FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Doporučovací modely založené na rekurentních neuronových sítích
Student:	Bc. Ladislav Martínek
Vedoucí:	Ing. Tomáš Řehořek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Seznamte se s problematikou
1) předzpracování interakčních dat v doporučovacích systémech,
2) maticové faktorizace pro datasety s implicitní zpětnou vazbou pro generování reprezentace položek (embeddings),
3) možnostmi použití rekurentních neuronových sítí v těchto systémech.

Navrhněte a implementujte algoritmus pro tvorbu modelů, které generují doporučení na základě sekvenčních dat z chování uživatelů pomocí rekurentních hlubokých neuronových sítí (např. LSTM) a faktorizačních embeddingů.

Vyhodnoťte úspěšnost navržených modelů a jejich různých hyperparametrizací na 2 dodaných datových sadách pomocí offline metrik recall a catalog coverage, případně navrhněte modifikace těchto metrik berouc v potaz sekvenční povahu dat.

Dosažené výsledky vhodným způsobem prezentujte, porovnejte s tradičními metodami (např. kolaborativním filtrováním) a diskutujte další možné směry zlepšení modelů a jeho hyperparametrizací.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Karel Klouda, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 9. ledna 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Doporučovací modely založené na rekurentních neuronových sítích

Bc. Ladislav Martínek

Katedra aplikované matematiky
Vedoucí práce: Ing. Tomáš Řehořek, Ph.D.

27. května 2020

Poděkování

Nejdříve bych rád poděkoval svému vedoucímu, Ing. Tomáši Řehořkovi, Ph.D., za cenné rady a věnovaný čas při zpracování mé diplomové práce. Dále bych rád poděkoval firmě Recombee, s.r.o. za poskytnutý hardware pro testování, díky němuž bylo možné provádět velké množství experimentů. V neposlední řadě děkuji mé rodině a přítelkyni za podporu a pochopení, jak při tvorbě této práce, tak i během celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 27. května 2020 . . . . . .

Odkaz na tuto práci

Martínek, Ladislav. *Doporučovací modely založené na rekurentních neuronových sítích.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato diplomová práce řeší problematiku doporučovacích systémů. Cílem je predikce následujících položek na základě sekvenčních dat z chování uživatelů pomocí rekurentních neuronových sítí (LSTM, GRU). Reprezentace položek je tvořena pomocí maticové faktorizace upravené pro datasety s implicitní zpětnou vazbou. V práci je navržen a implementován algoritmus pro tvorbu rekurentních modelů využívající vytvořenou reprezentaci položek. Navržen je také způsob vyhodnocování respektující sekvenční povahu dat. Metoda vyhodnocování využívá metriky recall a catalog coverage. Experimenty jsou prováděny systematicky s cílem zjistit závislosti na sledovaných metodách a hyperparametrech. Měření je prováděno na třech datových sadách. Na největším datasetu se podařilo dosáhnout více jak dvojnásobného recallu proti dalším metodám, které byly zastoupeny kolaborativním filtrováním, reminder modelem a popularity modelem. Na závěr práce jsou diskutovány zjištěné poznatky, možné zlepšení hyperparametrizací a další možné směry vylepšení modelů.

Klíčová slova Doporučovací systémy, Datasety s implicitní zpětnou vazbou, Sekvenční doporučování, Strojové učení, Rekurentní neuronové sítě, LSTM, GRU, Maticová faktorizace

Abstract

This diploma thesis deals with matters of recommendation systems. The aim is to use recurrent neural networks (LSTM, GRU) to predict the subsequent interactions using sequential data from user behavior. Matrix factorization adapted for datasets with implicit feedback is used to create a representation of items (embeddings). An algorithm for creating recurrent models using the embeddings is designed and implemented in this thesis. Furthermore, an evaluation method respecting the sequential nature of the data is proposed. This evaluation method uses recall and catalog coverage metrics. Experiments are performed systematically to determine the dependencies on the observed methods and hyperparameters. The measurements were performed on three datasets. On the most extensive dataset, I managed to achieve more than double recall against other recommendation techniques, which were represented by collaborative filtering, reminder model, and popularity model. The findings, possible improvement by hyper-parametrization, and different possible means of model improvement are discussed at the end of the work.

Keywords Recommender systems, Implicit feedback dataset, Sequential recommendation, Machine learning, Recurrent neural networks, Long Short-Term Memory, Gated recurrent units, Matrix factorization

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešním moderním světě se stává cílená a rychle dostupná informace tím nejdůležitějším, co může být lidem poskytnuto. Doporučovací systémy jsou přesně takovými systémy, které umí relevantní informace a položky označit a vyhledat v nepřeberném množství obsahu nabízeného uživatelům.

V poslední době toto množství obsahu velice strmě roste a mnohdy není v silách uživatele celou nabídku konkrétní domény prozkoumat. Uživatel je tedy velice často závislý na tom, které produkty jsou mu nabídnuty jako první a ty jsou většinou seřazené chronologicky nebo abecedně. Uživatel pak nemusí vůbec nalézt většinu obsahu, který hledá. Například by se mohlo jednat o články z oblíbené kategorie (např. sport), které by měl rád ve výběru a nemusel je pokaždé složitě dohledávat. Takové články by uživatele zaujaly, přečetl by si je a udržely by ho na této doméně. V tento moment hrají doporučovací systémy velice důležitou roli, protože uživateli mohou pomoci nalézt přesně to, co hledá. Doporučení představující hledaný obsah je vytvořeno na základě předchozího chování uživatele a jeho preferencí. Takový systém je užitečný nejen pro uživatele, kterému usnadňuje vyhledávání, ale i pro poskytovatele obsahu, jemuž pomáhá systém udržet spokojené uživatele na doméně.

Motivace

Většina klasických přístupů v doporučovacích systémech, jako jsou kolaborativní filtrování, maticová faktorizace nebo doporučování založené na podobnosti položek nebo uživatelů, využívá interakční data ke generování doporučení. Interakční data mohou představovat hodnocení, komentář, rozkliknutí produktu, přídání do košíku, nákup produktu nebo sdílení článku a další akce, které uživatelé typicky provádějí na dané doméně.

Přístupy využívající tyto interakční data se na ně dívají jako na množinu ntic (uživatel, položka, akce nebo typ interakce, atd.), což je úplně základní pohled. Tyto přístupy ale vůbec nezohledňují ve svém doporučení sekvenční po-

vahu těchto dat v čase. Uživatelé nemusí mít stále stejné preference. Například rodiče nakupující pro své děti budou mít postupem času jiné požadavky na zboží. Výše zmíněné standartní přístupy tyto souvislosti vůbec nezvažují a mohou doporučovat věci v nelogickém sledu či nabízet věci již nepotřebné. Dalším příkladem může být navštěvování stránky více uživateli (např. rodinou), kde by klasický přístup bral v úvahu všechny interakce, ale model uvažující sekvenční data by po několika interakcích mohl změnu preferencí postřehnout a přizpůsobit doporučení.

Cíl práce

Cílem rešeršní části práce je nastudovat a prozkoumat interakční data v doporučovacích systémech a to výhradně zdroje interakčních dat a jejich předzpracování. Dále je důležité se seznámit s maticovou faktorizací a její modifikací na datové sady s implicitní zpětnou vazbou a využití maticové faktorizace pro generování vhodné reprezentace položek (angl. embeddings). Posledním cílem rešeršní části bude seznámení se s neuronovými sítěmi a to především s rekurentními neuronovými sítěmi, konkrétněji pak s LSTM (angl. Long Short-Term Memory, LSTM) a jejich využitím v doporučovacích systémech.

V praktické části je cílem vytvořit algoritmus pro tvorbu modelů založených na rekurentních neuronových sítích. Vytvořené modely budou sloužit ke generování doporučení na základě sekvenčních dat. K tomuto algoritmu je třeba vytvořit vhodný model pro vytváření embeddingů založených na maticové faktorizaci a tyto embeddingy využít v modelech generující doporučení. Dále je nutné všechny implementované modely vyhodnotit pomocí metrik recall a coverage a případně navrhnout způsob vyhodnocování zvažující sekvenční povahu dat. V neposlední řadě je potřebné dosažené výsledky vhodně prezentovat, porovnat s existujícími metodami a diskutovat další možné směry zlepšení.

Struktura práce

Tato práce je členěna do 5 kapitol a pokračuje v následující struktuře. V kapitole 1 je představen teoretický základ pro tuto práci, který zahrnuje především předzpracování interakčních dat, maticovou faktorizaci pro datasety s implicitní zpětnou vazbou a neuronové sítě se zaměřením na ty rekurentní. V kapitole 2 je představen návrh algoritmu pro generování modelů založených na rekurentních neuronových sítích, dále model maticové faktorizace a návrh metodiky pro vyhodnocování zahrnující sekvenčí povahu dat. Na tuto kapitolu plynule navazuje kapitola 3, ve které je popsána implementace a konfigurace běhu algoritmů. V kapitole 4 jsou prezentovány dosažené výsledky měřených experimentů, na které v kapitole 5 navazuje diskuze možných směrů zlepšení modelů.

Kapitola 1

Analýza

V této kapitole jsou představeny základní pojmy a principy, které byly využity a vztahují se k diplomové práci. První sekce je věnována interakčním datům pro doporučovací systémy a jsou popsány způsoby jejich předzpracování a reprezentace. V rámci této sekce jsou zmíněny nejběžnější doporučovací úlohy a především ty, které se v této práci používají a na jejichž principu budou vytvořeny modely. V následující sekci je popsána maticová faktorizace, která je využita ke generování vhodné reprezentace položek, ale také slouží jako jeden z referenčních modelů. V poslední části této kapitoly jsou rozebrány umělé neuronové sítě, konkrétně rekurentní neuronové sítě (LSTM, GRU, RNN), které jsou využity k sestavení modelů pro doporučování založeného na sekvenčních datech.

1.1 Interakční data

Doporučovací systémy (RSs) jsou systémy, které se snaží doporučit relevatní položky. K tomu, aby tyto systémy mohly doporučení generovat, tak RSs aktivně shromažďují různé druhy dat [2, s. 7]. Tato data jsou zpravidla buď atributová nebo interakční. Atributovými daty se v této práci zabývat nebudu. Většinou popisují jednotlivé položky nebo uživatele a využívají se v doporučování založeném na obsahu (angl. content-based recomendation). Oproti tomu interakční data tvoří množina položek (angl. items) a uživatelů (angl. users), kde interakce je nějaký zájem nebo nezájem uživatele o danou položku. Aby mohl být doporučovací systém úspěšný, musí se naučit uživatelské preference, ale je však velice obtížné získat kvalitní a reprezentativní zpětnou vazbu [3].

Formální definice a značení, které je upraveno, aby bylo v souladu s celou prací, je použito z knihy [4, s. 14]. Mějme tedy množinu n položek $I = \{i_1, i_2, \ldots, i_n\}$ a množinu m uživatelů $U = \{u_1, u_2, \ldots, u_m\}$. Interakční data jsou určité akce uživatelů s danými položkami. Interakční data tvoří množinu

$Y = \{y_1, y_2, \ldots, y_k\}$, kde y_i je jedna interakce reprezentovaná následující n-ticí $(u_i, i_i, t_i, t_i, d_i)$ s následujícím významem jednotlivých prvků n-tice [5]:

- $u_i \in U$ - identifikátor uživatele, kterému náleží interakce
- $i_i \in I$ - položka se kterou uživatel interagoval
- $t_i \in \mathbb{R}$ - časový údaj, kdy se interakce stala, který je velice důležitý pro využití sekvenční povahy dat
- d_i - doplňující data o dané interakci, která obsahují například typ dané interakce. Dále můžou zahrnovat třeba dobu trvání interakce nebo procentuální část viděného obsahu, například u videa

1.1.1 Zdroje interakčních dat

Pro interakční data, akce uživatelů s jednotlivými položkami, se nejčastěji používá označení zpětná vazba (angl, feedback) a dělí se na explicitní a implicitní zpětnou vazbu (interakce) [5] [3] [6] [7]. Tradiční literatura o doporučovacích systémech vychází většinou z explicitní zpětné vazby, ze které je možné přímo zjistit preference uživatele. Uživatel dává explicitně najevo, zdali se mu položka líbí, či nikoliv, nebo zdali k ní má neutrální vztah (počet hvězdiček nebo líbí/nelíbí). Problém explicitní zpětné vazby je ten, že vyžaduje určité úsilí od uživatelů. Ti však nemusí být ochotni věnovat svůj čas vyplnění hodnocení, což se může projevit na množství hodnocení a ve výsledku i na kvalitě doporučovacího systému [4, s. 22].

