į i-tąjį sąrašo elementą įdedamas kandidato generavimo modelio spėjimas, išskyrus tada, kai jis tuščias. Jei spėjimas yra tuščias, dedamas būsenos radimo tekste spėjimas. Tai reiškia, jog galimai kandidatų generavimo modelis suklydo ir Gaunamas sąrašas atitinka hibridinės architektūros spėjimų sąrašą,

Neteisingi spėjimai. Didžiausia šios architektūros spraga yra tai, jog daroma prielaida, kad tuščią spėjimą grąžinęs pirmasis modelis suklydo. Gali būti, kad iš tiesų tą ėjimą nebuvo jokio spėjimo. Pavyzdžiui, dialoge, kur naudotojas padėkojo už pagalbą, kandidatų generavimo modelis

nerado jokios būsenos (kaip ir turėtų būti), tačiau būsenos radimo tekste modelis rado. Vadinasi, hibridinė architektūra darytų klaidingą spėjimą.

To būtų išvengiama, jeigu antrasis modelis taip pat teisingai grąžintų tuščią spėjimą. Tai galima pasiekti sugriežtinant antrojo modelio spėjimus. Kiekvienas spėjimas yra tikimybė, jog tam tikras žodis reprezentuoja būseną. Parametrą, reiškiantį mažiausią priimtiną tikimybę (angl. *threshold*), galima konfigūruoti. Į jį galima žiūrėti kaip į filtrą: kuo didesnė parametro reikšmė, tuo didesnis filtras ir sunkiau praleidžiami spėjimai. Vadinasi, sumažinama atvejų, kai atliktas prastas spėjimas sugadina teisingą.

Toliau pateiktoje 9 lentelėje pateikiami pirmojo modelio, antrojo modelio bei hibridinio modelio tikslumai su WOZ 2.0 duomenų rinkiniu konfigūruojant slenksčio parametrą.

9 lentelė. Skirtingų modelių tikslumo palyginimas keičiantis slenksčio parametrui apmokius WOZ 2.0 duomenų rinkiniu

Modelis	**Tikslumas**	**Būsenos radimo tekste slenksčio parametro reikšmė**
Kandidatų generavimo	63,9%	0,3
Būsenos radimo tekste	60,1%	0,3
Hibridinis	62,7%	0,3
Kandidatų generavimo	63,9%	0,5
Būsenos radimo tekste	59,5%	0,5
Hibridinis	63,3%	0,3
Kandidatų generavimo	63,9%	0,7
Būsenos radimo tekste	60,1%	0,7
Hibridinis	63,9%	0,7
Kandidatų generavimo	63,9%	0,9
Būsenos radimo tekste	59,5%	0,9
Hibridinis	63,73%	0,9

Iš šių rezultatų matyti, jog nustačius gan aukštą ribą (0,7), hibridinės architektūros spėjimų tikslumas sutampa su kandidatų generavimo tikslumu. Įdomu tai, jog slenksčio parametro reikšmė 0,9, nors ir reiškianti, kad priimami tik užtikrinti spėjimai, sumažina hibridinio modelio spėjimų tikslumą.

Lentelėje 10 pateikiami tikslumai su DSTC2 duomenų rinkiniu.

10 lentelė. Skirtingų modelių tikslumo palyginimas keičiantis slenksčio parametrui apmokius DSTC2 duomenų rinkiniu

Modelis	**Būsenos radimo tikslumas**
Kandidatų generavimo	73,8%
Būsenos radimo tekste	76,05%
Hibridinis	77,4%

Tikrinant su DSTC2 rinkiniu, pastebėta, jog hibridinė architektūra leidžia pasiekti geriausius

rezultatus. Klaidų analizės metu pastebėta, jog pasitaiko atvejų, kai kandidatų tikrinimo modelis neatspėja būsenos, tuo tarpu būsenos radimo tekste nurodo teisingą būseną.

Deja, bet nei DSTC2, nei WOZ 2.0 duomenų rinkiniai nėra tinkami teisingai įvertinti hibridinės architektūros tikslumui, kadangi būsenos kintamieji ir reikšmės yra apibrėžti ontologijoje, tad visą laiką būsena slypi viename iš kandidatų. Nėra tokio dialogo, kuriame būseną būtų galima rasti tik tekste – būtent tais atvejais kandidatų tikrinimo modelis suklystų ir būsenos radimo tekste modelis pateiktų teisingą atsakymą. Tam, kad būtų galima galutinai įsitikinti šio modelio verte, galima sugeneruoti tokius duomenis, tačiau darbo metu pritrūko tam laiko.

3.3.5. Bandymai ieškant optimalių parametrų modeliams

Toliau pateikti aprašymai bei rezultatai atliktų bandymų, kurių metu siekta suprasti 1) kokį efektą daro įvairios apmokymo metodikos bei konfiguruojamų parametrų dydžiai ir 2) kokie yra optimalūs parametrai tolimiems modelių apmokymams.

Visi bandymai – išskyrus ten, kur parašyta kitaip – atlikti su kandidatų generavimo architektūra, aprašyta 1.4.1 poskyryje.

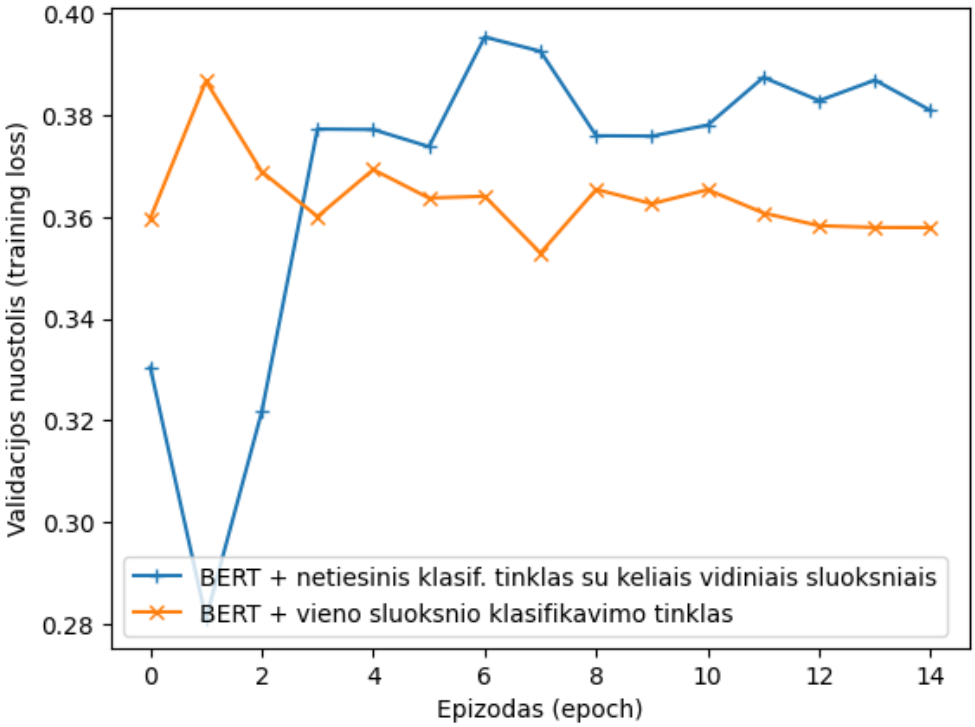
3.3.5.1. Tiesinio sluoksnio sudėtingumas

Architektūroje, pristatytoje [LTB+20], galutinis BERT sluoksnis tėra vienas tiesinis sluoksnis. Tačiau kyla klausimas, ar tai optimalu? Praktika rodo, jog daugiau vidinių sluoksnių padeda „giliau“ apmokyti modelį. BERT grąžinami vektoriai turi 768 komponentus (mašininio mokymosi terminais – dimensijas), tad kadangi kiekviename vektoriuje slypi nemažai informacijos, peršasi mintis, jog galbūt klasifikavimo sluoksnis galėtų būti ne tiesinis.

Atliktas eksperimentas. Eksperimento parametrai: 15 epizodų (angl. *epochs*). Duomenų rinkinio dydis – 16. Apmokymui skirta 8000 įrašų, o validacijai – 1000.

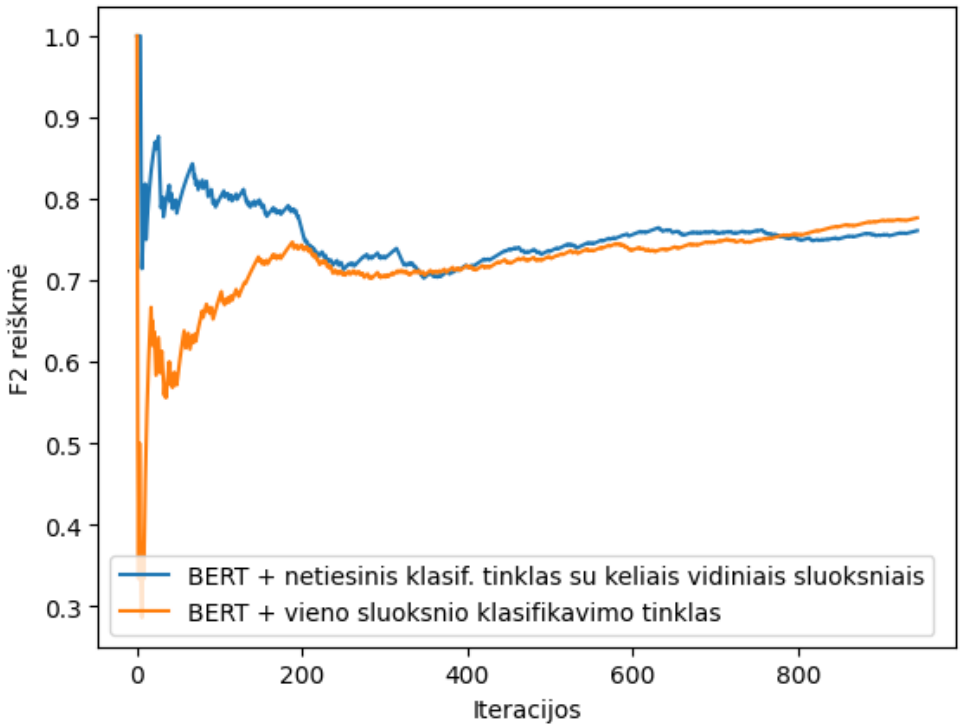
Apmokymo metu nuostolių funkcijos reikšmės parodė, jog papildomi vidiniai sluoksniai įtakos mažai daro. Validacijos nuostolių funkcijos reikšmės tapo didesnės. Peršasi išvada, jog taip dėl to, nes atsiranda daugiau parametrų – vadinasi, reikia daugiau informacijos, kad apmokyti. Galiausiai F2 funkcijos reikšmės taip pat parodė jog nėra privalumų.