Asi nejznámějším příkladem, kde byla k dispozici datová sada složená pouze z explicitních hodnocení, které bylo cílem predikovat, byla Netflix Prize uskutečněná v letech 2006-2009 [8]. Hlavní cenou byl 1 000 000 $, který přilákal mnoho soutěžících a velice pomohl výzkumu v oblasti doporučovacích systémů. Soutěž například ukázala, že modely maticové faktorizace mohou být velice úspěšné [9]. Na explicitní zpětnou vazbu je také zaměřeno hodně výzkumů a často je cílem doporučovacího systému predikovat uživatelská hodnocení [6]. Chybu takového systému je pak možné jednoduše měřit pomocí odmocniny střední kvadratické odchylky (angl. root mean square error, RMSE) a porovnávat.

Na druhou stranu implicitní zpětná vazba může být sbírána, aniž by to uživatele nějak obtěžovalo a mnohdy o tom ani nemusí vědět. Tato zpětná vazba se přímo vybízí k modelování uživatelské preference v čase a využití sekvenčnosti dat. Například u přehrávaných písniček jednoduše zjistíme, že je písnička aktuálně oblíbená, pokud si ji uživatel pustí několikrát za den. Nejjednodušší cestou jak tuto vazbu získat je například log webového serveru. V publikaci [10] jsou uvedeny následující nejdůležitější charakteristiky implicitní zpětné vazby, které jsou uvedeny v následujícím seznamu.

- **Nezápornost** – Jsou k dispozici pouze informace o zkonzumovaném obsahu. O filmu, který uživatel neviděl, není možno prohlásit, že se uživateli nelíbil. Uživatel se k danému filmu například pouze nemusel dostat. Toto je jeden z největších rozdílů oproti explicitní zpětné vazbě, kde mohou uživatelé přímo říct, co se jim nelíbí. Je předpokládáno, že mezi chybějícími daty se potom pravděpodobně nachází většina obsahu, pro který by měl uživatel negativní zpětnou vazbu.

- **Zašumělost** – Implicitní zpětná vazba nutně obsahuje šum. Pokud máme pouze pozitivní zpětnou vazbu a data z internetového obchodu, tak se uživateli nemusely líbit všechny položky, které si zobrazil, ale mohl si zobrazit podrobnosti o položce jen pro zajímavost, nebo mohl zakoupený produkt vrátit.

- **Implictní zpětná vazba vyjadřuje důvěru** – U explicitní zpětné vazby můžeme přesně určit preference uživatele. Naopak u implicitní zpětné vazby se jedná spíše o spolehlivost nebo důvěru pro konkrétní položku. Produkt, který je kupován pravidelně, bude pravděpodobně pro uživatele důležitější než produkt, který si prohlédl jednou.

- **Nutnost odlišného vyhodnocování** – Pří vyhodnocování explicitní zpětné vazby je nejčastější metrikou střední kvadratická odchylka (MSE) nebo její odmocnina (RMSE), protože máme přesnou číselnou hodnotu, kterou je možné měřit na nějakém spojitém intervalu. Pro implicitní zpětnou vazbu je tedy nutné zavést jinou metriku, která bude závislá i na konkrétní doporučovací úloze. Obojí bude popsáno dále v této kapitole.

Pro doporučovací systémy se většinou sbírají oba typy zpětné vazby, aby bylo možné co nejvěrněji modelovat uživatelské preference a zájmy. Například v publikaci [6] byl vytvořen klasifikační framework pro porovnání explicitní a implicitní zpětné vazby, včetně využití obou společně. V článku [11] naopak popisují doporučovací systém založený na mapování implicitní zpětné vazby na explicitní pomocí logistické regrese.

Dále jsou uvedeny nejčastější zdroje implicitní zpětné vazby, na kterou je práce zaměřena, protože slouží převážně jako zdroj dat v doporučovacích systémech a poskytnuté datasety obsahují právě tuto zpětnou vazbu.

- **Zobrazení detailu produktu** – Uživatel rozklikne produkt v internetovém obchodě, podrobnosti o filmu ve videotéce nebo nějaký článek [5]. Jedná se o typický příklad zpětné vazby, který patří mezi ty nejčastěji získávaná data.

- **Přidání do košíku** – Představuje další typický příklad implicitní interakce, která je běžná z internetových obchodů. Většinou znamená, že produkt zaujal uživatele natolik, že si ho přidal do košíku [5].

- **Uložení položky** – Vyjadřuje uživatelův zájem o položku, kterou pravděpodobně nechce vidět naposled. Může se objevit ve streamovacích službách, diskuzních fórech, portálech inzerce nebo internetových novinách a uživatel se k položce pravděpodobně vrátí [5].
- **Napsání komentáře** – Uživatel k položce přidá komentář. Počet komentářů může představovat obecný zájem uživatele o danou položku [7]. Akci přidání komentáře je možné chápat jako implicitní zpětnou vazbu, ale pokud by se jednalo i o analýzu, zdali je komentář pozitivní nebo negativní, už by tato zpětná vazba patřila mezi ty explicitní.
- **Sdílení** – Sdílení jako zpětná vazba, která je známá ze sociálních sítí, může mít několik typů. Uživatel může sdílet položku veřejně na svůj profil nebo naopak ji může sdílet zprávou svému kamarádovi. To může představovat zájem, ale také například upozornění nebo znechucení. Nemusí se jednat o obsah, který chce uživatel pravidelně vídat, ale když už na něj narazil, tak se mu nelíbí natolik, že ho sdílí jako odstrašující.
- **To se mi líbí** – Vyjadřuje uživatelův zájem o položku, který je chápán jako pozitivní a může být považován za implicitní zpětnou vazbu, pokud zde není možnost opačné reakce *to se mi nelíbí* [5]. Pokud by existovaly obě možnosti, tak by se jednalo o explicitní zpětnou vazbu.
- **Nákup položky** – Uživatel si danou položku koupil. Této interakci může například předcházet interakce přidání do košíku. Pokud si uživatel položku kupuje pravidelně, je vhodné mu ji ve vhodný okamžik opět nabídnout.
- **Doba strávená na produktu** – Jak dlouho strávil uživatel na dané položce. Tento parametr je důležitý, protože nám může umožnit určit zájem uživatele na základě průměrného času prohlížení obsahu nebo čtení obsahu, pokud se jedná o článek [7].
- **Přehrání** – Jedná se o akci, kde si uživatel přehraje písničku nebo film. U písniček počet přehrání může přímo odrážet oblíbenost písničky. U filmů se vícenásobné přehrávání nevyskytuje v takové míře, ale o to více bude pro uživatele film důležitý, pokud si jej pustí vícekrát.
- **Poschlechnutá/zobrazená část** – Jedná se o typ interakce, která může souviset s hudbou, filmy, videem nebo článkem. Pokud uživatel viděl celý film nebo přečetl celý článek, má interakce určitě větší váhu, než pokud ukončil sledování filmu v půlce, přečetl pouze jeden odstavec článku a zavřel ho, nebo přešel na jiný. Tento typ implicitní interakce by se mohl nejvíce přiblížit explicitní zpětné vazbě bez toho, aby uživatel interakci přímo generoval nějakou svou cílenou akcí, např. hodnocením hvězdičkami. Několikrát přeskočená písnička hned po prvních pár tónech určitě neznamená pozitivní vztah uživatele k ní.

1.1.2 Předzpracování a reprezentace interakčních dat

Základními čtyřmi body v předzpracování dat jsou: čištění, integrace, redukce a jejich transformace [12].

- **Čištění dat** – Jedná se především o odstranění hraničních dat tzv. outlierů, doplnění chybějících hodnot a odhalení inkonzistence dat. V datech pro doporučovací systémy představuje čistění dat především identifikaci a odstranění robotů, dále odstranění uživatelů a položek s příliš málo interakcemi. Může to být například i odstranění duplicit se stejnou časovou známkou.

- **Intergrace dat** – Jedná se o kombinaci dat z více zdrojů. Jelikož vstupní data jsou předpokládaná jako množina interakcí Y, tak není integrace nutná. Jinak se může jednat například o integraci dat z mobilní aplikace nebo webu.

- **Redukce** – Nejčastěji je prováděná redukce dimenze. V doporučovacích systémech je nejběžnější redukcí pravděpodobně maticová faktorizace, kterou popíši v následující sekci.

- **Transformace** – Převedení dat do podoby, aby je bylo možné přímo využít pro doporučovací úlohu. V RSs pro implicitní data je využívána například agregace do ratingové matice nebo tvorba požadovaných sekvencí. U ratingové matice může být provedena například normalizace nad jejími řádky (uživatelskými vektory) k odstranění vychýlení (angl. bias) způsobeného zaujatým uživatelem.

Množinu interakcí Y, která byla uvedena na začátku 1.1, je nutné předzpracovat a reprezentovat v nějaké podobě, aby ji bylo možné využít v jednotlivých modelech a doporučovacích úlohách. Nejčastějším způsobem, jak reprezentovat data pro doporučovací systém, je využití matice hodnocení (angl. rating matrix), která tyto data agreguje.

1.1.2.1 Ratingová matice

Data ze sekce 1.1 lze agregovat do ratingové matice o rozměrech $m \times n$, kde m je počet uživatelů a n počet položek. Ratingovu matici tedy můžeme označit následovně $\mathbf{R} \in \mathbb{R}^{m \times n}$, kde každý prvek matice $r_{u,i}$ reprezentuje vztah uživatele $u \in U$ a položky $i \in I$ [5].

Pokud vezmeme v úvahu kapitolu 1.1.1, tak je jasné, že interakční data bude nutné do ratingové matice nějak agregovat. Pokud by bylo k dispozici pouze explicitní hodnocení, můžou hodnoty v matici představovat právě pouze hodnocení. V této práci se však předpokládá implicitní zpětná vazba, kde není známa žádná přesná hodnota. Jednotlivé váhy pro dané interakce je nutné vhodně zvolit. Z článku [11] je například možné implicitní zpětnou

vazbu namapovat na explicitní nebo využít možnost společné agregace implicitní a explicitní zpětné vazby. V tomto případě je však ještě důležitější zvolit odpovídající váhy a aplikovat například normalizaci pro implicitní zpětnou vazbu.

Pro implicitní zpětnou vazbu můžou hodnoty v ratingové matici představovat například počet interakcí uživatele u s položkou i [13]. Například nákup položky má určitě větší váhu než její zhlédnutí. Je potřeba každému typu interakce určit jeho váhu a ty poté agregovat do matice. Například nedoposlechnutá písnička by mohla mít váhu i negativní. Nejčastější agregací pro tyto hodnoty je sčítání. Hodnoty mohou být omezeny maximem a minimem pro omezení hranic zájmu o danou položku.

Jednou z motivací pro použití doporučovacího systému je usnadnění vyhledávání položek pro uživatele. Je proto nemožné, aby uživatel interagoval s většinou položek z I. Z tohoto důvodu je ratingová matice velice řídká (angl. sparse matrix). O nevyplněných dvojicích, kdy uživatel s danou položkou ještě neiteragoval, není možné nic prohlásit, protože se s ní ještě vůbec nemusel setkat. Z disertační práce [5] je možné rozšířit původní označení na $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{m \times n}$. Matice je velmi řídká, je vyplněno velmi málo hodnot. Nevyplněné hodnoty nejsou známé a jsou brány jako chybějící. V mnoha algoritmech je však předpokládáno, že jsou tyto hodnoty 0, aby bylo možné s maticí jednoduše pracovat v její husté reprezentaci.

Reprezentace pomocí ratingové matice se řadí mezi standartní a často používaný přístup. Tuto reprezentaci předpokládají například v [14] [9] [15] a je používána především v kolaborativním filtrování. Jednou z nevýhod je, že není použitelná pro modely, které mají být založeny na sekvenční povaze dat. Ratingová matice vůbec nebere v úvahu čas, kdy byla daná interakce provedena.

Způsob reprezentace dat pomocí ratingové matice v RSs je v práci využita především pro maticovou faktorizaci, která je popsána v sekci 1.2. Pro rekurentní neuronové sítě je nutné zavést reprezentaci respektující sekvenční povahu dat.

1.1.2.2 Reprezentace sekvenčních dat

Použití sekvenční povahy dat pro doporučování je cílem výzkumu v mnohem menší míře než klasické použití ratingové matice a metod kolaborativního filtrování. V reprezentaci dat tato práce vychází z [16], kde se zabývají získáváním častých sekvencí z dat, ale především popisují data v podobném formátu, který bude využit i v této práci. V případě poslechu hudby si uživatelé rádi pustí oblíbenou písničku víckrát. Bylo by tedy vhodné ji v ten správný čas doporučit a umět doporučovat opakovaně. Tento přístup není přímo možný v běžné formě kolaborativního filtrování nebo maticové faktorizace.

Sekvenční povaha dat také může zachytit vývoj uživatelských preferencí v čase [17]. Uživatelské preference se mění s roční dobou, růstem dětí, atd.

V případě kolaborativního filtrování je uvažována celá historie uživatele, která ho popisuje. Například při poslouchání hudby, kdy uživatel přejde na jiný hudební žánr, mu kolaborativní fitrování může předchozí hudební styl nabízet ještě po nějakou delší dobu.

Pro každého uživatele chceme ukládat celou jeho historii interakcí v čase a umět předpovědět další položku, na kterou se bude chtít podívat (zkonzumovat). Reprezentace může být stejná jako 1.1 a jedná se o n-tice, které je možné rozdělit podle jednotlivých uživatelů.

V článku [16] autoři uvádějí historii pro uživatele jako sekvenci sekvencí (jednotlivých nákupů), která by s respektem na značení v 1.1, kde máme množinu interakcí Y, mohla být zápsána jako $((y_1), (y_6, y_7, y_1), (y_1, y_2))$.

Pro tuto práci je reprezentace zjednodušena, protože data především z online webových stránek neobsahují interakce v subsekvencích, které jsou typické například pro jednotlivé nákupy. Tyto subsekvence jsou v datech reprezentovány pomocí časové známky, protože položky pro uživatele se stejnou časovou známkou určitě představují právě jednu subsekvenci. V této práci bude snahou předpovídat následující interakci. Z tohoto důvodu je výše uvedená sekvence relaxována do následujícího zápisu. Mějme nějakou sekvenci $s = (y_1, y_6, y_7, y_1, y_1, y_2) \in S$ uživatele u, kde S představuje množinu všech možných sekvencí, pro které platí $y_k \in Y$. Dále musí platit, že v jedné sekvenci jsou interakce pouze pro jednoho uživatele $\forall(u_i, u_j \in s_l)(s_l \in S \ \wedge u_i = u_j)$ a zároveň je daná sekvence seřazená v pořadí, jak byly provedeny jednotlivé interakce v čase.

Také není nutné, aby pro jednoho uživatele byla pouze jedna sekvence. Sekvence je možné předzpracovat a z jedné sekvence můžeme vynechat poslední položku a vytvořit tím novou sekvenci. To je výhodné například u případů, kde poslední interakce jsou v nějaký společný časový okamžik.

Takto vytvořené sekvence mohou představovat data, která budou sloužit jako vstup pro modely založené na rekurentních neuronových sítích. Dále tato data jsou využita pro vyhodnocování úspěšnosti, kde oproti metodám vyhodnocování použitých v mé bakalářské práci [1], bude nutné navrhnout nový systém pro testování úspěšnosti, který bude sekvenční povahu dat respektovat.

1.1.3 Standartní přístupy v doporučovacích systémech

Cílem doporučovacích systémů je poskytnout uživateli položky, které by pro něj mohly být relevantní a které by se mu nejvíce líbily v obrovském množství položek, které jsou nabízeny [10]. Výběr těchto položek může být výsledkem různých doporučovacích úloh.

Na základě výsledku těchto úloh vybereme ty nejrelevantnější položky, které jsou uživateli doporučeny. V této části jsou vyjmenovány různé existující doporučovací úlohy a představeny ty nejčastější. Dále je popsán princip

kolaborativní filtrování a krátce uvedeny metodiky vyhodnocování úspěšnosti v souvislosti s motivací pro tvorbu sekvenčních modelů.

1.1.3.1 Doporučovací úlohy

Nejprve je uvedena definice obecného učícího algoritmu s hyperparametrizacemi [5]: Předpokládejme učící algoritmus $\mathcal{A}$, který je sám o sobě funkcí, která jako vstup bere data a vytváří model. Jedná se o úlohu učení s učitelem. Mějme tedy množinu k trénovacích dat $\mathcal{T} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_k, \mathbf{y}_k)\}$, kde pro nějaké abstraktní množiny $\mathcal{X}$ a $\mathcal{Y}$ musí platit, že $\forall l \in 1, \ldots, k : \mathbf{x}_l \in \mathcal{X} \wedge \mathbf{y}_l \in \mathcal{Y}$. Algoritmus $\mathcal{A}$ potom produkuje model m: $\mathcal{X} \to \mathcal{Y}$, který můžeme zapsat:

$$\mathcal{A}\colon 2^{\mathcal{X}} \times \mathcal{Y} \to \mathcal{Y}^{\mathcal{X}}, \tag{1.1}$$

kde $2^{\mathcal{X}\times\mathcal{Y}}$ je množina všech možných učících sad a $\mathcal{Y}^{\mathcal{X}}$ jsou všechny projekce z $\mathcal{X}$ do Y. Model vždy obsahuje nějakou hyperparametrizaci $P \in \mathcal{P}_{\mathcal{A}}$. Potom $\mathcal{P}_{\mathcal{A}}$ představuje prostor všech hyperparametrizací a po přidání do definice mějme:

$$\mathcal{A}\colon \mathcal{P}_{\mathcal{A}} \times 2^{\mathcal{X}\times\mathcal{Y}} \to \mathcal{Y}^{\mathcal{X}} \tag{1.2}$$

Hyperparametrizaci může představovat například počet sousedů v algoritmu k-NN nebo počet neuronů ve skryté vrstvě u neuronové sítě apod.

První konkrétní doporučovací úlohou bude **predikce ratingu** (angl. rating prediction, RP). Díky Netflix price [8] je RP asi nejznámější doporučovací úlohou a dlouhou dobu tato predikce byla také nejčastěji studovanou úlohou [9] [18]. Úlohou modelu je pro každou položku, co uživatel neviděl, odhadnout rating, který by jí uživatel dal. Podle článku [6] v této úloze mějme:

$$score\colon U \times I \ \to \mathbb{R}, \tag{1.3}$$

kde *score* představuje funkci užitečnosti položky $i \in I$ pro uživatele $u \in U$. Hodnocení, které v matici není vyplněné $r_{i,j} \in R$ (označené jako „?“), potom můžeme predikovat $r_{i,j} = score(u, i)$. Podle výše zmíněné obecné definice zápisu ratingové matice ze sekce 1.1.2.1 a z disertační práce [5], je možné úlohu zapsat následujícím způsobem:

$$\mathcal{A}^{\mathrm{RP}}\colon \mathcal{P}_{\mathcal{A}^{\mathrm{RP}}} \times (\mathbb{R} \cup \{?\})^{m\times n} \to \mathbb{R}^{m\times n}, \tag{1.4}$$

kde $(\mathbb{R} \cup \{?\})^{m\times n}$ představuje všechna možná tréninková data jako řídkou matici s neznámými hodnotami ? a $\mathbb{R}^{m\times n}$ už představuje vyplněnou (angl. dense) matici s predikovanými hodnoceními.

Další velmi známou doporučovací úlohou je **top-N** doporučovací úloha. Jak je jasné z názvu, cílem je doporučit n nejvíce relevantních položek. Tato úloha je asi tou nejčastější úlohou, která je využívána v doporučovacích systémech v praxi. Například na webové stránce je zobrazeno 5 boxů, ve kterých se

nachází ty nejrelevantnější doporučení. Z předchozí úlohy je však možné přejít na top-N doporučení bez nějakého problému. Vygenerujeme hodnocení pro všechny položky a doporučíme uživateli položky s největší hodnotou skóre. V top-N jsou položky při generování také nějak interně oskorované a jsou vráceny jen ty nejrelevantnější. Tuto úlohu lze v návaznosti na obecnou definici 1.2 zapsat jako [5]:

$$\mathcal{A}^{\text{top}-N}\colon \mathcal{P}_{\mathcal{A}^{\text{top}-N}} \times (\mathbb{R} \cup \{?\})^{m \times n} \to \{I' \subset I \ \mid |I'| = N\}^m \tag{1.5}$$

Tato úloha předpokládá jako vstup matici hodnocení s neznámými hodnotami a je typická pro doporučování pomocí kolaborativního filtrování s využitím ratingové matice. Pro sekvenční modely potřebujeme ale využít sekvenční data definovaná v sekci 1.1.2.2. Pro sekvenční data mějme model upravený následovně:

$$\mathcal{A}^{\text{top}-N}\colon \mathcal{P}_{\mathcal{A}^{\text{top}-N}} \times S \ \to \{I' \subset I \ \mid |I'| = N\}^m, \tag{1.6}$$

kde S je množina všech sekvencí uvedená v sekci 1.1.2.2. Pokud bychom uvažovali, že každý uživatel je zastoupen v trénovací množině pouze jednou sekvencí, je možné označení zpřesnit a označit S jako S^m. V této práci je však předpodkládáno, že v modelu může být využito i více sekvencí pro jednoho uživatele. Tyto sekvence mohou být vhodně upraveny a předzpracovány.

1.1.3.2 Kolaborativní filtrování (angl. Collaborative filtering, CF)

Jak je zmíněno v 1.1.3.1, tak v klasických doporučovacích metodách je snahou doporučit položku, kterou uživatel ještě neviděl a byla by pro něj zajímavá. Tento přístup je především známý z metod CF, které byly představeny v publikaci [19] v roce 1992. Algoritmy je možné rozdělit na metody pro hledání nejbližších vektorů (sousedů) a metody latentních příznaků (angl. latent factors). Jedním z nejpoužívanějších je algoritmus pro hledání nejbližších uživatelů (angl. user-knn). Z metod latentích příznaků se jedná určitě o maticovou faktorizaci, které je v práci věnována následující sekce 1.2, protože je využita i při návrhu modelu.

Zrychlování těchto algoritmů a měření jejich úspěšnosti byla věnována má bakalářská práce [1], kde bylo ovšem uvažováno pouze doporučování položek, které uživatel ještě neviděl a také vyhodnocování modelů bylo na tomto principu založeno. Tato práce vyjde z těchto principů, ale testování bude rozšířeno na sekvenční modely.

Tento přístup a tyto modely jsou velice úspěšné i přes skutečnost, že umožňují doporučování pouze takových položek, které uživatel neviděl. CF může být stále velice vhodné u článků nebo zboží, které se nekupuje tak často. Použití sekvenčních modelů může být výhodné pouze pro nějaké domény.

U CF a nejbližších sousedů je často využita nějaká metrika podobnosti, pomocí které jsou hledáni podobní uživatelé nebo položky. Algoritmus poté

Tabulka 1.1: Matice záměn [2]

		doporučení	
		ano	ne
skutečnost	relevantní	true-positive (TP)	false-negative (FN)
	nerelevantní	false-positive (FP)	true-negative (TN)

využívá metriku k počítání podobnosti mezi vektory v ratingové matici. Nejčastěji používanou metrikou je kosinová podobnost

$$s\left(\mathbf{x},\mathbf{y}\right)=\frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}=\frac{\sum_{i=0}^{n-1}x_{i}y_{i}}{\sqrt{\sum_{i=0}^{n-1}x_{i}^{2}}\sqrt{\sum_{i=0}^{n-1}y_{i}^{2}}},\tag{1.7}$$

kde $\mathbf{x}$ a $\mathbf{y}$ mohou být buď řádkové nebo sloupcové vektory z ratingové matice $\mathbf{R}$. Podobnost je zde uvedena, protože bude využita v navrhovaném modelu pro sekvenční data.

Na druhou stranu, pokud bychom se zaměřili na doménu poslouchání hudby, sledování filmů nebo nakupování spotřebního zboží, může být vhodné umět do doporučení vložit položky, se kterými již uživatel interagoval, ale představují pro něj položku, kterou by právě teď chtěl opět koupit, zhlédnout nebo přehrát.

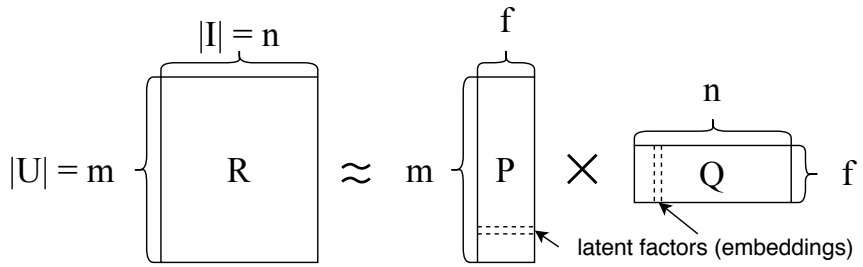
1.1.3.3 Vyhodnocování úspěšnosti

V mé bakálářské práci [1] byly popsány metody vyhodnocování úspěšnosti a byla použita leave-one-out křížová validace, která ovšem pro účely této práce není vhodná, protože nerespektuje sekvenčnost chování. V kapitole 2 jsou navrženy úpravy metod vyhodnocování, které budou k vyhodnocení úspěšnosti využity a vycházejí ze stejných základů. Dále jsou stručně zmíněné hlavní metriky, které je možné měřit v top-N doporučovací úloze. Oproti predikci ratingu, kde je možné využít například MSE nebo RMSE, u top-N doporučování můžeme uvažovat pouze o relevantnosti doporučení.