Validacijos nuostolių funkcijos reikšmės, lyginant tiesinį sluoksnį su netiesiniu matyti 18 paveikslėlyje:



18 pav. Validacijos nuostolių funkcijos reikšmės

F2 funkcijos reikšmės, lyginant tiesinį sluoksnį su netiesiniu matyti 19 paveikslėlyje:



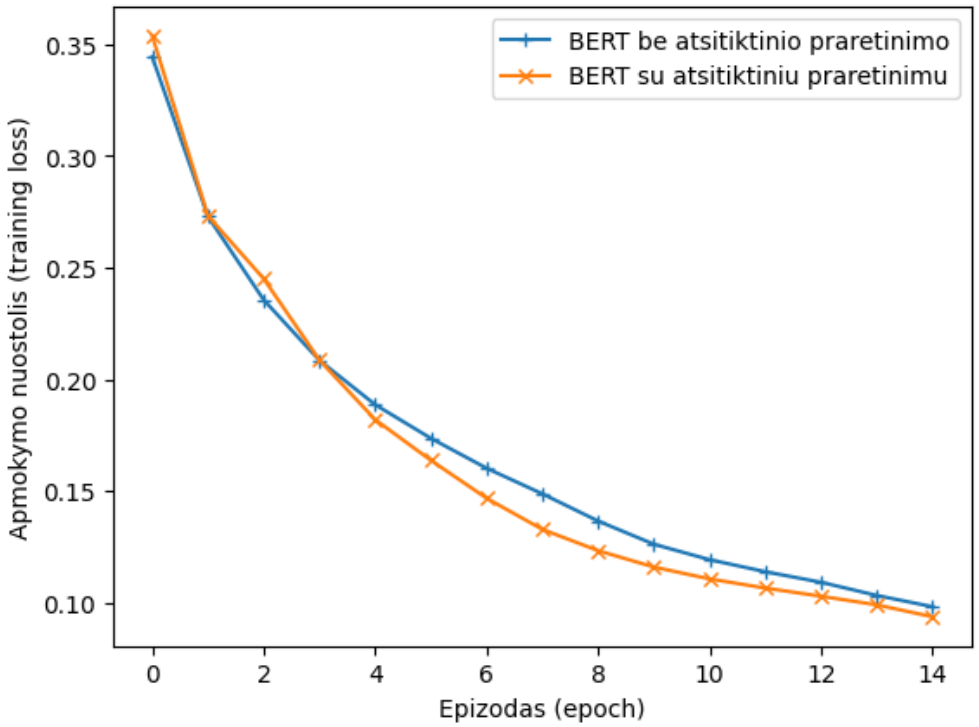
19 pav. F2 funkcijos reikšmės

Taigi, remiantis tuo, jog netiesinis klasifikavimo tinklas nepagerina F2 rezultatų, toliau šiame darbe naudojamas paprastas tiesinis klasifikavimo tinklas.

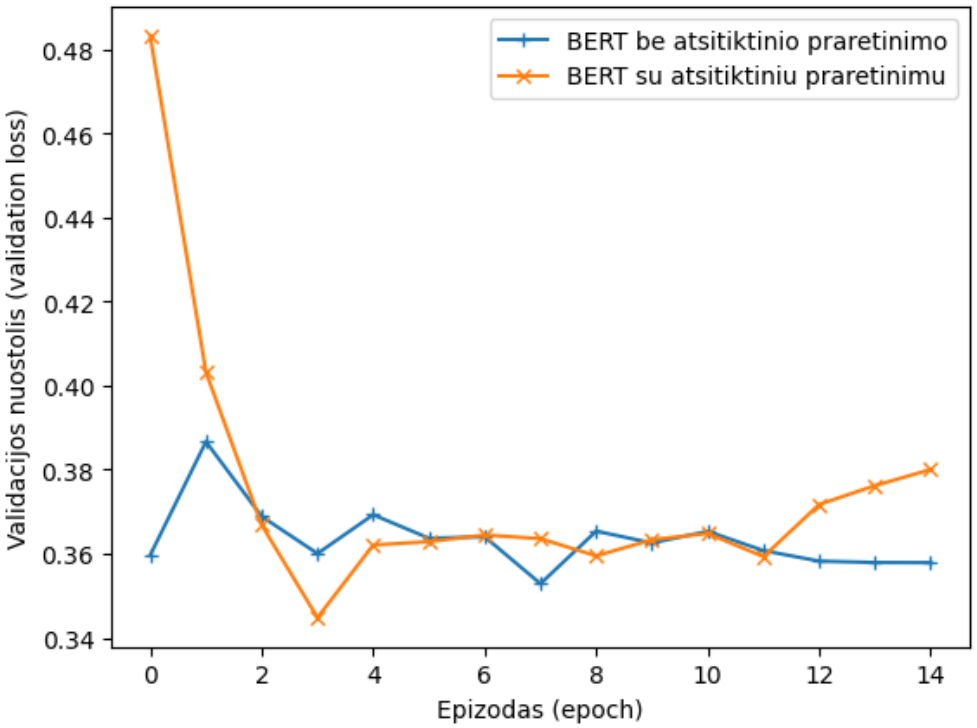
3.3.5.2. Atsitiktinio praretinimo transformacija

Toliau šio darbo metu buvo tikrinama, kiek naudingas yra atsitiktinis praretinimas (angl. *dropout*). Naudojantis atsitiktinio praretinimo transformacija (angl. *dropout*), kurios dydis yra 0,1, įrašų sklidimo į priekį (angl. *forward pass*) metu 1 iš 10 BERT sukuriamų vektorių įterpinių yra išmetami. Tai gali padėti išvengti permokymo, kai kartojasi tam tikri duomenys ar šablonai.

Validacijos bei apmokymo nuostolių funkcijų reikšmės nerodo didelio skirtumo:

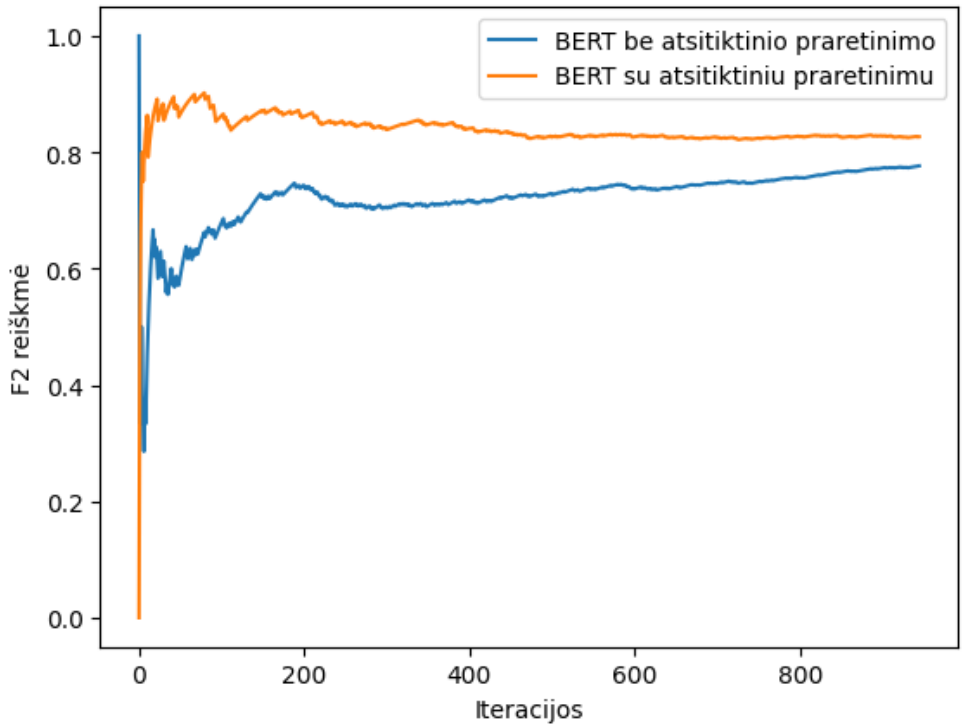


20 pav. Apmokymo nuostolių funkcijos reikšmės su ir be atsitiktinio praretinimo transformacijos



21 pav. Validacijos nuostolių funkcijos reikšmės su ir be atsitiktinio praretinimo transformacijos

Tačiau F2 funkcijos reikšmės su atsitiktinio praretinimo transformacija gerokai viršija funkcijos reikšmes be jos, kaip matyti 22 paveikslėlyje:



22 pav. F2 reikšmės su ir be atsitiktinio praretinimo transformacijos

Remiantis šiais rezultatais galima teigti, jog atsitiktinio praretinimo transformacijos naudojimas pasitvirtina: su tuo pačiu duomenų kiekiu, atsitiktinio praretinimo transformacija leido pasiekti apie

5 % didesnę F2 funkcijos reikšmę.

3.3.5.3. Perdėtas atsitiktinis atrinkimas

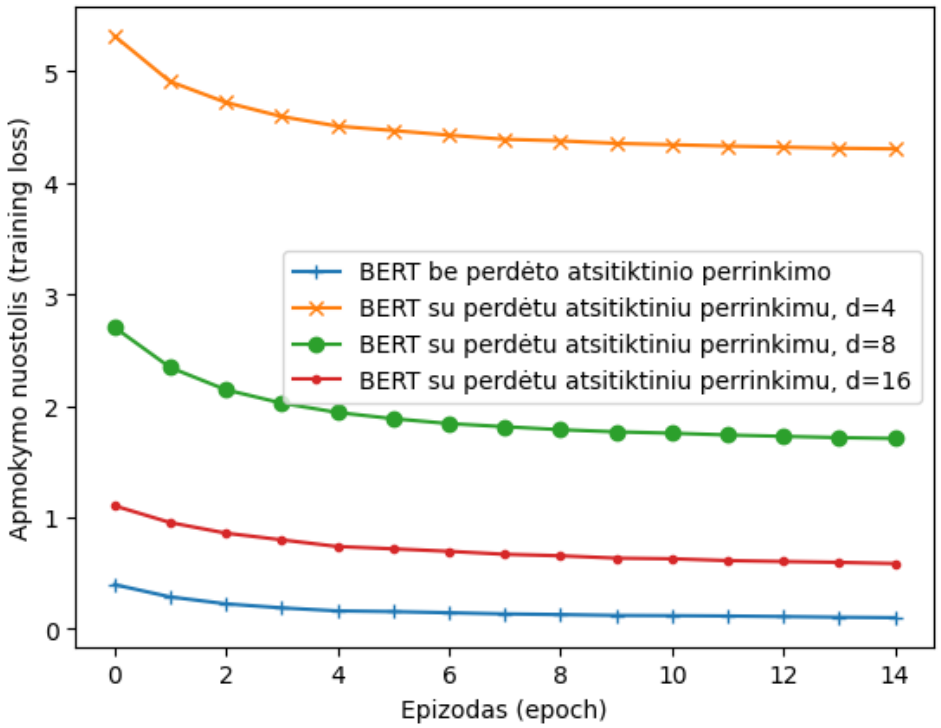
Kai generuojami kandidatai, teigiamų ir neigiamų skaičius yra neproporcingas. Pvz., jeigu ontologijoje kintamasis *area* turi 5 galimas reikšmes, bus sugeneruoti 5 kandidatų kintamajam *area*. Tačiau tik vienas iš jų bus teigiamas (pvz., *north*). Visi kiti (*south, east, center* ir t.t.) bus neigiami. Vadinasi, bus 1 teigiamas ir 4 neigiami pavyzdžiai. Tai tampa problema, kai kiekvienas kintamasis turi daug daugiau galimų reikšmių, kadangi modelis geriau išmoksta atpažinti neigiamas reikšmes, ir prastai atpažįsta teigiamas.

Kad išvengti šio imbalanso verta pagalvoti apie perdėtą atsitiktinį atrinkimą (angl. *oversampling*). Šiuo metodu siekiama, kad neigiami ir teigiami pavyzdžiai būtų išskirstyti kuo tolygiau. Kad tą pasiekti, galima su tam tikra tikimybe teigiamą pavyzdį parinkti papildomą kartą apmokymo metu.

Pasirinkus papildomus teigiamus pavyzdžius sukuriama papildomų teigiamų pavyzdžių aibė. Tegu parametras $1/d$ parodo, koks šios naujos aibės santykis su neigiamų pavyzdžių aibe. Pvz., jei $d = 8$, tuomet papildoma teigiamų pavyzdžių aibė bus 8 kartus mažesnė, nei neigiamų pavyzdžių. Ši papildoma aibė pridedama prie teigiamų pavyzdžių.

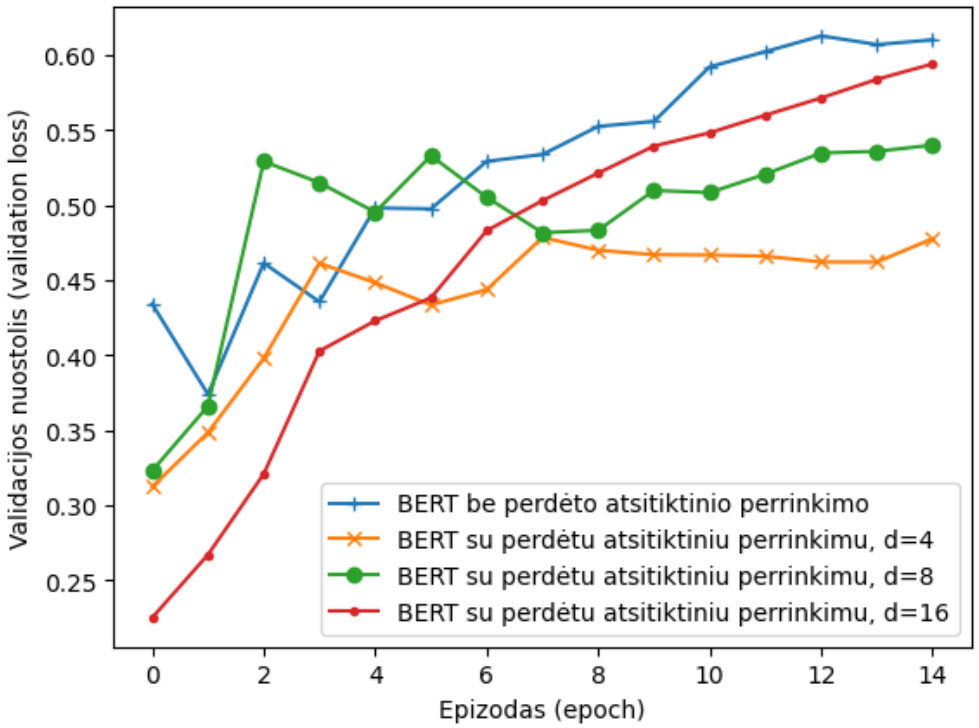
Parametras d gali daryti įtaką ne tik galutiniam modelio tikslumui, bet ir permokymui. Kuo mažesnis d, tuo teigiamų pavyzdžių aibė bus artesnė neigiamų pavyzdžių aibei, tačiau kadangi teigiami pavyzdžiai kartojasi, gali būti, jog modelis bus greitai permokamas. Bandymų metu bandyta nustatyti, kokia d reikšmė tinkamiausia.

Apmokymo nuostolius tampa sunku palyginti, kadangi keičiasi tikslo funkcija, tuo pačiu ir nuostolių funkcija bei jos reikšmės, kaip matyti 23 paveikslėlyje:



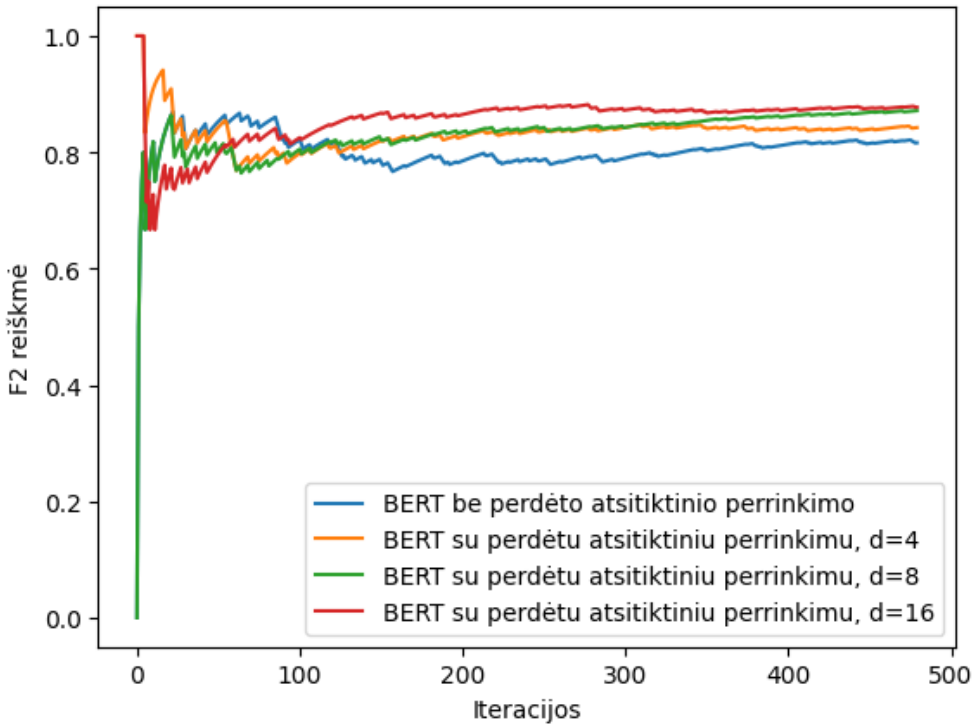
23 pav. Apmokymo nuostolių funkcijos reikšmės su ir be atsitiktinio perdėto atrinkimo ir įvairiomis parametro d reikšmėmis

Tuo tarpu validavimo nuostolių grafikai, pavaizduoti 24 paveikslėlyje, rodo, jog su parametru $d = 4$ validavimo nuostolių funkcijos reikšmės yra ne tik mažesnės, bet ir funkcijos grafikas kyla ne taip sparčiai, palyginus su kitomis reikšmėmis. Parametras $d = 8$ yra antroje vietoje pagal paklaidos reikšmę. Taip pat verta paminėti, jog validavimo nuostolių reikšmės be atsitiktinio perdėto atrinkimo yra prasčiausios – tuomet paklaidos gaunamos didžiausios.



24 pav. Validacijos nuostolių funkcijos reikšmės su ir be atsitiktinio perdėto atrinkimo ir įvairiomis parametro d reikšmėmis

Galiausiai F2 funkcijos, tikrinančios modelio tikslumą, grafikas pavaizduotas 25 paveikslėlyje rodo, jog metodas padeda pasiekti tikslesnius rezultatus. Skirtumas tarp geriausios F2 reikšmės, kai $d = 16$ bei blogiausios reikšmės be atsitiktinio perrinkimo yra apie 6 %



25 pav. F2 funkcijos reikšmės su ir be atsitiktinio perdėto atrinkimo ir įvairiomis parametro d reikšmėmis

Akivaizdu, jog atsitiktinis perrinkimas daro teigiamą įtaką modelio apmokymo metu, tačiau kurią d reikšmę pasirinkti? Iš grafikų panašu, jog $d = 8$ yra optimalus pasirinkimas, jei lyginama pagal validacijos nuostolius (kuo mažiau, tuo geriau) bei F2 reikšmes (kuo daugiau, tuo geriau). Šio parametro reikšmės yra labiausiai „per vidurį“, tad galima sakyti, jog tai – saugiausias pasirinkimas.

Iš šių bandymų rezultatų galima daryti išvadą, jog perdėtas atsitiktinis perrinkimas duoda teigiamus rezultatus; geriausi rezultatai pasiekiami, kai teigiamų pavyzdžių aibė praplečiama dydžiu, lygiu 1/8 neigiamų pavyzdžių aibės dydžiui. Tad toliau šiame bandymuose naudojamas parametras yra $d = 8$.

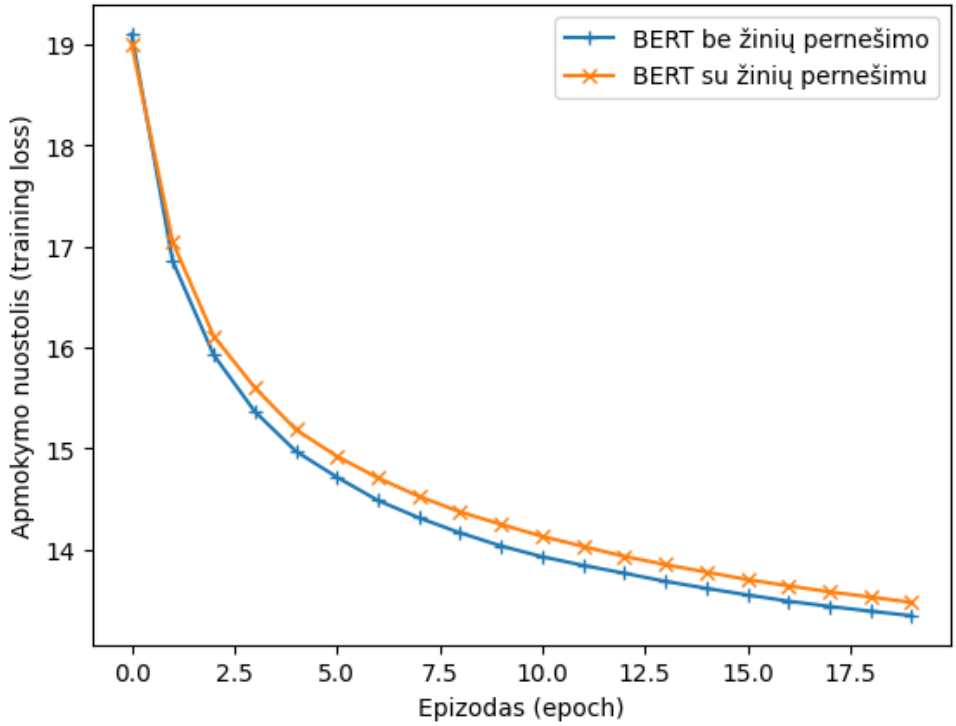
3.3.5.4. Žinių pernešimas

Abu pagrindiniai šiame darbe nagrinėjami būdai (būsenos kandidatų tikrinimas ir būsenos tekste ieškojimas) yra iš esmės labai panašūs: jų siekis – turint tam tikrą įterpinį pasakyti, ar jame slypi informacija apie būseną, ar ne. Didelis skirtumas tarp šių architektūrų yra tai, kaip sąveikaujama tarp BERT ir klasifikavimo sluoksnio. Vienu atveju, į klasifikavimo sluoksnį ateina informacija apie visą sakinį, taip pat – būsenos kandidatą. Kitu atveju klasifikavimo sluoksnis gauna tik vieną žodį bei kintamojo, kurios būseną šis žodis galimai nusako, reikšmę. Paprastai tariant, vienu atveju veikiama ant ištiso sakinio, kitu – ant žodžių atskirai.