Z tabulky 1.1, která představuje matici záměn, můžeme podle [2] a [1] uvažovat následující nejčastěji používané metriky, které jsou využity například v [17] a [20]. Metriky jsou využity v top-N doporučovací uloze, kde N představuje počet doporučených položek:

$$Precision@N=\frac{\#TP}{\#TP+\#FP}\tag{1.8}$$

$$Recall@N=\frac{\#TP}{\#TP+\#FN}\tag{1.9}$$



Obrázek 1.1: Ukázka hlavní myšlenky maticové faktorizace. Aproximace, která je naznačená na obrázku platí pouze pro známé hodnoty v matici **R**. Obrázek byl vytvořen na základě obrázku publikovaném na blogu [21]

Catalog coverage je poslední metrikou, která měří procento položek, které je model schopen doporučit. U této metriky je důležité její hodnoty porovnávat vždy pro stejnou nebo alespoň stejně velkou testovací sadu, protože se mohou výrazně lišit:

$$catalog\ coverage = \frac{|\bigcup_{u \in U} I'_u|}{|I|}, \tag{1.10}$$

kde I'_u je množina položek doporučených jednomu uživateli.

1.2 Maticová faktorizace (MF)

Maticová faktorizace je jedna z nejčastěji používaných metod v CF. MF patří mezi modely latentních příznaků, jak je uvedeno v sekci 1.1.3.2. Latentní příznaky jsou většinou husté vektory nižší dimenze, které se snaží nějakým způsobem vysvětlit interakce z původního vektoru. Latentní příznaky můžeme mít, jak pro uživatele, tak i pro položky, jak je možné vidět z obrázku 1.1. Dalo by se říci, že se jedná o zakódovanou informaci o interakcích a vektor latentních příznaků může reprezentovat danou položku nebo uživatele. Tyto myšlenky jsou využity i při návrhu.

Jak je již zmíněno v sekci 1.1.1, MF se stalo populární v RSs především díky Netflix price [8]. Díky této soutěži je nejčastěji popisovaná základní varianta založená na explicitní zpětné vazbě. Tato diplomová práce vychází z předpokladu ratingové matice, která byla představena v 1.1.2.1. Matice se může skládat buď z explicitní zpětné vazby, implicitní zpětné vazby nebo být jejich kombinací. Nejprve jsou uvedeny zakládní myšlenky a metody maticové faktorizace, které většinou předpokládají explicitní ratingy. Poté je představena klíčová myšlenka pro maticovou faktorizaci, která předpokládá datasety složené z implicitní zpětné vazby.

Na obrázku 1.1 je možné vidět hlavní myšlenku maticové faktorizace. Jak je zmíněno v publikaci [10] a [22], MF aproximuje pouze známé hodnoty v $\mathbf{R}$ součinem dvou matic $\mathbf{P} \in \mathbb{R}^{f \times m}$ a $\mathbf{Q} \in \mathbb{R}^{f \times n}$ nižší dimenze. Pro hodnoty $r_{u,i} = ?$ nemáme vůbec žádnou informaci, a proto se nemůže jednat o jejich aproximaci. Jedná se pouze o jejich predikci.

Podle článku [9] mějme latentní vektor uživatele $\mathbf{p}_{:u}$, který představuje řádek matice $\mathbf{P}$ s indexem u, kde $\mathbf{p}_{:u} \in \mathbb{R}^f$. Obdobně pro položky $\mathbf{q}_{:i}$, kde $\mathbf{q}_{:i} \in \mathbb{R}^f$. Dále v souladu s literaturou [9] [10] jsou tyto vektory značené zkráceně $\mathbf{p}_u$ a $\mathbf{q}_i$.

Volba f je většinou taková, že je několikanásobně menší než m a n. Jedná se tedy o redukci dimenzionality. Prvky latentních vektorů představují nějakým způsobem zakódovanou informaci o interakcích. Například u $\mathbf{p}_u$, vektoru uživatele, představují jeho zájem o odpovídající položky [9]. Pro neznámou hodnotu $r_{u,i} = ?$ v ratingové matici $\mathbf{R}$ predikujeme rating pomocí matic $\mathbf{P}$ a $\mathbf{Q}$ následujícím způsobem [10]:

$$\hat{r}_{u,i} = \mathbf{q}_i^T \mathbf{p}_u \tag{1.11}$$

Pro známé hodnoty $r_{u,i} \neq ?$ v $\mathbf{R}$ se jedná o aproximaci těchto hodnot. Z toho také vycházejí algoritmy, které se snaží $\mathbf{P}$ a $\mathbf{Q}$ najít. U MF se na rozdíl oproti algoritmům k-NN nachází výpočetně náročná trénovací fáze, kde je snahou najít rozklad na matice latentních příznaků. Vybavování v klasické MF je poté jen rychlé násobení dvou vektorů nízké dimenze. V rámci MF se chyba aproximace nejčastěji počítá pomocí kvadratické chyby:

$$(r_{u,i} - \mathbf{q}_i^T \mathbf{p}_u)^2, \tag{1.12}$$

která pak odpovídá následující optimalizační úloze

$$\min_{\substack{\mathbf{P} \in \mathbb{R}^{f \times m} \\ \mathbf{Q} \in \mathbb{R}^{f \times n}}} \sum_{\substack{u \in U \\ i \in I \\ r_{u,i} \neq ?}} (r_{u,i} - \mathbf{q}_i^T \mathbf{p}_u)^2. \tag{1.13}$$

Počítání chyby i optimalizační úloha se pro jednotlivé metody a přístupy liší. Ty nejznámější a nejpoužívanější metody pro řešení tohoto problému představím v následujících podsekcích.

1.2.1 Singulární rozklad matice (SVD)

Singulární rozklad matice, dále jen zkráceně SVD, je velice známa metoda pro získání rozkladu matice a tedy i latentních příznaků. Nalezení tohoto rozkladu spočívá v hledání vlastních čísel a vlastních vektorů. Předpokladem pro metodu je existence ratingové matice, která je značena M. Jedná se o ratingovou matici, ale značení je upraveno z důvodu, že jako vstup do metody tato matice nemůže obsahovat neznáme prvky „?“. Podle publikace [4] mějme tedy matici

M o rozměrech $m \times n$, kterou můžeme rozložit na součin tří matic následujícím způsobem:

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{1.14}$$

kde **U** je ortogonální matice o rozměrech $m \times n$, **V** je také ortogonální matice o rozměrech $n \times n$ a $\mathbf{\Sigma}$ je diagonální matice nazývaná singulární a má rozměry $m \times n$, kde prvky na diagonále jsou seřazené sestupně podle velikosti.

Hledaní vlastních vekttorů je prováděno na maticích $\mathbf{M}\mathbf{M}^T$ a $\mathbf{M}^T\mathbf{M}$, kde v tomto pořadí U a V mají vlastní vektory zapsané do sloupců. Singulární hodnoty jsou odmocniny z vlastních čísel jednoho ze zmíněných součtů.

Jak je ukázáno v článku [23], tak matice nalezené pomocí metody SVD jsou optimální aproximací při použití x největších singulárních hodnot. Platí tedy, že $(h(\mathbf{\Sigma}) = h(\mathbf{M})) \leq min(m, n)$. Pokud $m \neq n$, tak se může stát, že některé řádky nebo sloupce $\mathbf{\Sigma}$ budou nulové a ty je možné bez ztráty přesnosti odstranit. Podle článku [23] je takové řešení přesné a rozklad existuje. Při dalším odstraňování nenulových řádků (cíl je mít počet latentních příznaků výrazně nižší než jsou rozměry původní matice) od nejmenších singulárních hodnot v matici $\mathbf{\Sigma}$ (ponecháme pouze f největších) získáváme opět optimální aproximaci.

Jak je uvedeno v publikaci [9], nevýhodou této metody je fakt, že neumí zacházet s neznámými hodnotami **R**. Tyto hodnoty je možné doplnit například nulami, ale to bude matice obsahovat mnoho nevýznamných dat a metoda se stane výpočetně neupočitatelnou, pokud rozměr **R** může být až v desítkách či stovkách miliónů. Proto existují i další metody, které ovšem nemusí a většinou nenajdou optimální rozklad, avšak jsou lehce paralelizovatelné, umí pracovat s řídkými maticemi nebo jsou výpočetně méně náročné na rozdíl od metody SVD.

1.2.2 Stochastický gradientní sestup (SGD)

Jednou z takových metod je stochastický gradientní sestup. Při optimalizaci a hledání rozkladu pomocí SGD je nutné optimalizační problém uvedený ve výrazu 1.13 rozšířit o regularizaci, aby nedocházelo k přeučování na známé hodnocení. Regularizace může být využita nejen pro metody SGD. Podle článku [9] pak máme následující optimalizační úlohu:

$$\min_{\substack{\mathbf{P}\in\mathbb{R}^{f\times m}\\ \mathbf{Q}\in\mathbb{R}^{f\times n}}} \sum_{\substack{u\in U\\ i\in I\\ r_{u,i}\neq ?}} (r_{u,i} - \mathbf{q}_i^T\mathbf{p}_u)^2 - \lambda(\parallel \mathbf{p}_u \parallel^2 + \parallel \mathbf{q}_i \parallel^2) \tag{1.15}$$

Použití regularizace je zde z důvodu přecházení přeučení na známé hodonoty, protože cílem MF je generalizace a poté predikce $r_{u,i} = ?$. Míra regularizace je řízena parametrem λ.

V gradientním sestupu je snaha predikovat podle výrazu 1.11, kde $r_{u,i} \neq ?$ a predikci porovnat s hodnotou, kterou by měla predikce nabýt. Ná základě chyby poté upravíme hodnoty v $\mathbf{p}_u$ a $\mathbf{q}_i$. Podle článku [9] tedy iterujeme přes všechny hodnoty $r_{u,i} \neq ?$, počítáme chybu a provádíme aktualizaci hodnot podle pravidel. Takto je vypočtena chyba pro jedno hodnocení [9]:

$$e_{u,i} = r_{u,i} - \mathbf{q}_i^T \mathbf{p}_u \tag{1.16}$$

Aby bylo možné podle chyby aktualizovat dosavadní hodnoty $\mathbf{p}_u$ a $\mathbf{q}_i$, je nutné odvodit pravidla pro jejich aktualizaci pomocí gradientního sestupu. Výraz v optimalizační úloze 1.15 je označen jako objektivní funkce F:

$$F(\mathbf{P}, \mathbf{Q}) = \sum_{\substack{u \in U \\ i \in I \\ r_{u,i} \neq ?}} (r_{u,i} - \mathbf{q}_i^T \mathbf{p}_u)^2 - \lambda(\| \mathbf{p}_u \|^2 + \| \mathbf{q}_i \|^2) \tag{1.17}$$

Poté jsou spočítány parciální derivace následujícím způsobem:

$$\frac{\partial F}{\partial p_u^{(j)}} = -2(r_{u,i} - \mathbf{q}_i^T \mathbf{p}_u) q_i^{(j)} + \lambda p_u^{(j)} \tag{1.18}$$

$$\frac{\partial F}{\partial q_i^{(j)}} = -2(r_{u,i} - \mathbf{q}_i^T \mathbf{p}_u) p_u^{(j)} + \lambda q_i^{(j)}, \tag{1.19}$$

kde (j) značí jednotlivé prvky vektorů $\mathbf{p}_u$ a $\mathbf{q}_i$. Výraz v kulatých závorkách je nahrazen chybou uvedenou výše a je dodán parametr pro učení γ a konstantu 2 je možno zanedbat stejně jako v [9]. Pravidla pro aktualizaci hodnot je možné zapsat následujícím způsobem [9]:

$$\mathbf{p}_u \leftarrow \mathbf{p}_u + \gamma(e_{u,i}\mathbf{q}_i - 2\lambda\mathbf{p}_u) \tag{1.20}$$

$$\mathbf{q}_i \leftarrow \mathbf{q}_i + \gamma(e_{u,i}\mathbf{p}_u - 2\lambda\mathbf{q}_i) \tag{1.21}$$

S využitím pravidel poté můžeme zapsat SGD způsobem, který je uveden v algoritmu 1. Tato metoda předpokládá matici obsahující explicitní zpětnou vazbu.

Tato metoda je velice oblíbenou pro svoji jednoduchost a výpočetní rychlost. Iterace přes všechny známe hodnoty můžeme opakovat v cyklu pro dosažení co nejmenší celkové chyby. Ale jelikož neznámé jsou $\mathbf{p}_u$ i $\mathbf{q}_i$, tak optimalizace není konvexní [9]. Může být také složitější určit parametr γ pro optimální kroky. Dále může být optimalizace pomalá a může být vhodné použití jiné metody a to například metody ALS popsané v další podsekci. Metoda gradientního sestupu, která je také používaná při trénování neuronových sítí, je popsána v kapitole 1.3.