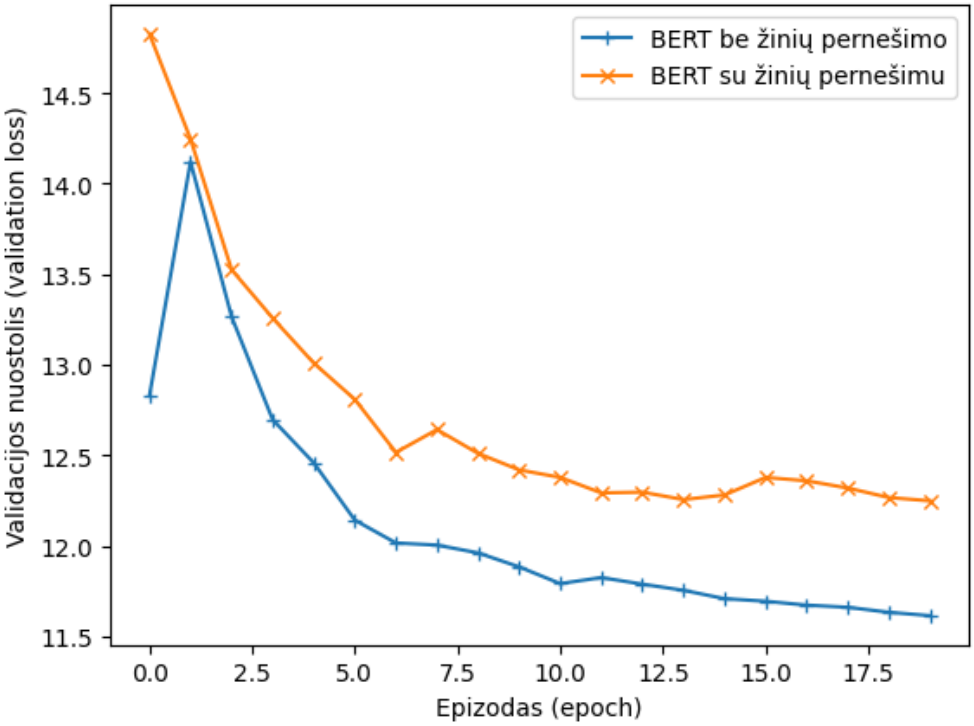
Nepaisant to, duomenys abiem atvejam naudojami vienodi (tik apdorojami skirtingai), abu sprendimai yra klasifikavimo problemos ir abiejų sprendimų klausiamas labai panašus klausimas: kokia yra būsena? Taigi, panašu, kad kuris nors sprendimų yra gan geras kandidatas pernešti žinias į kitą. Kadangi generuojant kandidatus gaunama žymiai daugiau duomenų, tad ir modelį apmokyti gaunasi geriau, kyla klausimas, ar negalima tuo pasinaudoti ir pagerinti kito sprendimo – būsenos radimo tekste – tikslumą?

Tam išsiaiškinti apmokyti keli modeliai: pirmasis – būsenos radimo tekste modelis DSTC2 duomenimis. Šis modelis yra be žinių pernešimo ir bus naudojamas palyginimui. Antrasis – kandidatų tikrinimo modelis, apmokytas taip pat DSTC2 duomenimis. Trečiasis – būsenos radimo tekste modelis, kuris yra apmokomas naudojantis antruoju modeliu, kaip savo pagrindu.

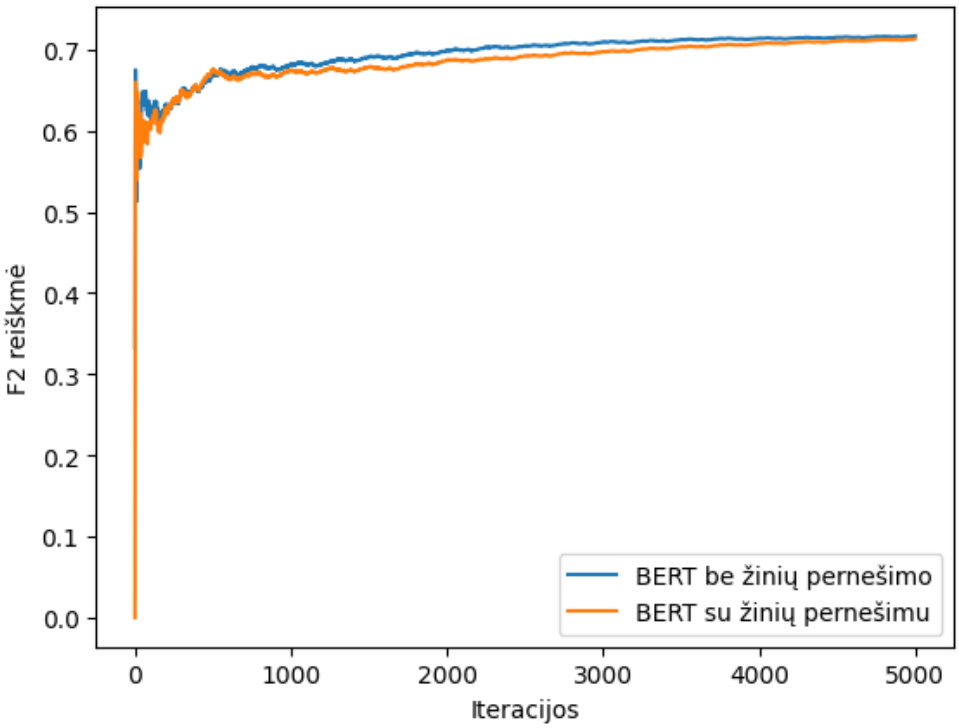
Trečiojo ir pirmojo modelių apmokymo parametrai yra identiški: pagrindinis kalbos apdorojimo modelis – BERT, naudota 47 tūkst. įrašų apmokymui, 20 epizodų (epochų), praretinimo parametras 0,1, nenaudota atsitiktinio perrinkimo. Rezultatai pateikti 26, 27 ir 28 pav.



26 pav. Apmokymo nuostolių funkcijos reikšmės su ir be žinių pernešimu



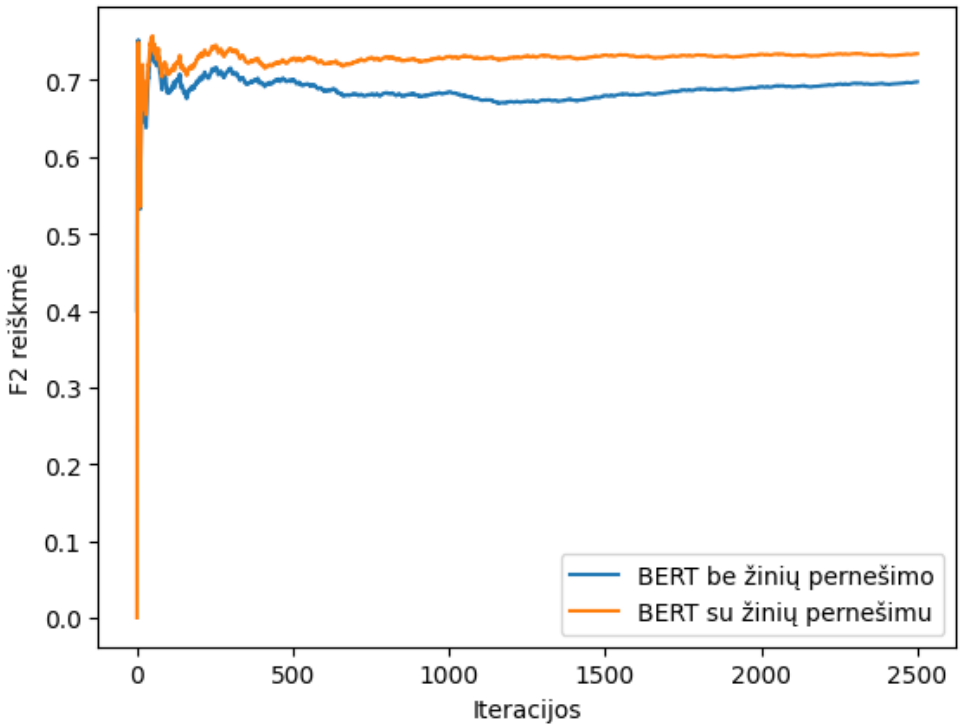
27 pav. Validacijos nuostolių funkcijos reikšmės su ir be žinių pernešimu



28 pav. F2 funkcijos reikšmės su ir be žinių pernešimu

Iš pateiktų grafikų matyti, jog apmokymo metu paklaidos reikšmės mažėja ne taip greitai su žinių pernešimu negu kad palyginus be žinių pernešimo. Taip, matyt, yra todėl, kad žinių pernešimo atveju apmokomas jau apmokytas modelis, tad jam reikia daugiau laiko „persikvalifikuoti“. Ši priežastis galėtų paaiškinti taip pat ir tai, kad validacijos paklaidos reikšmės žinių pernešimo atveju „nepaveja“ paprasto modelio.

Tačiau iš F2 funkcijos grafiko matyti, jog nors ir modelio be žinių pernešimo tikslumas viršija modelio su žinių pernešimu tikslumą, pastarojo tikslumo reikšmės beveik pasivijo kito modelio reikšmes. Nuspręsta eksperimentą pratęsti dar 10-ia epizodų, siekiant nustatyti, ar nuo šios ribos modelio su žinių pernešimu tikslumo reikšmės gali aplenkti modelio be žinių pernešimo tikslumo reikšmes. 29 pav. pateikiamas F2 grafikas.



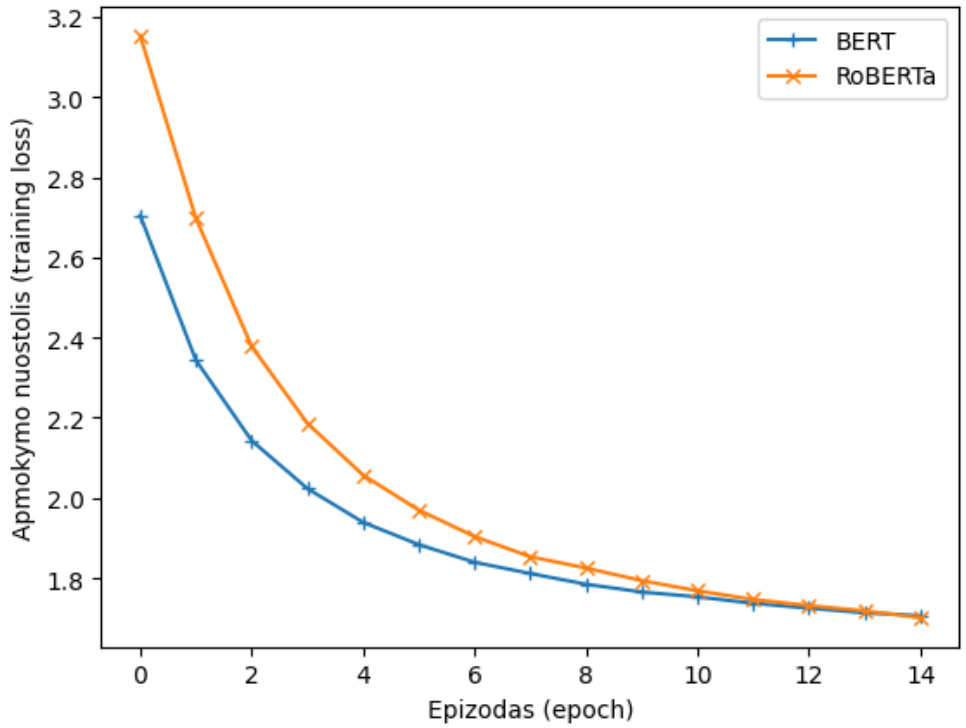
29 pav. F2 funkcijos reikšmės su ir be žinių pernešimu po papildomo apmokymo

Apmokius modelį papildomai, paaiškėjo, jog BERT su žinių pernešimu galutinės F2 reikšmės siekia 0,73, tuo tarpu be žinių pernešimo – 0,69. Vadinasi, galime teigti, jog žinių pernešimas iš modelio, apmokyto tikrinti generuotus kandidatus, leidžia pasiekti geresnius tikslumo rezultatus, kai apmokoma su pakankamai daug duomenų. Šiuo atveju pakankamai daug yra 46 tūkst. įrašų ir 30 epizodų (epochų). Kitų skirtumų tarp modelių, pvz. dydžio ar apmokymo trukmės, nepastebėta.

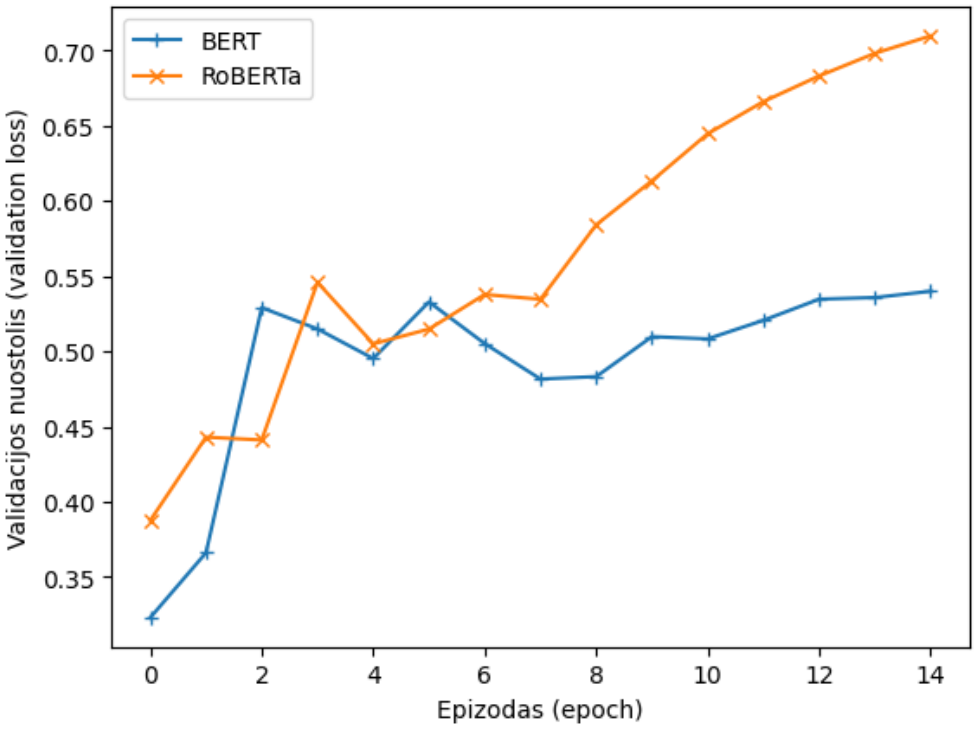
3.3.5.5. RoBERTa

Darbo metu atlikti bandymai, siekiant patikrinti hipotezę, jog RoBERTa modelis duoda galimybę pasiekti tikslesnius rezultatus, lyginant su BERT. Bandymų metu buvo sprendžiamas kandidatų tikrinimo uždavinys su identiškais parametrais, išskyrus pagrindinį teksto apdorojimo modelį, kuris vieno bandymo metu buvo BERT, o kito – RoBERTa.

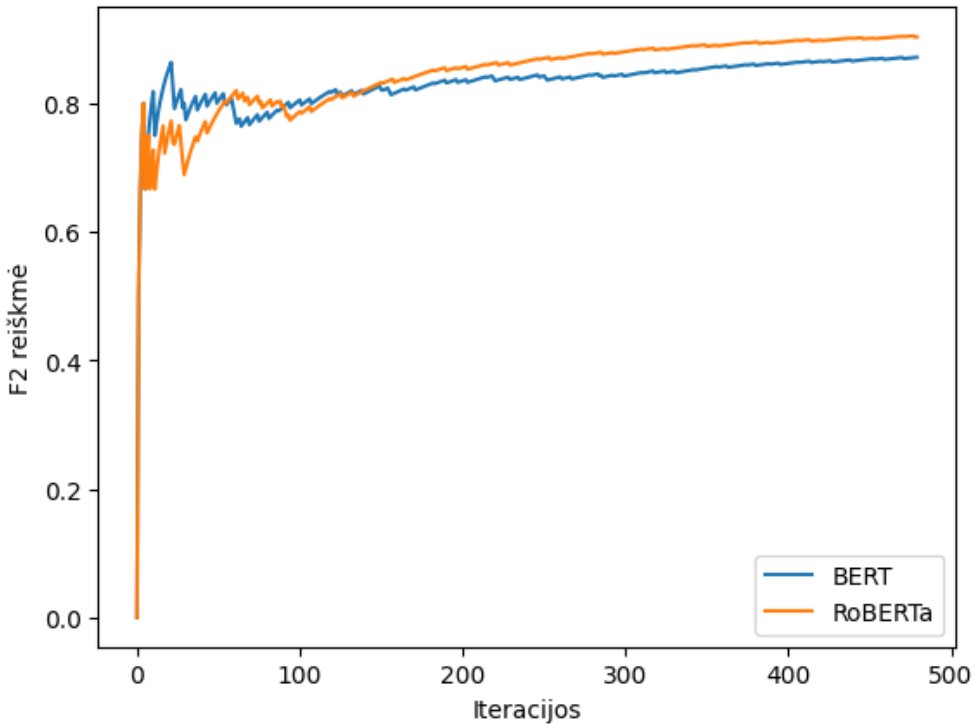
Bandymo parametrus buvo bandoma parinkti taip, kad apmokymas neužtruktų labai ilgai, tačiau duomenų būtų pakankamai, jog atspindėtų rezultatus apmokant su visa duomenų aibe. Apmokymui skirta 15 epizodų (epochs), kiekviename epizode po 3000 pokalbių pavyzdžių, rinkinio dydis (batch) – 16, atsitiktinio praretinimo tikimybė 0,1, atsitiktinio perrinkimo dydis – 8 (papildomų teigiamų pavyzdžių kiekis lygus 1/8 neigiamų pavyzdžių aibės dydžio). Rezultatai pateikti 30, 31 ir 32 pav.



30 pav. BERT ir RoBERTa apmokymo nuostolio palyginimas



31 pav. BERT ir RoBERTa validacijos nuostolio palyginimas



32 pav. BERT ir RoBERTa F2 reikšmės palyginimas

Iš pateiktų figūrų matyti, jog apmokymo nuostolio kritimas sabejais atvejais yra panašus, o validacijos duomenų rinkinio nuostolis RoBERTA atveju pradeda kilti kiek drąstiškiau, nei BERT. Tai sufleruoja, jog šiuo atveju RoBERTa modelis persimoko greičiau, nei BERT.

Nepaisant to, jog validacijos nuostolis bandymo su RoBERTa metu buvo didesnis, nei BERT, validacijos duomenų aibės F2 reikšmių grafikas rodo, jog RoBERTa modelio F2 reikšmės greitai paveja BERT reikšmes bei gan smarkiai jas aplenkia. Po 500 iteracijų, BERT pasiekia F2 reikšmę 87 %, o RoBERTa – 90 % Tad šio eksperimento rezultatai veda link išvados, jog apmokius modelius su tiek pat duomenų, architektūros, kurios naudoja RoBERTa, pasiekia kiek didesnį tikslumą, nei architektūros, naudojančios BERT.

Esminiai eksperimento rezultatai pateikti 11 lentelėje.

11 lentelė. RoBERTa ir BERT apmokymo palyginimo rezultatai

Modelis	Apmokyto modelio dydis	Apmokymo trukmė (hh:mm:ss)	Galutinė F2 reikšmė apmokius	Galutinė validacijos nuostolio reikšmė
RoBERTa	498,7 MB	0:07:41	0.9	0.7
BERT	438,0 MB	0:06:47	0.87	0.53

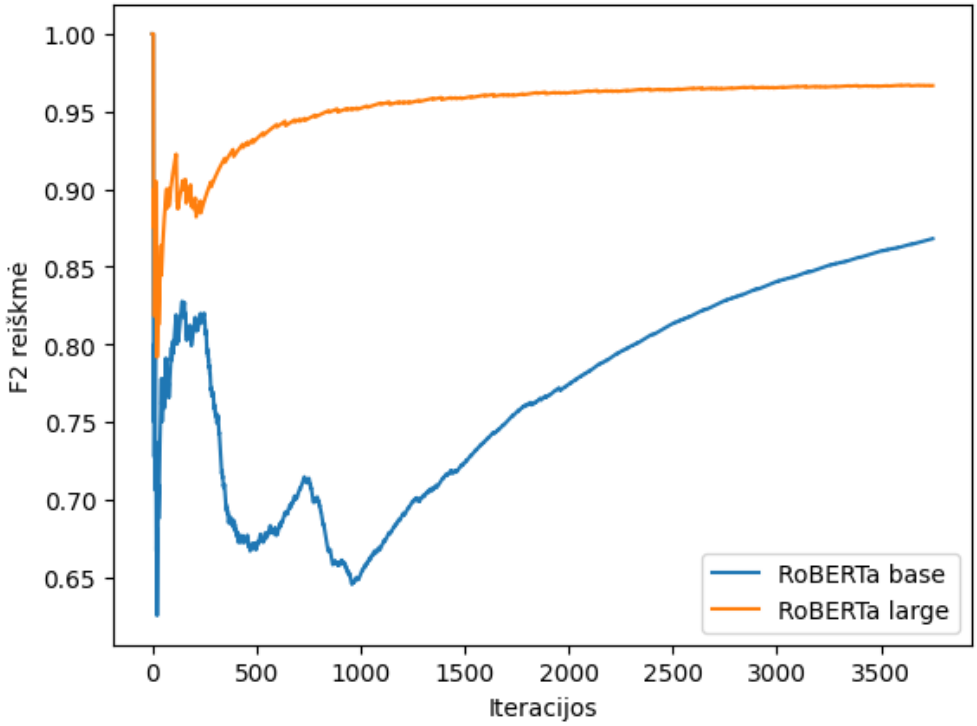
Taigi, remiantis tuo, jog RoBERTa pasiekia geresnius tikslumo rezultatus, toliau šiame darbe kandidatų tikrinimo architektūroje vietoje BERT yra naudojama RoBERTa.

3.3.5.6. Didesnis RoBERTa modelis

Didelis RoBERTa modelis, arba – RoBERTa-large [11], yra didesnė RoBERTa modelio versija. Pagrindinis modelių skirtumas – didelis RoBERTa modelis turi daugiau parametrų ir yra ilgiau apmokytas. Modelio išvesties įterpinių dimensija yra ne 768, o 1024 – vadinasi, reikalingas ir skirtingas klasifikavimo sluoksnis, kurio įeities dimensija yra 1024.

Kaip ir kitų bandymų metu, tikrintas apmokymo bei validacijos nuostolių funkcijos reikšmės pokytis bei pasiekiama F2 funkcijos reikšmė. Bandymai su abejais modeliais turėjo identiškus parametrus.