Algoritmus 1 Stochastický gradientní sestup [5] [9]

Input: ratingová matice $\mathbf{R} \in \mathbb{R}^{m \times n}$, počet iterací $l \in \mathbb{N}$, počet faktorů $f \in \mathbb{N}$, regularizace $\lambda \in \mathbb{R}$ a učící parametr $\gamma \in \mathbb{R}$
Output: $\mathbf{P} \in \mathbb{R}^{f \times m}$ a $\mathbf{Q} \in \mathbb{R}^{f \times n}$

```
initialization P                          ▷ většinou náhodná inicializace
initialization Q
for _ from 0 to l do   ▷ lze použít pravidlo konvergence místo konečného l
    for u from 0 to m do
        for i from 0 to n do
            if r_{u,i} ≠ ? then
                e_{u,i} ← r_{u,i} − q_i^T p_u
                p_u ← p_u + γ(e_{u,i} q_i − 2λ p_u)
                q_i ← q_i + γ(e_{u,i} p_u − 2λ q_i)
return: P, Q
```

1.2.3 Metoda alternujících nejmenších čtverců (ALS)

Hlavní myšlenkou této metody je nemít dvě neznámé $\mathbf{p}_u$ i $\mathbf{q}_i$, ale vždy pouze jednu. Toho je dosaženo střídavým fixováním $\mathbf{p}_u$ a $\mathbf{q}_i$ a odtud také vychází název metody. Když je $\mathbf{p}_u$ fixováno, metoda optimalizuje $\mathbf{q}_i$, kde díky fixaci je zajištěna konvexnost a tedy i konvergence [9]. V této podsekci se opět vychází z předpokladu, že $\mathbf{R}$ je složena z explicitní zpětné vazby.

Pro odvození ALS je nutné vyjít opět z optimalizační úlohy 1.13, přesněji z objektivní funkce 1.17. Pro odvození vyjdeme z derivace F. Oproti výrazu 1.18 a 1.19 je provedena derivace buď podle vektoru $\mathbf{q}_i$ nebo $\mathbf{p}_u$ vždy s tím, že opačná proměnná (matice) je fixována a je k ní přistupováno jako ke konstantě. F je zderivováno podle $\mathbf{q}_i$ následujícím způsobem:

$$\frac{\partial F}{\partial \mathbf{q}_i} = -2 \sum_u (r'_{i,u} - \mathbf{q}_i^T \mathbf{p}_u) \mathbf{p}_u^T + 2\lambda \mathbf{q}_i^T \tag{1.22}$$

První derivaci je možné přepsat s využitím matic do následujícího tvaru za předpokladu, že $\mathbf{p}_u$ je konstanta a tedy i $\mathbf{P}$:

$$-2(\mathbf{r}'_i - \mathbf{q}_i^T \mathbf{P}) \mathbf{P}^T + 2\lambda \mathbf{q}_i, \tag{1.23}$$

kde $\mathbf{r}_i$ představuje vektor hodnocení pro položku i o rozměru $1 \times m$. A platí pro něj následující:

$$\mathbf{r}'_i = \begin{cases} r_{u,i} & r_{u,i} \neq ? \\ 0 & r_{u,i} = ? \end{cases} \forall u \ \in U$$

Jedná se tedy o nahrazení chybějících hodnot nulami. Poté je gradient položen rovný 0 a je získána tzv. normální rovnice (angl. normal equation):

$$0 = -(\mathbf{r}_i - \mathbf{q}_i^T\mathbf{P})\mathbf{P}^T + \lambda\mathbf{q}_i \tag{1.24}$$

Pomocí maticových operací pak z rovnice 1.24 vyjádříme $\mathbf{q}_i$:

$$\begin{aligned} 0 &= -(\mathbf{r}_i - \mathbf{q}_i^T\mathbf{P})\mathbf{P}^T + \lambda\mathbf{q}_i^T \\ 0 &= -\mathbf{r}_i\mathbf{P}^T + \mathbf{q}_i^T\mathbf{P}\mathbf{P}^T + \lambda\mathbf{q}_i^T \\ \mathbf{r}_i\mathbf{P}^T &= \mathbf{q}_i^T(\mathbf{P}\mathbf{P}^T + \lambda\mathbf{I}) \\ \mathbf{P}\mathbf{r}_i^T &= (\mathbf{P}\mathbf{P}^T + \lambda\mathbf{I})\mathbf{q}_i \\ \mathbf{q}_i &= (\mathbf{P}\mathbf{P}^T + \lambda\mathbf{I})^{-1}\mathbf{P}\mathbf{r}_i^T, \end{aligned} \tag{1.25}$$

kde I je jednotková matice o rozměru $f \times f$. Obdobně pro $\mathbf{p}_u$:

$$\frac{\partial F}{\partial \mathbf{p}_u} = -2\sum_i (r_{u,i} - \mathbf{p}_u^T\mathbf{q}_i)\mathbf{q}_i^T + 2\lambda\mathbf{p}_u^T \tag{1.26}$$

$$\begin{aligned} 0 &= -(\mathbf{r}_u - \mathbf{p}_u^T\mathbf{Q})\mathbf{Q}^T + \lambda\mathbf{p}_u^T \\ \mathbf{r}_u\mathbf{Q}^T &= \mathbf{p}_u^T(\mathbf{Q}\mathbf{Q}^T + \lambda\mathbf{I}) \\ \mathbf{p}_u &= (\mathbf{Q}\mathbf{Q}^T + \lambda\mathbf{I})^{-1}\mathbf{Q}\mathbf{r}_u^T \end{aligned} \tag{1.27}$$

Pro aktualizaci v ALS dostaneme následující pravidla:

$$\mathbf{q}_i \leftarrow (\mathbf{P}\mathbf{P}^T + \lambda\mathbf{I})^{-1}\mathbf{P}\mathbf{r}_i^T \tag{1.28}$$

$$\mathbf{p}_u \leftarrow (\mathbf{Q}\mathbf{Q}^T + \lambda\mathbf{I})^{-1}\mathbf{Q}\mathbf{r}_u^T \tag{1.29}$$

Obdobně jako u SGD jsou pravidla využita pro konstrukci algoritmu. Celé ALS pak probíhá způsobem, jako je uvedeno v algoritmu 2, kde je také vidět, že je algoritmus mnohem lépe paralelizovatelný než metoda SGD. U ALS může být každá iterace spuštěna na libovolném počtu vláken a synchornizace je aplikována pouze mezi iteracemi.

1.2.4 Přidání vychýlení pro MF

Tento přístup opět vychází z ratingové matice bez implicitní zpětné vazby. Vychází z myšlenky, že některý uživatel může být více kritický a využívat celou škálu hodnocení. Na druhou stranu některé uživatelské vektory mohou být vychýlené, protože uživatel dává systematicky vyšší hodnocení [9]. Je patrné, že toto vychýlení se bude vyskytovat především u explicitní zpětné vazby.

Algoritmus 2 Alternujících nejmenší čtverce

Input: ratingová matice $\mathbf{R} \in \mathbb{R}^{m \times n}$, počet iterací $l \in \mathbb{N}$, počet faktorů $f \in \mathbb{N}$, regularizace $\lambda \in \mathbb{R}$
Output: $\mathbf{P} \in \mathbb{R}^{f \times m}$ a $\mathbf{Q} \in \mathbb{R}^{f \times n}$

initialization $\mathbf{P}$ ▷ většinou náhodná inicializace
initialization $\mathbf{Q}$
for _ **from** 0 **to** l **do** ▷ lze použít pravidlo konvergence místo konečného l
  for u **from** 0 **to** m **do**
    $\mathbf{p}_u \leftarrow (\mathbf{Q}\mathbf{P}^T + \lambda\mathbf{I})^{-1}\mathbf{Q}\mathbf{r}_u^T$
  for i **from** 0 **to** n **do**
    $\mathbf{q}_i \leftarrow (\mathbf{P}\mathbf{P}^T + \lambda\mathbf{I})^{-1}\mathbf{P}\mathbf{r}_i^T$
return: P, Q

Obdobné je to i pro položky (nějaké položky mají rádi téměř všichni). Vychýlení je součtem globálního vychýlení a vychýlení pro jednotlivé vektory [5] [9]:

$$b_{u,i} = \mu + b_i^I + b_u^U,$$

kde $\mathbf{b}^I \in R^n$ a $\mathbf{b}^U \in R^m$ a $\mu \in R$. Vychýlení je potom složkou při výpočtu $\hat{r}_{u,i}$, kde výraz 1.11 je rozšířen:

$$\hat{r}_{u,i} = \mu + b_i^I + b_u^U + \mathbf{q}_i^T \mathbf{p}_u$$

Jak je možné vidět, predikovaný rating se skládá z globálního vychýlení a poté z vychýlení pro uživatele a pro položku. Problém je pak řešen hledáním minima následujícího výrazu [9] [5]:

$$\min_{\substack{\mathbf{P}\in\mathbb{R}^{f\times m}\\ \mathbf{Q}\in\mathbb{R}^{f\times n}\\ \mathbf{b}^I\in R^n\\ \mathbf{b}^U\in R^m\\ \mu\in R}} \sum_{\substack{u\in U\\ i\in I\\ r_{u,i}\neq ?}} (r_{u,i} - \mu - b_i^I - b_u^U - \mathbf{q}_i^T \mathbf{p}_u)^2 - \lambda(\| \mathbf{p}_u \|^2 + \| \mathbf{q}_i \|^2 + {b_i^I}^2 + {b_u^U}^2) \tag{1.30}$$

Z výrazu 1.30 je možné vytvořit příslušnou objektivní funkci, pro jejíž minimalizaci a získání pravidel pro aktualizaci hodnot můžeme vyjít z odvození SGD v 1.2.2 a ALS v 1.2.3. Tím získáme potřebná pravidla pro tyto metody, které poté zahrnují i vychýlení.

1.2.5 Maticová faktorizace pro datasety s implicitní zpětnou vazbou

Všechny představené metody předpokládaly jako vstup ratingovou matici složenou z explicitních ratingů. Na explicitním hodnocení bylo možné jednoduše ukázat principy maticové faktorizace, ale v praktických problémech máme

k dispozici právě většinou pouze implicitní zpětnou vazbu, která byla popsána v sekci 1.1.1.

V této podkapitole je představeno rozšíření MF na implicitní zpětnou vazbu. Strategie pro využití implicitní zpětné vazby byla představena v publikaci [10], kde autoři mnohonásobně rozšířili možnosti použití MF v doporučovacích systémech. Nejprve byly uvedeny nejčastější charakteristiky implicitní zpětné vazby, které byly již zmíněny v sekci 1.1.1 a patří mezi ně například fakt, že tato zpětná vazba není negativní.

Pro tento problém předpokládejme matici $\mathbf{R} \in \mathbb{R}^{m \times n}$ jako v 1.1.2.1, kde hodnoty představují například počet nákupů dané položky nebo počet zhlédnutí seriálu, kde je možné v procentech uvést i předčasné ukončení sledování. Rozšířením poté může být nějaká agregace více druhů implicitní zpětné vazby. Model představený v článku [10] přichází s přístupem, kde je z ratingové matice s implicitní zpětnou vazbou vytvořena matice preferencí, která je označena $\mathbf{Z} \in \{0,1\}^{m \times n}$ a obsahuje pouze binární hodnoty. V [10] je používáno značení $p_{u,i}$, ale to by v této práci kolidovalo s označením uživatelských latentních příznaků, a proto bude preference značena jako $z_{u,i}$:

$$z_{u,i} = \begin{cases} 1 & r_{u,i} \neq ? \\ 0 & r_{u,i} = ? \end{cases}$$

Jde tedy o označení alespoň jedné interakce uživatele u s položkou i. K této zavedené preferenci je v článku [10] zavedena matice spolehlivosti $\mathbf{C} \in \mathbb{R}^{+m \times n}$ a představuje spolehlivost pro každou hodnotu $z_{u,i}$ a je možno závést:

$$c_{u,i} = 1 + \alpha r_{u,i},$$

kde α je konstanta zvyšující hodnotu spolehlivosti. V publikaci je uvedeno, že $\alpha = 40$ je dobrá hodnota pro solidní výsledky. Tímto způsobem je zvyšována důvěra v položky $z_{u,i} = 1$.