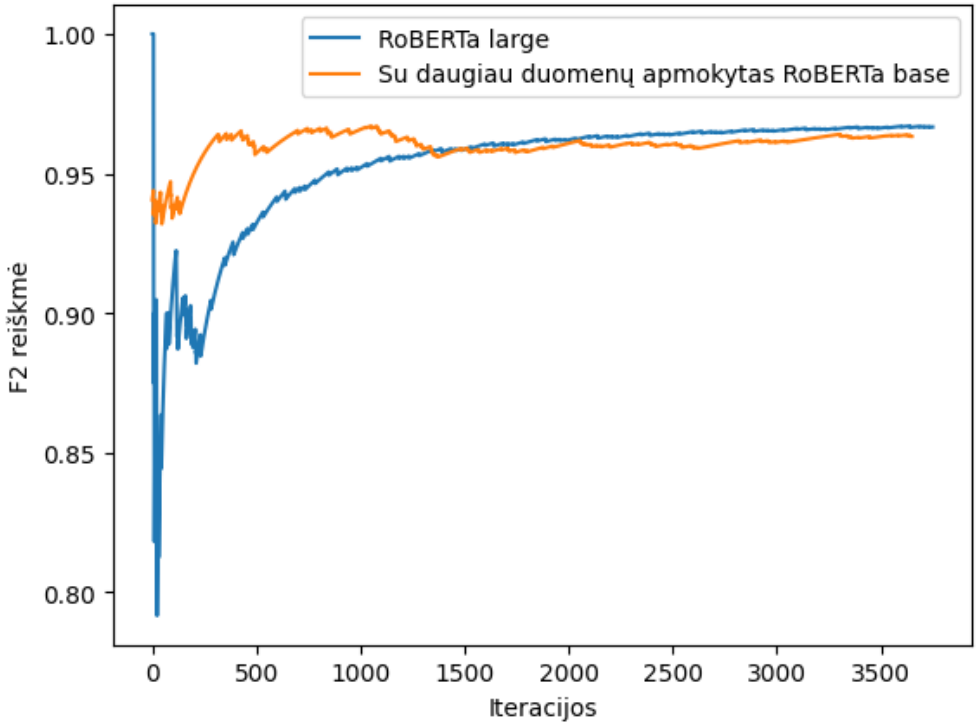
33 pav. pateikiami F2 funkcijos rezultatai.



33 pav. RoBERTa ir RoBERTa large F2 reikšmių palyginimas

Naudojantis dideliu RoBERTa modeliu žymiai greičiau pasiekiama didelė F2 reikšmė. Tačiau kyla klausimas, ar paprastas modelis galiausiai nebūtų pasiekęs tos pačios reikšmės? Šiame darbe atsakyti į šį klausimą neužteko laiko, tačiau toliau pateikiamas F2 grafikas, lyginantis tą patį RoBERTa large modelį su modeliu, kuris buvo apmokytas ilgesnį laiką su keletą kartų daugiau duomenų.

[11] Pasiekiamas https://huggingface.co/roberta-large



34 pav. Su daugiau duomenų apmokyto RoBERTa modelio ir RoBERTa large F2 reikšmių palyginimas

Iš 34 grafiko matyti, jog net ir su daugiau duomenų apmokytas RoBERTa modelio tikslumas pralenkia didelio RoBERTa modelio tikslumą tik pradžioje, tačiau apmokius su daugiau duomenų ši persvara dingsta. Nepaisant to, šie rezultatai rodo, jog RoBERTa-large modelis turi pranašumą net ir neapmokius su daug duomenų.

Svarbu paminėti, jog didelis RoBERTa modelis užima daugiau atminties, nei paprastas modelis. Apmokant modelį su vaizdo plokštę su 8 GB atminties kilo daug problemų – ne kartą atmintis buvo viršyta. Taip pat apmokant didelį modelį apmokymo trukmė padidėjo daugiau nei du kartus. Dėl šių priežasčių darbe šis modelis nėra naudojamas, tačiau turint tinkamą įrangą (vaizdo plokštės atmintis turėtų būti bent 12 GB arba daugiau), siūloma naudoti didelį modelį.

Esminiai eksperimento rezultatai pateikti 12 lentelėje.

12 lentelė. RoBERTa-base ir RoBERTa-large apmokymo palyginimo rezultatai

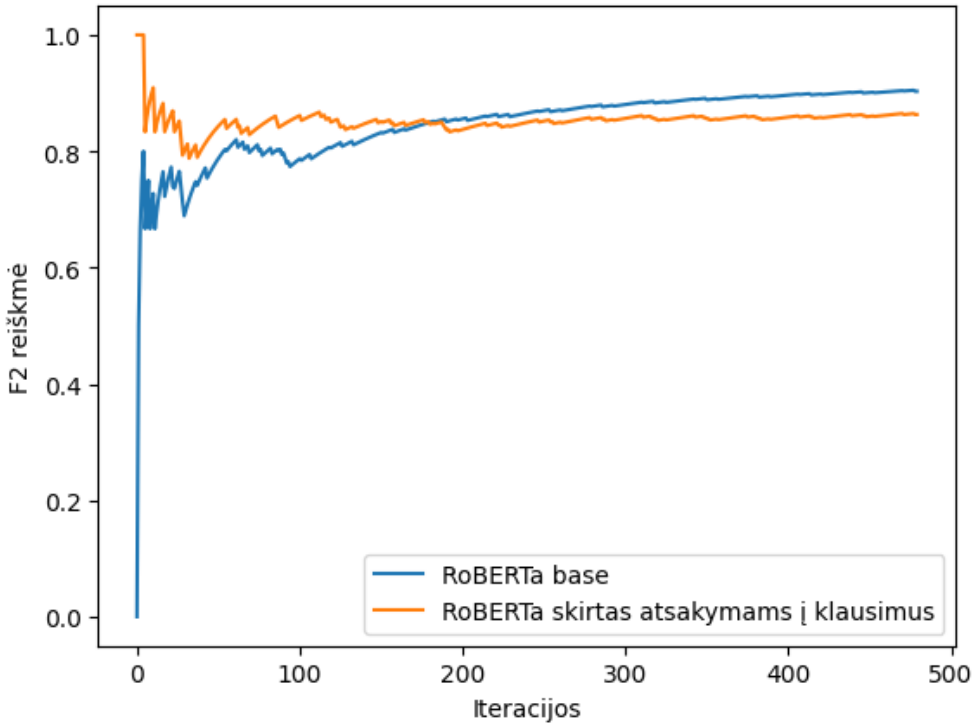
Modelis	Apmokyto modelio dydis	Apmokymo trukmė (hh:mm:ss)	Galutinė F2 reikšmė apmokius
RoBERTa-base	498,7 MB	1:06:41	0.86
RoBERTa-large	1,4 GB	2:38:59	0.96

3.3.5.7. Modelis, iš anksto apmokytas atsakyti į klausimus

Dialogo būsenos radimo uždavinys yra gan panašus į klausimo-atsakymo uždavinį, šiuo atveju klausimas yra „kokia dialogo būsena?“. Nors ir šiame darbe modelių architektūros nebuvo kuria-

mos spręsti klausimo-atsakymo uždaviniui, tačiau verta patikrinti, kaip žinių pernešimas veikia ant modelių, kurie yra papildomai apmokyti atsakyti į klausimus, palyginus su standartiniais modeliais (BERT ir pan.).

Buvo atlikti bandymai su standartiniu *roberta-base* modeliu bei *roberta-base-squad2*, kuris yra tas pats *roberta-base* modelis papildomai apmokytas ant duomenų, kuriuose yra klausimai bei atsakymai. 35 pav. pateikiamas F2 reikšmių grafikas.



35 pav. RoBERTa ir RoBERTa, apmokyto atsakyti į klausimus, F2 reikšmių palyginimas

Iš šio grafiko aiškiai matosi, jog nors ir iš pradžių klausimams skirto RoBERTa modelio tikslumas viršija paprasto RoBERTa modelio tikslumą, RoBERTa modeliui neilgai trunka pastarąjį pasivyti ir aplenkti. Iš to daroma išvada, jog klausimų ir atsakymų modelis nėra tinkamas 1.4.1 aprašytai architektūrai.

3.3.5.8. Kiti parametrai

Visus modelius apmokant buvo naudojama Adam optimizatorius su mokymo greitumo (angl. *learning rate*) parametru $2e-5$. Ši parametro reikšmė tinka daugumai modelių, tačiau pastebėta, jog apmokant RoBERTa-large modelį, apmokymo nuostolių funkcija artėja prie 0 greičiau, jei šis parametras yra $1e-4$. Tai gali būti dėl to, jog modelis turi daugiau parametrų, tad su mažesniu mokymo greičio parametru didesnė tikimybė, jog nuostolių funkcijos reikšmė užstrigs lokaliame minimume. Tuo tarpu parametro reikšmė $2e-3$ pasirodė per didelė – apmokymo nuostolių funkcijos reikšmė mažėjo lėčiau, nei praeitame bandyme.

Apmokant visus modelius buvo naudojama kiek galima didesnė duomenų rinkinio dydžio (angl. *batch*) parametro reikšmė. Dažnai tai būdavo 16 – dydis, apribotas turimos vaizdo plokštės atminties, tačiau RoBERTa-large modelį buvo galima apmokyti tik nustačius parametro reikšę

4. Skirtingi dydžiai turėjo skirtingą įtaką apmokymo greičiui, tačiau rezultatų tikslumo skirtumų nepastebėta.

Taip pat buvo pastebėta, jog naudojant gradientų akumuliavimą, aprašytą 1.2.3, pasiekiamas kelis kartus greitesnis apmokymas. Apmokant RoBERTa-large modelį, rinkinio *(batch)* dydis buvo nustatytas 4. Be gradiento akumuliavimo, modelis apdorojo tik 4 iteracijas per sekundę, tuo tarpu nustačius gradiento akumuliavimo parametro reikšmę lygią 8, kas efektyviai ekvivalentu rinkinio dydžiui $4 * 8 = 32$, apmokymo greičio vidurkis tapo apie 12 iteracijų per sekundę.

3.3.6. GPT 3.5

Atliktas eksperimentas, siekiant išbandyti, kaip gerai GPT 3.5 – modelis, kuriuo paremtas gerai žinomas ChatGPT – išsprendžia dialogo būsenos sekimo problemą. Eksperimentas atliktas naudojantis OpenAI API, leidžiantį siųsti žinutes GPT 3.5 ir gauti rezultatus iš kodo.

Kiekvienas API kvietimas susideda iš visos reikalingos informacijos dialogo būsenai rasti JSON formatu, t.y., sistemos bei naudotojo pasisakymai bei tuo metu žinoma dialogo būsena. Tikimasi, jog GPT 3.5 grąžins atsakymą JSON formatu su 1) naujais dialogo būsenos kintamaisiais 2) atnaujinta dialogo būsena.

Nurodymas, kokią užduotį ir kokiu formatu grąžinti atsakymą GPT 3.5 nusiunčiamas pradžių sukuriama sistemos instrukcija. Ši instrukcija – vieną kartą išsiunčiama žinutė, kurioje glaustai paaiškinama, kas yra dialogo būsenos sekimas, kokios įvesties tikėtis, kokiu formatu turi būti pateikta išvestis bei užklausos bei tikimosi atsakymo pavyzdys.

API iškvietimų skaičius bei turinio ilgis yra ribojimas, tad vienu API iškvietimu gali būti išsiunčiami keli (apie 10) dialogų, o iškvietimų galima išsiųsti apie 3-10 per minutę (priklausomai nuo to, kiek užtrunka užklausos apdorojimas bei koks API planas yra naudojamas).

Gauti rezultatai apvalomi ir sudedami į Pandas DataFrame, kaip matyti 36 paveiksliuke.

	row_id	system_transcript	user_transcript	turn_label_target	turn_label_gpt	belief_state_target	belief_state_gpt
0	0	NaN	What is the phone number and postcode of a che...	[{'area': 'east'}, {'price range': 'cheap'}, {...	[{'request': 'phone'}, {'request': 'postcode'}...	[{'slot': 'phone', 'act': 'request'}, {'slot':...	[{'slot': 'phone', 'act': 'request'}, {'slot':...
1	1	There is The Missing Sock. They serve internat...	Thanks so much !	[]	[]	[{'price range': 'cheap', 'act': 'inform'}, {'...	[{'phone': '01223 812660', 'postcode': 'C.B 25...
2	2	Is there anything else I can help you with tod...	No , thank you .	[]	[]	[{'price range': 'cheap', 'act': 'inform'}, {'...	[{'phone': '01223 812660', 'postcode': 'C.B 25...
3	3	NaN	hello , i 'm looking for a restaurant with fai...	[{'price range': 'moderate'}]	[{'price range': 'fair', 'act': 'inform'}]	[{'price range': 'moderate', 'act': 'inform'}]	[{'price range': 'fair', 'act': 'inform'}]
4	4	There are 31 places with moderate price range....	well I want to eat in the North , what 's up t...	[{'area': 'north'}]	[{'area': 'north'}]	[{'price range': 'moderate', 'act': 'inform'},...	[{'price range': 'moderate', 'act': 'inform'},...
...	...	...	...	...	...	...	...
1641	1641	Nandos is in the cheap price range.	Excellent . What is their address and phone nu...	[{'request': 'phone'}, {'request': 'address'}]	[{'slot': 'address', 'act': 'request'}, {'slot...	[{'slot': 'phone', 'act': 'request'}, {'slot':...	[{'price range': 'cheap', 'act': 'inform'}, {'...
1642	1642	The phone number and address are 01223 327908 ...	Thank you , goodbye !	[]	[]	[{'price range': 'cheap', 'act': 'inform'}, {'...	[]
1643	1643	NaN	I want to find a restaurant in the centre and ...	[{'area': 'center'}, {'food': 'japanese'}]	[{'slot': 'price range', 'act': 'request'}, {'...	[{'food': 'japanese', 'act': 'inform'}, {'area...	[{'price range': 'cheap', 'act': 'inform'}, {'...
1644	1644	There's a very nice Japanese restaurant in the...	That 's OK because it 's for a special occasio...	[{'request': 'phone'}]	[{'slot': 'phone', 'act': 'request'}]	[{'slot': 'phone', 'act': 'request'}, {'food':...	[{'food': 'japanese', 'act': 'inform'}, {'area...
1645	1645	Yes, It's 01223462354. What else can I help ...	No thank you , goodbye	[]	[]	[{'food': 'japanese', 'act': 'inform'}, {'area...	[]

36 pav. Apdorotų ChatGPT atsakymų lentelė

Tikslumas. Pagal žingsnio užklausos tikslumą *turn request accuracy*, aprašytą 1.1.4, GPT 3.5 pasiekia apie 29,1 %. Tai reiškia, jog apie ketvirtadalį visų įrašų, GPT 3.5 atspėta dialogo būsena sutampa su tikrąja. Pagal jungtinio siekio tikslumą (Joint Goal Accuracy), aprašytą 1.1.4, GPT 3.5 pasiekia apie 23,57 %., kas taip pat reiškia, jog GPT 3.5 sugeba būseną atnaujinti vystantis dialogui ir tai atlieka sėkmingai apie ketvirtadalį dialogų. Nors rezultatai atrodo mažokas, visgi reiktų turėti omenyje, jog jungtinio siekio tikslumo metrika yra itin griežta, kadangi tikroji ir atspėtosios būsenos aibės turi visiškai sutapti – tad net jei ir būsena susideda iš 10 kintamųjų ir jų reikšmių, viena ne tokia raidė ar kokia nors papildoma reikšmė, nesanti tikrojoje aibėje, yra traktuojama kaip neatspėta būsena.

Jei sušvelnintume tikslumo formulę, leisdami suklysti, ir kiekvienam ėjimui skaičiuodami, koks santykis yra tarp atspėtų elementų aibėje lyginti su elementų tikrojoje aibėje, gauname geresnį rezultatą – 53,8 % Tai reiškia, jog GPT 3.5 atspėja apie pusę tikrų būsenos kintamųjų ir jų reikšmių kiekvieną ėjimą.

Lyginant su [LTB+20] bei [CL19], kuriuose pasiekiamas virš 90 % jungtinio siekio tikslumas, GPT 3.5 rezultatai yra prasti, tačiau verta pabrėžti, jog GPT 3.5 nereikalauja ontologijos tad gali būti laikomas kaip daug lankstesnis, tačiau mažiau tikslus, būdas sekti būseną.

3.3.6.1. Klaidų analizė

Nėra garantijos, jog GPT 3.5 atsakydamas laikysis nurodytos struktūros. Tad daug klaidų kyla iš to, jog pateikti atsakymai buvo neapdorojami (apie 1/10). Dauguma jų turėjo nevalidžią JSON struktūrą dėl įvairių sintaksės klaidų. Pavyzdžiui, GPT 3.5 turėjo tendenciją grąžinti būlio reikšmes False/True be kabučių (kas įprasta Python programavimo kalboje, kuri leidžia interpretuoti JSON objektus), kai turėtų būti kabutėse „false“/„true“.

Kitos klaidos kilo iš to, jog GPT 3.5 nebuvo nurodyta ontologija, tad ne visais atvejais buvo žinoma, kokia reikšmė yra leistina. Pavyzdžiui, jei tikimasi, kad į užklausą „price range“ atsakyta „moderate“, GPT 3.5, to nežinodamas, priskiria „fair“, kas jau yra neatspėta būsena tikrinant su WOZ 2.0 duomenų rinkinu.

Rezultatai ir išvados

Rezultatai: Šiame magistro baigiamajame darbe buvo išnagrinėti ir palyginti dialogo būsenos sekimo sprendimo būdai naudojantis dvi pagrindines transformerių architektūras – BERT ir GPT. Aprašyti šių nagrinėtų sprendimų trūkumai ir privalumai, vienas su kitu palyginami atliekant įvairius bandymus bei bandyta atkartoti literatūroje užfiksuoti rezultatai.

Pasiūlyta hibridinė architektūra, gebanti išspręsti identifikuotus BERT architektūromis paremtų sprendimų lankstumo trūkumą, identifikuojant rakinius dialogo būsenos žodžius. Nagrinėti darbai informaciją išgauna arba iš teksto, arba iš apibrėžtos ontologijos, siūloma architektūra sukombinuoja abu sprendimus. Šios architektūros pranašumas buvo patikrintas bandymo metu, kurio metu bandyta patikrinti modelio spėjimų tikslumą. Gauti rezultatai parodė, jog tam tikrais atvejais architektūra neatsilieka ar net lenkia kitus darbe aprašytus sprendimus (tam tikrais atvejais pasiekiamas iki 1.4 % didesnis tikslumas), tačiau išsamesniam palyginimui reikalingi turtingesni duomenų rinkiniai, pvz., tokie, kurių ontologijos nėra pilnai apibrėžtos.

Taip pasiūlyti būsenos radimo tekste naudojantis BERT metodų patobulinimai. Pirmasis patobulinimas – vietoje keleto skirtingų klasifikavimo sluoksnių kiekvienam ontologijos kintamajam (angl. *slot*), kaip aprašyta [CL19], informaciją apie kintamąjį *slot* laikyti įeities tekste, taip sumažinant modelio parametrų skaičių. Antrasis pasiūlymas – siekiant išvengti spėjimų, kai modelis būsenos ieško klaidinguose įeities teksto segmentuose, pasinaudoti nenorimų segmentų maskavimu bei taikyti klaidingus spėjimus baudžiančius svorių parametrus nuostolių funkcijoje. Šių patobulinimų nauda pagrįsta bandymais, kurių metu parodoma, jog supaprastinus architektūrą galima pasiekti apie 2.5 % didesnį būsenos radimo tikslumą. Tuo tarpu naudojant maskavimą, pasiekiama apie 10 % didesnė F2 reikšmė apmokant modelį.

Tam, kad apmokyti būsenos radimo tekste modelius, sukurtas būdas praplėsti įvairius duomenis, išsaugant informaciją apie tai, kur tekste slypi žodžiai, minintys būseną. Ruošiant duomenis naudojamas apytikslės atikties (angl. *fuzzy match*) algoritmas, leidžiantis rasti žodžius, atitinkančius ieškomą būseną. Šis būdas gali būti naudojamas išplėsti duomenų rinkinius būsenos radimo tekste užduočiai.

Negana to, atlikti įvairūs smulkesni bandymai, kurių tikslas yra pagrįsti tam tikras naudojamas metodikas apmokant modelius, sprendžiančius dialogo būsenos radimo problemą bei rasti optimalius parametrus siekiant išgauti kuo geresnę F2 reikšmę tikrinant modelių spėjimus.

Galiausiai atliktas bandymas su GPT 3.5 – modeliu, kuriuo paremtas ChatGPT – siekiant patikrinti, ar GPT modeliai tinkami spręsti dialogo būsenos radimo užduotį bei kaip jų pasiekiami rezultatai lyginasi su BERT šakos modeliais. Rezultatai parodė, jog GPT 3.5 pavyksta pasiekti apie 29,1 % žingsnio užklausos tikslumą, kas neprilygsta geriausiems rezultatams, publikuotiems mokslinėje literatūroje. Tačiau skirtingai nei kiti modeliai, GPT nereikalauja ontologijos, tad atsiranda lankstesnis būdas spręsti DST problemą be ontologijos; taip pat dėl to daroma daugiau klaidų, kadangi ontologija apibrėžia galimų žodžių aibę.

Naudojant RoBERTa modelį kandidatų generavimui, atsitiktinį praretinimą (angl. *dropout*) bei perteklinį perrinkimą (angl. *oversampling*), duomenų rinkinius DSTC2 bei WOZ 2.0, toliau pateiktoje lentelėje pateikti gautų galutinių rezultatų palyginimas (naudojamos metrikos aprašytos

1.1.4 poskyryje). Apmokymui su WOZ duomenų rinkiniu skirta 15 epizodų, tuo tarpu DSTC – 5. Galutiniai rezultatai pateikiami 13 lentelėje.

13 lentelė. Galutiniai šiame darbe apmokytų modelių, naudojantis darbe aprašytus metodus, rezultatai

Modelis	**Žingsnio užklausos tikslumas**	**Duomenų rinkinys**
Kandidatų generavimo	96.9 %	DSTC2
Būsenos radimo tekste	86.9 %	DSTC2
GPT 3.5	29,1 %	DSTC2
Kandidatų generavimo	90.4 %	WOZ 2.0
Būsenos radimo tekste	81.7 %	WOZ 2.0
GPT 3.5	30,7 %	WOZ 2.0

Deja, šio darbo metu nepavyko pagerinti literatūroje egzistuojančių rezultatų. Pirmoji priežastis – ruošiant DSTC duomenis būsenos radimo tekste modeliui, kiti sprendimai, pvz., [CL19] naudoja įvairius žodžių suvienodinimo metodus, kadangi būsena, randama tekste, gali nesutapti su būsena, apibrėžta ontologijos. Šiame darbe tai nebuvo daroma, kadangi norėta sukurti lanksčią architektūrą, nedarant prielaidų apie ontologijos reikšmes. Antroji priežastis – buvo siekiama nepermokyti šiame darbe įgyvendintų modelių (atsižvelgiant į validacijos nuostolių funkcijos reikšmes) dėl laiko stokos bei tam, jog rezultatai kuo labiau atspindėtų realybę. Dėl šių priežasčių modeliai buvo apmokomi su daugiausia 15 epizodų, nors literatūros šaltiniai rodo, jog geriausiems rezultatams pasiekti gali prireikti 100 epizodų.