Stále je cílem najít latentní příznaky $\mathbf{p}_u \in \mathbf{P}^{f \times m}$ a $\mathbf{q}_i \in \mathbf{Q}^{f \times n}$, kde preference je výsledkem násobení $z_{u,i} = \mathbf{q}_i^T \mathbf{p}_u$. Tento přístup je podobný přístupu v MF pro explicitní zpětnou vazbu, která byla představena výše. Rozdíl je v tom, že je nutné vzít v úvahu matici spolehlivosti a také je nutné optimalizovat všechny možné dvojice u a i a ne jen ty, které odpovídají $z_{u,i} = 1$. To způsobuje problém, že není možné použít standartní algoritmy jako SGD nebo ALS, které byly popsány dříve. Nejprve je uvedena následující funkce, která je minimalizována [10]:

$$\min_{\substack{\mathbf{P} \in \mathbb{R}^{f \times m} \\ \mathbf{Q} \in \mathbb{R}^{f \times n}}} \sum_{\substack{u \in U \\ i \in I}} c_{u,i}(z_{u,i} - \mathbf{q}_i^T \mathbf{p}_u)^2 - \lambda \Big(\sum_{u \in U} \| \mathbf{p}_u \|^2 + \sum_{i \in I} \| \mathbf{q}_i \|^2 \Big). \tag{1.31}$$

Jak je zmíněno v této publikaci [10], λ je datově závislá a optimální hodnotu je nutné najít například použitím křížové validace. V článku [10]

představují následující proces optimalizace, pomocí kterého je možné problém řešit. Opět je využita myšlenka z ALS: matice $\mathbf{P}$ nebo $\mathbf{Q}$ je fixována a druhá naopak optimalizována. Ovšem ALS představené v části 1.2.3 bere nevyplněné hodnoty jako chybějící. Zde je nutné brát v úvahu, jak hustou ztrátovou funkci k minimalizaci, tak i míru spolehlivosti $\mathbf{C}$. Při zafixování matice $\mathbf{Q}$ můžeme pomocí derivací obdobně jako v 1.2.3 odvodit následující předpis pro aktualizaci každého uživatele nezávisle [10]:

$$\mathbf{p}_u = (\mathbf{Q}\mathbf{C}^u\mathbf{Q}^T + \lambda\mathbf{I})^{-1}\mathbf{Q}\mathbf{C}^u\mathbf{z}_u^T, \tag{1.32}$$

kde $\mathbf{z}_u \in \mathbb{R}^n$ je vektor preferencí uživatele a $\mathbf{C}^u \in \mathbb{R}^{n\times n}$ je diagonální matice důvěry definovaná jako $\mathbf{C}^u_{i,i} = c_{u,i}$. Výpočetně náročným místem pro tyto aktualizace je $\mathbf{Q}\mathbf{C}^u\mathbf{Q}^T$. V [10] řeší tento problém využitím následující úpravy:

$$\mathbf{Q}\mathbf{C}^u\mathbf{Q}^T = \mathbf{Q}\mathbf{Q}^T + \mathbf{Q}^T(\mathbf{C}^u - \mathbf{I})\mathbf{Q},$$

kde $\mathbf{Q}\mathbf{Q}^T \in \mathbb{R}^{f\times f}$ je nezávislé na u a může být jednoduše předpočítáno. $\mathbf{C}^u - \mathbf{I}$ má pouze n_u nenulových prvků, kde platí $z_{u,i} > 0$ a typicky n_u je velmi výrazně menší než n. To je stejné pro výpočet $\mathbf{C}^u\mathbf{z}_u^T$. Výpočetně nejnáročnější částí je poté provádění inverze $(\mathbf{Q}\mathbf{C}^u\mathbf{Q}^T + \lambda\mathbf{I})^{-1}$, která má složitost $O(f^3)$, ale hodnota f je typicky malá.

Obdobně je totéž provedeno pro aktualizace vektorů položek. Mějme fixovanou $\mathbf{P}$, vektor preferencí položky $\mathbf{z}_i \in \mathbb{R}^m$ a diagonální matici spolehlivosti $\mathbf{C}^i \in \mathbb{R}^{m\times m}$ definovanou jako $\mathbf{C}^i_{u,u} = c_{u,i}$. Aktualizace položek je poté po zderivování 1.31 podle $\mathbf{q}_i$ prováděna [10]:

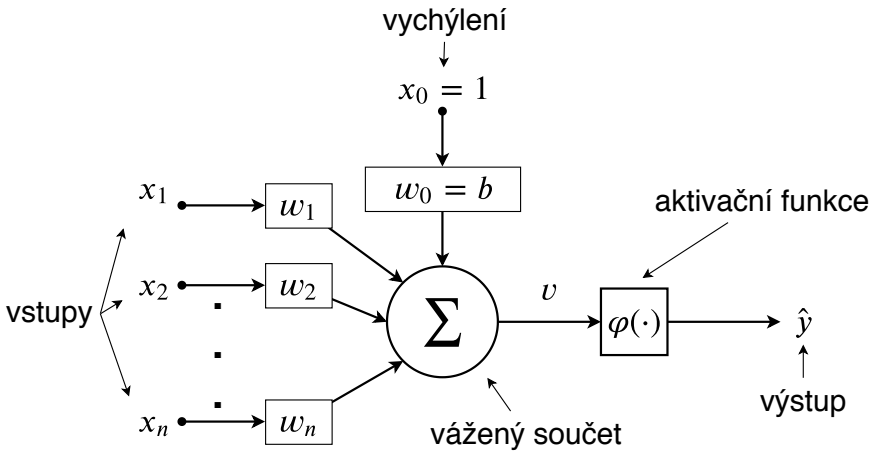
$$\mathbf{q}_i = (\mathbf{P}\mathbf{C}^i\mathbf{P}^T + \lambda\mathbf{I})^{-1}\mathbf{P}\mathbf{C}^i\mathbf{z}_i^T, \tag{1.33}$$

kde je $\mathbf{P}\mathbf{P}^T$ je předpočítáno a je použita stejná optimalizace jako pro uživatele.

S využitím této teorie byla implementována knihovna *Implicit* [24], která využívá tyto myšlenky z článku [10] a metody z publikace [22].

1.3 Umělé neuronové sítě (ANN)

V této kapitole jsou popsány umělé neuronové sítě. Jedná se o výpočetní model, který je inspirován biologickými neuronovými sítěmi, které je snaha napodobit, protože mozek skládající se z neuronů dokáže řešit komplexní úlohy v krátném čase a velice úspěšně. Stejně jako se učí lidský mozek, je nutné učit i neuronové sítě. V této kapitole je popsána struktura umělých neuronových sítí, po které následuje popis způsobu učení takové sítě. Poté je na NN navázáno rozšířením na rekurentní neuronové sítě a jejich rozšíření, které jsou v této práci využívány. Je popsáno jejich vybavování a trénování. Speciálně jsou pak popsány modifikace rekurentních neuronových sítí LSTM a GRU a jejich využití pro sekvenční modelování.



Obrázek 1.2: Popis umělého neuronu v umělé neuronové síti. [25]

Neuronová síť se skládá ze základních stavebních jednotek, kterými jsou umělé neurony. Umělé neurony představují zjednodušený model biologických neuronů, ze kterých pochází i inspirace pro umělé neurony, obzláště u principu generování a šíření elektrických impulsů. Každý neuron tedy představuje výpočetní buňku, jak je zobrazeno na obrázku 1.2. Neuronová buňka má vstupy $x_1, \ldots, x_n$, které mohou představovat vstup z jiného neuronu, nějaký vstup ze senzoru nebo přímo číselná data. Poté zde jsou ke vstupům odpovídající váhy $w_1, \ldots, w_n$. Další trochu odlišný vstup je $x_0 = 1$, který je využit s váhou $w_{k,0}$ jako vychýlení (angl. bias). Vážený vstup do neuronu je zapsán [25]:

$$v = \sum_{j=0}^{n} w_j x_j, \tag{1.34}$$

který je možné zapsat vektorově následovně:

$$v = \mathbf{wx}$$

Výstup neuronu $\hat{y}$ je výsledek po aplikování aktivační funkce na vážený vstup:

$$\hat{y} = \varphi(v)$$
$$\hat{y} = \varphi(\mathbf{wx}).$$

Nejjednodušším modelem neuronu je perceptron, který představuje jednoduchý binární klasifikátor a aktivační funkci je možné podle [25] zapsat následujícím způsobem:

$$\varphi(v) = \begin{cases} 1 & \text{pokud } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \text{pokud } \mathbf{w} \cdot \mathbf{x} < 0 \end{cases}.$$

Základní učící algoritmus je popsán v článku [26] – jedná se o rotaci vektoru v prostoru. Algoritmus má několik problémů, které jsou vyřešeny v jiných algoritmech. Tento algoritmus využívající rotace vektorů skončí pouze v případě lineárně separabilního problému. V opačném případě nikdy neskončí [26].

Tento způsob není kvůli konečnosti a nalezení optima pro neseparabilní třídy hojně využíván, ale lze si na něm dobře představit funkci jednoho neuronu v modelu. Pro trénování jsou využívány algoritmy založené na počítání gradientu. Pro učení pomocí gradientu je nutné definovat chybu mezi hodnotou ve výstupní vrstvě a požadovaným výstupem. Chyba bude optimalizována a cílem bude dosáhnout její co nejnižší hodnoty. Pro začátek je možné uvažovat jednoduchou lineární aktivační funkci a ztrátovou funkci jako sumu kvadratických odchylek:

$$F(\mathbf{w}) = \frac{1}{2}\sum_{i=0}^{m}(y_i - \hat{y}_i)^2,$$

kde m je počet trénovacích vzorků, $\hat{y}_i \in \mathbb{R}$ je hodnota výstupu z perceptronu a y_i je správná hodnota výstupu. Velikost kroku nebo učící faktor je roven γ. Cílem učení tedy budou kroky proti směru gradientu:

$$\Delta\mathbf{w} = -\gamma\nabla F(\mathbf{w}),$$

kde spočítáme parciální derivace pro každou váhu ve vektoru:

$$\Delta w_j = -\gamma\frac{\partial F}{w_j}. \tag{1.35}$$

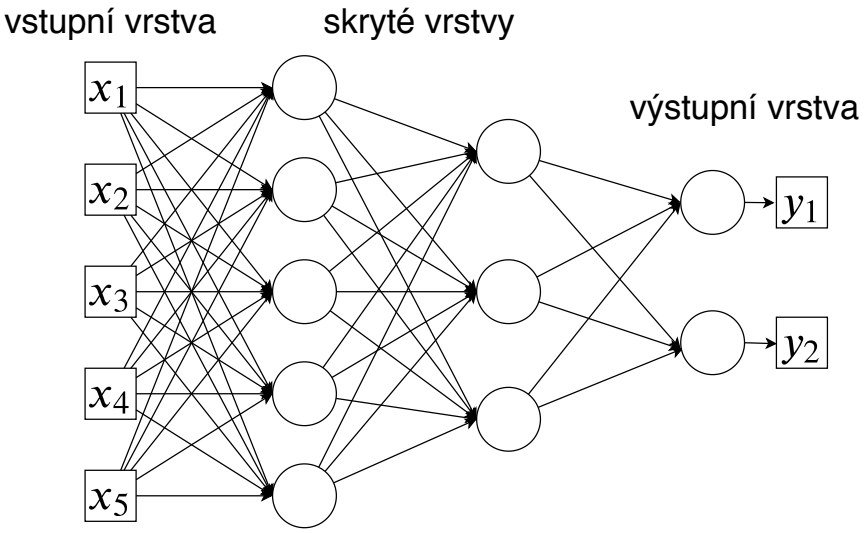
Chybovou funkci můžeme zderivovat následujícím způsobem:

$$\begin{aligned}\frac{\partial F}{\partial w_j} &= \frac{1}{2}\sum_{i=0}^{m}\frac{\partial F}{w_j}(y_i - \hat{y}_i)^2 \\ &= \frac{1}{2}\sum_{i=0}^{m}2(y_i - \hat{y}_i)\frac{\partial F}{\partial w_j}(y_i - \sum_{j=0}^{n}w_j x_{i,j}) \\ &= \sum_{i=0}^{m}(y_i - \hat{y}_i)(-x_{i,j}).\end{aligned} \tag{1.36}$$

Změny vah pak při dosazení do 1.35 dostaneme jako:

$$\Delta w_j = \gamma\sum_{i=0}^{m}(y_i - \hat{y}_i)x_{i,j}. \tag{1.37}$$

Pro učící algoritmus jsou tedy vhodné a hojně využívané funkce, pro které existuje první derivace v každém jejich bodě. První používaná funkce se nazývá logistická funkce [27]:



Obrázek 1.3: Architektura neuronové sítě. Obrázek byl vytvořen podle předlohy v publikaci [25]

$$\varphi(v) = \frac{1}{1 + e^{-\beta x}}, \tag{1.38}$$

kde speciálním případem je $\beta = 1$. Taková funkce je nazývá sigmoidou. β parametr určuje sklon funkce. Další používanou funkcí je hyperbolický tangens [27]:

$$\varphi(v) = \frac{1 - e^{-\beta x}}{1 + e^{-\beta x}}, \tag{1.39}$$

generující hodnoty v intervalu $[-1, 1]$. Další možnou funkcí může být například lineární funkce, která byla použita na učení neuronu výše, protože pro ni existuje jednoduchá první derivace v každém jejím bodě.