Išvados:

1. Darbe pasiūlytas būsenos radimo tekste ir kandidatų generavimo metodų apjungimas (darbe vadinamas hibridinė architektūra) gali suteikti pranašesnius rezultatus (pagal žingsnio užklausos tikslumo metriką), nei kiekvienas iš minėtų metodų atskirai;

2. Darbe parodyta, jog būsenos radimo tekste modelių, kai naudojami skirtingi klasifikavimo sluoksniai skirtingiems ontologijos kintamiesiems, tikslumą (pagal žingsnio užklausos tikslumo metriką) galima pagerinti bei architektūrą supaprastinti reikalingą informaciją apie ontologiją užkoduojant į įeitį;

3. Darbe parodyta, jog būsenos radimo tekste modelių, kai naudojami skirtingi klasifikavimo sluoksniai skirtingiems ontologijos kintamiesiems, tikslumą (pagal žingsnio užklausos tikslumo metriką) begalima pagerinti naudojant skirtingų segmentų maskavimą;

4. GPT architektūros taip pat gali būti tinkamos spręsti dialogo būsenos radimo uždavinį, kaip parodė eksperimentai su ChatGPT, tačiau kadangi GPT yra tekstą generuojantis modelis, jo išvestis yra neprognozuojama ir gaunami rezultatai yra prastesni (pagal žingsnio užklausos tikslumo metriką) nei BERT modeliais gauti rezultatai; taip nėra daug literatūros detaliai aprašančios, kaip tai įgyvendinti;

5. Bandymų metu parodyta, jog vykdant dialogo būsenos radimo užduotį su BERT:

- Tiesinio klasifikavimo sluoksnio sudėtingumas (papildomi vidiniai sluoksniai) neteikia jokios naudos modelio apmokymui bei spėjimų tikslumui;
- Atsitiktinio praretinimo transformacija (angl. *dropout*) apmokant modelį leidžia pasiekti didesnes tikslumo reikšmes greičiau, nei be jo;
- Perdėtas atsitiktinis atrinkimas (angl. *random oversampling*) gali padėti pasiekti geresnius tikslumo rezultatus;
- Žinių pernešimas (angl. *knowledge transfer*) tarp kandidatų tikrinimo ir būsenos radimo tekste nepadėjo pasiekti geresnių rezultatų, nei be žinių pernešimo;
- RoBERTa modelis gali padėti pasiekti aukštesnes spėjimų tikslumo reikšmes, nei BERT;
- Didelis RoBERTa modelis (RoBERTa-large) leidžia pasiekti dar didesnį tikslumą, nei RoBERTa, to kaina – beveik dvigubai didesnis vaizdo plokštės atminties naudojimas bei apmokymo trukmės pailgėjimas daugiau, nei dvigubai;
- Nepaisant problemų panašumo, iš anksto apmokyti atsakyti į klausimus (angl. *question answering*) modeliai nepadėjo pasiekti geresnių rezultatų sprendžiant būsenos ieškojimo tekste užduotį;

Tolimesni darbai:

1. Norint teisingai palyginti įvairias metodikas bei modelius tarpusavyje ir geriau pagrįsti gautus rezultatus, modelius reikėtų apmokyti ilgiau, nei buvo šiame darbe;
2. Siūloma būsenos radimo tekste modelius apmokyti su SIM-R ir SIM-M duomenų rinkiniu, kadangi duomenys yra pritaikyti būtent šiai problemai spręsti. Taip būtų galima pasiekti dar tikslesnius rezultatus;
3. Tam, kad būtų galima įsitikinti siūlomos hibridinės architektūros nauda, siūloma ją praktiškai įgyvendinti ir palyginti tokius kriterijus, kaip modelio dydis, apmokymo laikas, spėjimų laikas. Ši informacija būtų naudinga siekiant suprasti, ar toks sprendimas yra naudotinas produkcijoje;
4. Taip pat siūloma surinkti ar paruošti įvairesnius dialogo duomenis, siekiant geriau patikrinti minėtus sprendimus. WOZ 2.0 bei DSTC 2 duomenų rinkiniuose visos būsenos turi apibrėžtas, leidžiamas reikšmes, tačiau tai neatitinka realybės – tikruose dialoguose visų įmanomų reikšmių iš anksto apibrėžti neįmanoma arba labai nepraktiška (pvz., dialogai, kuriuose minimi įvairūs adresai, telefono numeriai, skirtingos pokalbių temos);
5. Išbandyti daugiau giliojo mokymosi metodų. Pavyzdžiui, šiame darbe nespėta pasinaudoti adaptyviu mokymusi (angl. *adaptive learning*), kuris galėtų padėti efektyviau apmokyti modelius bei pasiekti geresnius rezultatus.

Literatūra

[BCB14a] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. https://arxiv.org/pdf/1409.0473.pdf, 2014.

[BCB14b] D. Bahdanau, K. Cho ir Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BMR+20] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, and A. Neelakantan. Language models are few-shot learners. In pp. 1877–1901, 2020.

[BWT+18] P. Budzianowski, T. Wen, B. Tseng, I. Casanueva, S. Ultes, O. Ramadan ir M. Gašić. MultiWOZ–a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. *arXiv preprint arXiv:1810.00278*, 2018.

[CL19] G. Chao ir I. Lane. Bert-dst: Scalable end-to-end dialogue state tracking with bidirectional encoder representations from transformer. *arXiv preprint arXiv:1907.03040*, 2019.

[CMG+14] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. https://arxiv.org/pdf/1406.1078.pdf, 2014.

[DCL+18] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. https://arxiv.org/pdf/1810.04805.pdf, 2018.

[FWL20] Y. Feng, Y. Wang ir H. Li. A Sequence-to-Sequence Approach to Dialogue State Tracking. *arXiv preprint arXiv:2011.09553*, 2020.

[GJM13] A. Graves, A. Jaitly, and A. R. Mohamed. Hybrid speech recognition with deep bidirectional LSTM. In pp. 273–278, 2013.

[GSB+20] M. Geva, R. Schuster, J. Berant, and O. Levy. Transformer feed-forward layers are key-value memories. https://arxiv.org/pdf/2012.14913.pdf, 2020.

[HKG+15] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. In 2015.

[HP99] E. Horvitz and T. Paek. A Computational Architecture for Conversation. In pp. 201–210, 1999.

[HS97] S. Hochreiter and J. Schmidhuber. Long short-term memory. In pp. 1735–1780, 1997.

[HTY14] Matthew Henderson, Blaise Thomson, and Steve Young. Word-based dialog state tracking with recurrent neural networks. In pp. 292–299, 2014.

[HTW14a] M. Henderson, B. Thomson, and J. Williams. The Second Dialog State Tracking Challenge. *SIGdial*, 2014.

[HTW14b] M. Henderson, B. Thomson ir J. D. Williams. The third dialog state tracking challenge. *2014 IEEE Spoken Language Technology Workshop (SLT)*, p.p. 324–329. IEEE, 2014.

[HTW4b] M. Henderson, B. Thomson, and J. Williams. The Third Dialog State Tracking Challenge. *IEEE SLT*, 2014b.

[YLQ21] Y. Yang, Y. Li ir X. Quan. Ubar: Towards fully end-to-end task-oriented dialog system with gpt-2. *Proceedings of the AAAI Conference on Artificial Intelligence*, tom. 35 numeris 16, p.p. 14230–14238, 2021.

[YZP+20] T. Yu, R. Zhang, A. Polozov, C. Meek ir A. H. Awadallah. Score: Pre-training for context representation in conversational semantic parsing. *International Conference on Learning Representations*, 2020.

[Kel84] J. Kelley. An iterative design methodology for user-friendly natural language office information applications. *ACM Transactions on Information Systems (TOIS)*, 2(1):26–41, 1984.

[LCO21] C. Lee, H. Cheng ir M. Ostendorf. Dialogue state tracking with a language model using schema-driven prompting. *arXiv preprint arXiv:2109.07506*, 2021.

[LTB+20] T. M. Lai, Q. H. Tran, T. Bui, and D. Kihara. A simple but effective bert model for dialog state tracking on resource-limited systems. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8034–8038. IEEE, 2020.

[LTB21] W. Lin, B. Tseng ir B. Byrne. Knowledge-aware graph-enhanced gpt-2 for dialogue state tracking. *arXiv preprint arXiv:2104.04466*, 2021.

[McT21] M. McTear. *Conversational AI: Dialogue Systems, Conversational Agents, and Chatbots*. Morgan & Claypool Publishers, 2021. 253 p.

[MST+15] N. Mrkšić, D. Séaghdha, B. Thomson, M. Gašić, P. Su, D. Vandyke, T. Wen, and S. Young. Multi-domain dialog state tracking using recurrent neural networks. `https://arxiv.org/pdf/1506.07190.pdf`, 2015.

[Ola15] C. Olah. Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs, 2015.

[Ope22] OpenAI. Chatgpt: optimizing language models for dialogue. `https://openai.com/blog/chatgpt/`, 2022.

[RNS+18] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. https://www.cs.ubc.ca/ amuham01/LING530/papers/radford2018improving.pdf, 2018.

[RWC+19] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. OpenAI blog 1, no. 8, 2019.

[She20] S. Shead. Why everyone is talking about the A.I. text generator released by an Elon Musk-backed lab. https://www.cnbc.com/2020/07/23/openai-gpt3-explainer.html, 2020.

[SMH11] I. Sutskever, J. Martens, and G. Hinton. Generating text with recurrent neural networks. *ICML*, 2011.

[SVL14] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In p. 27, 2014.

[VFJ15] O. Vinyals, M. Fortunato ir N. Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

[VSP+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In 2017.

[VTM+19] E. Voita, D. Talbot, F. Moiseev, R. Senrich, and I. Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. https://arxiv.org/pdf/1905.09418.pdf, 2019.

[WY07] J. D. Williams and S. Young. Partially observable Markov decision processes for spoken dialog systems. In pp. 393–422, 2007.

[WRR+13] J. Williams, A. Raux, D. Ramachadran, and A. Black. The Dialog State Tracking Challenge. *SIGdial*, 2013.

[WVM+16] T. Wen, D. Vandyke, N. Mrksic, M. Gasic, L. Rojas-Barahona, P. Su, S. Ultes ir S. Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016.

[XH18] P. Xu and Q. Hu. An End-to-end Approach for Handling Unknown Slot Values in Dialogue State Tracking. `https://arxiv.org/pdf/1805.01555.pdf`, 2018.

Sąvokų apibrėžimai

- Dialogo būsena – būsenos kintamųjų ir reikšmių porų aibė, kurioje laikoma pašnekovo išsakyta informacija.
- Būsenos kintamasis (angl. *slot*) – konkreti išsakytos informacijos tema, pavyzdžiui, kaina.
- Būsenos kintamojo reikšmė (angl. *value*) – reikšmė, apibūdinanti kintamąjį. Pavyzdžiui, kaina (kintamasis) gali būti nedidelė (reikšmė).
- Jungtinis siekio tikslumas (angl. *joint goal accuracy*) – viena iš pagrindinių metrikų, naudojamų įvertinti dialogo būsenos radimo modelių tikslumą.

Santrumpos

- DST – dialogo būsenos sekimas, angl. *dialog state tracking*.
- NLP – natūralios kalbos apdorojimas, angl. *natural language processing*.