Jednotlivé neurony je možné skládat do neuronových sítí, které představují orientovaný graf, jak je zobrazeno na obrázku 1.3. Výstup z jednoho výše zmíněného neuronu může být jednoduše napojený na vstup jiného neuronu. Každý neuron má svou vlastní aktivační funkci.

Taková síť je pak většinou uspořádána do vrstev a vždy se skládá minimálně ze vstupní a výstupní vrstvy. Dále může obsahovat libovolný počet skrytých vrstev. Každá vrstva je vytvořena z libovolného počtu neuronů a neurony v těchto vrstvách jsou většinou propojeny každý s každým.

Umělé neuronové sítě jsou použity k modelování složitých vztahů mezi vstupem a výstupem. Jak je po sestavení modelu zřejmé, nachází se zde mnoho vah, které jsou optimalizovány pro nalezení vztahu mezi vstupem a výstupem.

Jak již bylo vidět u ukázky principu učení neuronu, učení typicky probíhá s učitelem – jsou známy vstupy a k nim příslušné správné výstupy. Aktualizaci vah, podle výrazu 1.37, je možné použít pouze u neuronových sítí s jednou vrstvou. Je ale také nutné učit neurony ve skrytých vrstvách, což je prováděno pomocí algoritmu zpětné propagace chyby (angl. back propagation). Tento algoritmus rozšiřuje výše zmíněný postup přidáním mechanismu propagace chyby do skrytých vrstev.

Pro **algoritmus zpětného šíření chyby** (angl. back propagation, BP) předpokládejme dvojice trénovacích vzorků $[\mathbf{x}^{(i)}, \mathbf{y}^{(i)}]$, kde značení (i) představuje í-tý trénovací prvek a platí $\mathbf{x}^{(i)} \in \mathbf{X}$, kde $\mathbf{X} \in \mathbb{R}^{N_0 \times m}$ a $\mathbf{y}^{(i)} \in \mathbf{Y}$, kde $\mathbf{Y} \in \mathbb{R}^{N_L \times m}$. Podle článku [27] mějme L vrstev, kde L je výstupní vrstva a index 0 představuje vstupní vrstvu. Vrstva l má N_l neuronů a aktivační funkci f^l. Funkce musí být diferencovatelná, ale nemusí být lineární. Nejčastěji používané funkce jsou 1.38 a 1.39. Počet trénovacích vzorků je roven m, N_0 je počet neuronů ve vstupní vrstvě a N_L je počet neuronů ve výstupní vrstvě. Popis algoritmu zpětné propagace chyby v této práci vychází z publikace [27], kde je algoritmus detailně popsán i s možnými optimalizacemi. V práci je uvedena základní varianta algoritmu, která je v 1.3.1 rozšířena na rekurentní neuronové sítě.

Vstup do aktivační funkce neuronu z 1.34 spočítáme:

$$v_j^{(l)} = \sum_{i=1}^{N_{l-1}} (\hat{y}_i^{(l-1)} w_{i,j}^{(l)}) + b_j^{(l)}, \tag{1.40}$$

kde $\hat{y}_i^{(l-1)}$ je výstup z neuronu v předchozí vrstvě a $b_j^{(l)}$ je vychýlení neuronu j ve vrstvě l a jedná se o váhu s indexem 0 z ukázky pro jeden neuron. Zde použiji zvláštní značení pro vychýlení, obdobně jako v publikaci [27] pro zřetelnější popis algoritmu i aktualizaci vychýlení. Jak je vidět z výrazu 1.40, váhy je možné zapsat do matice vah $w_{i,j} \in \mathbf{W}$, kde $\mathbf{W} \in \mathbb{R}^{N_{l-1}, N_l}$.

Výstup z neuronu nebo-li také aktivaci neuronu ve vrstvě l spočítáme pouze aplikací aktivační funkce $f^{(l)}$ na váženou sumu v 1.40:

$$\begin{aligned} \hat{y}_j^{(l)} &= f^{(l)}(v_j^{(l)}) \\ \hat{y}_j^{(l)} &= f^{(l)}\Big(\sum_{i=1}^{N_{l-1}} (\hat{y}_i^{(l-1)} w_{i,j}^{(l)}) + b_j^{(l)} \Big), \end{aligned} \tag{1.41}$$

kde aktivace neuronu $\hat{\mathbf{y}}^{(\mathbf{0})} = \mathbf{x}$. Pro první vrstvu je jako výstup předchozí vrstvy brán přímo vstup. U poslední vrstvy L je hodnota aktivace přímo hodnota výstupu. Jako učící algoritmus je použit opět gradientní sestup, kde jsou aktualizované jednotlivé váhy. Pro porovnání výstupu v poslední vrstvě s požadovaným výstupem použiji opět kvadratickou chybu $E = (\mathbf{y} - \hat{\mathbf{y}})^2$, kterou se budu snažit minimalizovat. Jednotlivé aktualizace vah mohou být

provedeny následujícím způsobem:

$$\begin{aligned} \Delta w_{i,j}^{(l)} &= -\gamma \frac{\partial E}{\partial w_{i,j}^{(l)}} \\ \Delta b_j^{(l)} &= -\gamma \frac{\partial E}{\partial b_j^{(l)}} \end{aligned} \tag{1.42}$$

Pro spočítání výše zmíněných derivací je nutné rozepsat výpočet chyby pro zjištění, jak je E závislá na vahách a vychýlení. Chyba ve výstupní vrstvě je přímo závislá na výstupu $\hat{\mathbf{y}}$ a ten na vstupu $\mathbf{v}^{(l)}$ s využitím aktivační funkce. Je nutné spočítat parciální derivace podle vah, kde s využitím řetězového pravidla pro derivace dostaneme [27]:

$$\frac{\partial E}{\partial w_{i,j}^{(l)}} = \sum_{k=1}^{N_l} \frac{\partial E}{\partial v_k^{(l)}} \frac{\partial v_k^{(l)}}{\partial w_{i,j}^{(l)}}, \tag{1.43}$$

kde první člen ze součtu je nazýván chybou:

$$\frac{\partial E}{\partial v_k^{(l)}} = \delta_k^{(l)} \tag{1.44}$$

a $k = 1, \ldots, N_l$. Druhý člen je poté:

$$\frac{\partial v_k^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial}{\partial w_{i,j}^{(l)}} \Big(\sum_{i=1}^{N_{l-1}} (\hat{y}_i^{(l-1)} w_{i,k}^{(l)}) + b_k^{(l)} \Big) = \hat{y}_i^{(l-1)}. \tag{1.45}$$

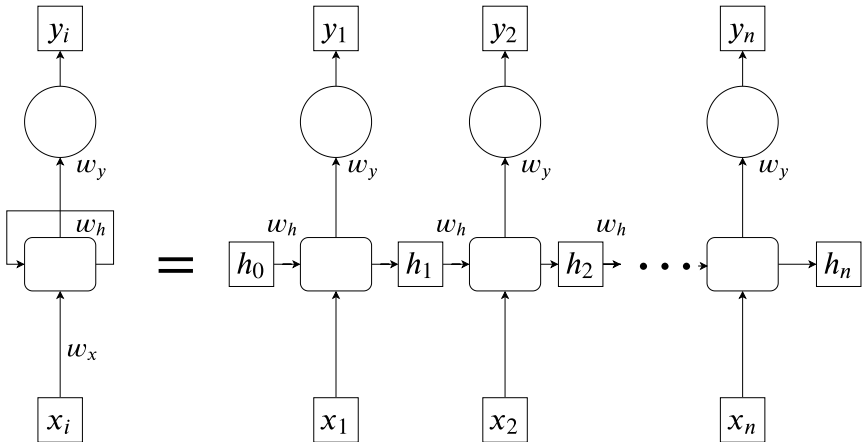
Společně pak dostaneme:

$$\begin{aligned} \frac{\partial E}{\partial w_{i,j}^{(l)}} &= \hat{y}_i^{(l-1)} \delta_k^{(l)} \\ \frac{\partial E}{\partial b_j^{(l)}} &= \delta_k^{(l)} \end{aligned} \tag{1.46}$$

a z toho je možné vytvořit pravidla pro aktualizaci hodnot:

$$\begin{aligned} w_{i,j}^{(l)} &\leftarrow w_{i,j}^{(l)} - \gamma \hat{y}_i^{(l-1)} \delta_j^{(l)} \\ b_j^{(l)} &\leftarrow b_j^{(l)} - \gamma \delta_j^{(l)}. \end{aligned} \tag{1.47}$$

Z aktualizace hodnot výše je vidět, že je nutné spočítat $\delta_j^{(l)}$ pro $l = 1, \ldots, L-1$ a L. Pro výstupní vrstvu L bude δ jiná než pro skryté vrstvy, protože je ji možné spočítat rovnou vůči výstupu $\mathbf{y}$. Pro spočítání $\delta_j^{(l)}$ vyjdeme z výrazu 1.44 a ze závislosti E na $v_j^{(l)}$. Můžeme vidět závislost $v_j^{(l)}$ na $v_i^{(l-1)}$ pro $i = 1, \ldots, N_{l-1}$. Vstup do vrstvy l závisí na hodnotě aktivace ve



Obrázek 1.4: Na obrázku je ukázka rekurentní neuronové sítě s jedním skrytým a výstupním neuronem a vstupní vrstvou velikosti 1. Vpravo je ukázka rozbalení této sítě v čase. Obrázek je vytvořen na základě obrázku v publikaci [28]

vrstvě $l-1$ a dále rekurzivně. Chybu $\delta_j^{(l-1)}$ pro $j = 1, \ldots, N_L$ opět s aplikací řetězového pravidla pro derivaci spočítáme:

$$\begin{aligned}\delta_j^{(l-1)} = \frac{\partial E}{\partial v_j^{(l-1)}} &= \sum_{i=1}^{N_l} \frac{\partial E}{\partial v_i^{(l)}} \frac{\partial v_i^{(l)}}{\partial v_j^{(l-1)}} \\ &= \sum_{i=1}^{N_l} \frac{\partial E}{\partial v_i^{(l)}} \frac{\partial}{\partial v_j^{(l-1)}} \left(\sum_{k=1}^{N_{l-1}} \left(f^{(l-1)}(v_k^{(l-1)}) w_{k,i}^{(l)} \right) + b_i^{(l)} \right) \\ &= \dot{f}^{(l-1)}(v_j^{(l-1)}) \sum_{i=1}^{N_l} w_{j,i}^{(l)} \delta_i^{(l)}\end{aligned} \tag{1.48}$$

Chyba neuronu v predikci ve vrstvě $l-1$ závisí na chybě každého neuronu ve vrstvě l. Toto zpětné šíření chyby je prováděno rekurzivně a odtud je také název metody zpětná propagace. Pro poslední vrstvu vyjdeme přímo z chybové funkce a můžeme pro $j = 1, \ldots, N_L$ zapsat jako:

$$\delta_j^{(L)} = 2(\hat{y}_j^{(L)} - y_j) \dot{f}^{(L)}(v_j^{(L)}) \tag{1.49}$$

Vybavování probíhá tedy od vstupní vrstvy. Nejprve probíhá vybavení a je spočítána chyba, která je šířena zpět sítí, kde jsou aktualizovány její váhy. Postup můžeme zapsat tak, jak je ukázáno v algoritmu 3.

1.3.1 Rekurentní neuronové sítě (RNN)

V této sekci jsou posány rekurentní neuronové sítě. Na začátku je popsána základní struktura RNN, zmíněny její výhody a problémy. V navazující části jsou představeny modifikace jako LSTM a GRU.

Algoritmus 3 Zpětné šíření chyby

Input: matice tréninkových dat $\mathbf{X}$, matice příslušných výstupů $\mathbf{Y}$ a učící faktor $\gamma \in \mathbb{R}$

Output: matice vah $\mathbf{W}^{(l)}$ a vychýlení $\mathbf{b}^{(l)}$ pro $l = 1, \ldots, L$

initialization $\mathbf{W}^{(l)}$ pro $l = 1, \ldots, L$ ▷ náhodná inicializace na malé hodnoty
initialization $\mathbf{b}^{(l)}$ pro $l = 1, \ldots, L$ ▷ náhodná inicializace na malé hodnoty
repeat
  $\hat{\mathbf{y}}_0 = x^{(i)}$ ▷ náhodně vybraný trénovací vzorek $x^{(i)} \in \mathbf{X}$
  for l **from** 1 **to** L **do**
    for j **from** 1 **to** N_l **do**
      $v_j^{(l)} = \hat{y}_j^{(l)} \mathbf{w}_j^{(l)} + b_j^{(l)}$
      $\hat{y}_j^{(l)} = f^{(l)}(v_j^{(l)})$
  for j **from** 1 **to** N_l **do**
    $\delta_j^{(L)} = 2(\hat{y}_j^{(L)} - y_j^{(i)}) \dot{f}^{(L)}(v_j^{(L)})$ ▷ $y^{(i)} \in \mathbf{Y}$
  for l **from** 1 **to** $L - 1$ **do**
    for j **from** 1 **to** N_l **do**
      $\delta_j^{(l)} = \dot{f}^{(l)}(v_j^{(l)}) \sum_{i=1}^{N_{l+1}} w_{j,i}^{(l+1)} \delta_i^{(l+1)}$
  for l **from** 1 **to** L **do**
    for j **from** 1 **to** N_l **do**
      $b_j^{(l)} \leftarrow b_j^{(l)} - \gamma \delta_k^{(l)}$
      for i **from** 1 **to** N_{l-1} **do**
        $w_{i,j}^{(l)} \leftarrow w_{i,j}^{(l)} - \gamma \hat{y}_i^{(l-1)} \delta_j^{(l)}$

until není splněno pravidlo konvergence
return: $\mathbf{W}^{(l)}$ a vychýlení $\mathbf{b}^{(l)}$ pro $l = 1, \ldots, L$

RNN jsou sítě, které mají alespoň jedno zpětné spojení. Tedy oproti dopředným neuronovým sítím, které mají topologické uspořádání, jak je vidět na obrázku 1.3, mají rekurentní sítě cykly v grafu, které síti umožňují uchovat nějakou informaci. Model rekurentní neuronové sítě je možné vidět na obrázku 1.4, kde je jednoduchý model RNN a ukázka rozbalení sítě v jednotlivých časových krocích. Zaoblené čtverce představují rekurentní neurony ve skryté vrstvě. Je patrné, že stejně jako jsou váhy nastaveny pro dopředné vazby, budou obdobně nastaveny váhy i pro rekurentní vazby, aby bylo možné tyto váhy v čase učit. Naučené váhy se nemění v časových krocích sítě.

Z obrázku 1.4 a myšlenky RNN je dále patrné jejich využití pro zpracování různých sekvencí a opakujících se vstupů stejného formátu, které mezi sebou mohou mít nějaké závislosti. Může se jednat například o analýzu řeči, kde

nejde pouze o jednotlivá slova, ale také o jejich pořadí. Dalším příkladem je předpověď počasí, která vychází z předchozích meterologických údajů v určitých časových okamžicích. V případě doporučovacích systémů se jedná o jednotlivé akce uživatele na nějaké doméně a může být cílem tyto akce predikovat.

Z chování uživatelů může být užitečné předpovědět, jakou další položku si zobrazují po sekvenci jiných položek nebo kdy uživatel bude chtít znovu zkonzumovat již viděnou položku. Přesně takové položky je vhodné uživateli doporučit. V těchto doménách, kde se vyskytují sekvence dat, jsou RNN velmi silné a díky vnitřnímu stavu dokáží zachytit mnoho závislostí oproti dopředným neuronovým sítím.

Pomocí RNN je možné modelovat sekvence. Jako vstupní data do sítě slouží nějaká vhodně zakódovaná sekvence událostí, která je sítí využita pro různé úlohy. V případě obdržení konkrétního kroku je možné měřit úspěšnost, upravit sekvenci pro další predikci a využít nová data k učení.

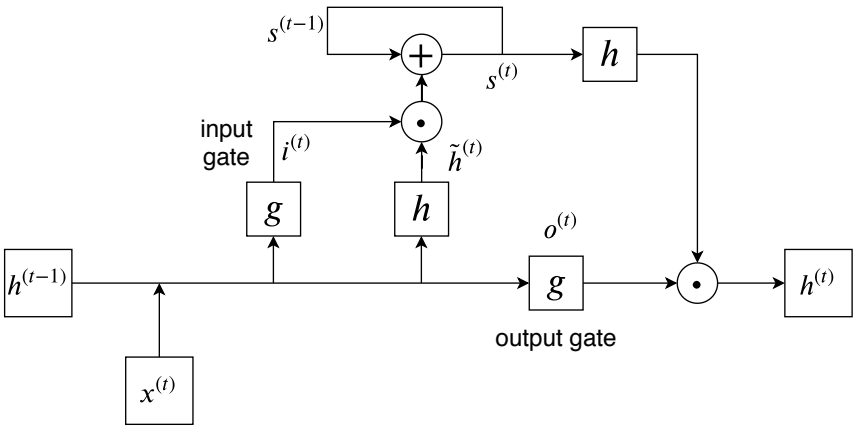
Jak je patrné, vstupem do sítě je sekvence interakcí, která je postupně vkládána jako $x_1, \ldots, x_n$ jak je patrné z obrázku 1.3. S každou položkou sekvence je aktualizována hodnota v rekurentní vazbě. Tyto rekurentní výstupy nejsou v těchto sítích přímo učeny vůči správné hodnotě a jejich hodnota se v čase mění. Dále se na rozdíl oproti dopředným sítím nacházejí váhy pro zpětné vazby.

Je zřejmé, že na učení nejde použít standartní algoritmus zpětné propagace chyby, ale je využíván algoritmus, který uvažuje zpětné šíření chyby v čase (angl. backpropagation through time, BPTT), protože i vliv chyby se mění v čase. Algoritmus je popsán v podsekci 1.3.2. Dalším problémem RNN je mizející gradient (angl. vanishing gradient) nebo explodující gradient (angl. exploding gradients), které jsou blíže popsány a rozebrány v publikaci [29]. Tyto problémy jsou ve velké míře řešeny pomocí složitějších rekurentních struktur, které se chovají jako jedna rekurentní buňka. Jedná se například o LSTM a GRU, které jsou popsány v následující částech.

Základní RNN odpovídající obrázku 1.4 můžeme zapsat [25]:

$$\begin{aligned} \mathbf{h}^{(t)} &= h\Big(\mathbf{b}_h + \mathbf{x}^{(t)}\mathbf{W}_h + \mathbf{h}^{(t-1)}\mathbf{U}_h\Big) \\ \hat{\mathbf{y}}^{(t)} &= g\Big(\mathbf{b} + \mathbf{h}^{(t)}\mathbf{W}\Big), \end{aligned} \tag{1.50}$$

kde h a g jsou aktivační funkce, pro které většinou platí $h = \tanh$ a g je sigmoida. Časové kroky jsou reprezentovány pomocí t. Vektor vnitřních stavů je $\mathbf{h}^{(t)}$, $\hat{\mathbf{y}}^{(t)}$ odpovídá vektoru výstupních hodnot, $\mathbf{x}^{(t)}$ vektoru vstupních hodnot, $\mathbf{W}$ představuje matici vah pro vstup, kde index h odkazuje na váhy k vnitřním stavům, stejně $\mathbf{U}$ představuje matici vah pro vnitřní stavy a $\mathbf{b}$ jsou obdobně vektory vychýlení.



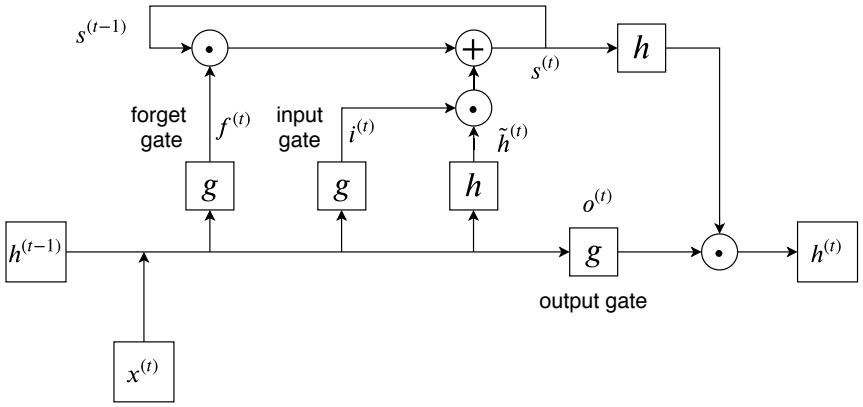
Obrázek 1.5: LSTM buňka s uzávěrkovým mechanismem pro řešení problému mizejícího a explodujícího gradientu při učení RNN [30]

1.3.1.1 LSTM

LSTM bylo představeno v roce 1997 v publikaci [30]. Jedná se o rozšíření klasické RNN pomocí bran, které se chovají jako uzávěrkový mechanismus. LSTM pomocí těchto bran řeší přístup k vnitřnímu stavu neuronu. S pomocí těchto bran je řešen problém explodujícího a mizejícího gradientu. Je umožněno, že gradient není změněn při učení buňky, pokud je aktivní uzávěr (brána). V zmíněné publikaci je také popsán vznik názvu *long short-term memory* (dlouhodobá a krátkodobá paměť), kde dlouhodobá paměť by měla odkazovat na naučené váhy a krátkdobá pamět odkazuje na hodnoty stavů uzávěrkových mechanismů, které se mění každým krokem v čase.

V článku [30] byl představen model, který je znázorněn na obrázku 1.5. Díky branám se LSTM může naučit přemostit časové intervaly, které mají i přes 1000 kroků v případě zašumělých nekomprimovatelných vstupních sekvencí, aniž by došlo ke ztrátě možností krátkodobé paměti [30]. Na obrázku 1.5 je vidět použití dvou aktivačních funkcí g a h, kde g ve většině případů představuje sigmoidu a je použitá pro uzávěrkový mechanismus. Sigmoida má hodnoty v intervalu $[0, 1]$ a řídí tok informace. Jako aktivační funkce buňky mimo uzávěrkový mechanismus je použit hyperbolický tangens, který má hodnoty v intervalu $[-1, 1]$ a s jeho pomocí je také předcházeno problému mizejícího gradientu, kde propagované hodnoty nepadají tak rychle k nule.

LSTM buňka, která je dnes používána, je zobrazena na obrázku 1.6. Původní LSTM bylo rozšířeno přidáním uzávěru pro vnitřní stav, který je nyní možné zapomenout. Tato brána je nazvána forget gate. Rozšíření bylo představeno v publikaci [31] o rok později. Vybavování rozšířené LSTM buňky lze zapsat následujícím způsobem s využitím publikace [31] a použitím vektorové



Obrázek 1.6: LSTM buňka s přidaným uzávěrem pro vnitřní stav [31]

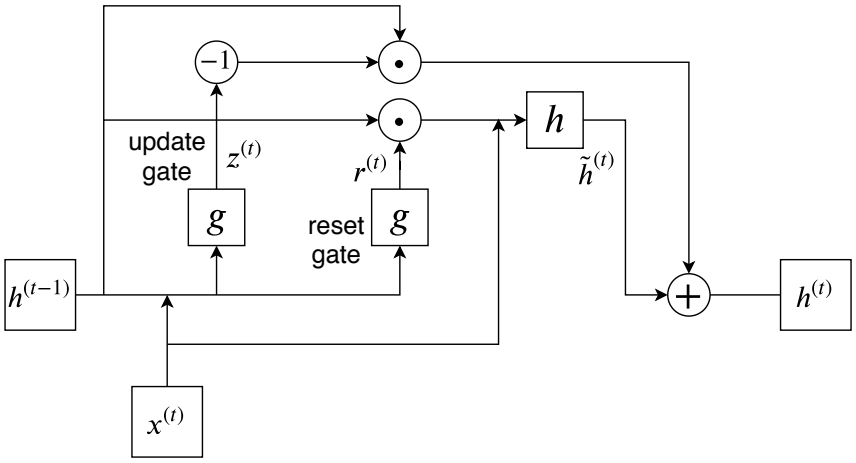
notace z článku [32]:

$$
\begin{aligned}
i_j^{(t)} &= g\left(b_{i:j} + [\mathbf{x}^{(t)}\mathbf{W}_i]_{:j} + [\mathbf{h}^{(t-1)}\mathbf{U}_i]_{:j}\right) \\
f_j^{(t)} &= g\left(b_{f:j} + [\mathbf{x}^{(t)}\mathbf{W}_f]_{:j} + [\mathbf{h}^{(t-1)}\mathbf{U}_f]_{:j}\right) \\
o_j^{(t)} &= g\left(b_{o:j} + [\mathbf{x}^{(t)}\mathbf{W}_o]_{:j} + [\mathbf{h}^{(t-1)}\mathbf{U}_o]_{:j}\right) \\
\tilde{h}_j^{(t)} &= h\left(b_{:j} + [\mathbf{x}^{(t)}\mathbf{W}]_{:j} + [\mathbf{h}^{(t-1)}\mathbf{U}]_{:j}\right) \\
s_j^{(t)} &= f_j^{(t)} \cdot s_j^{(t-1)} + i_j^{(t)} \cdot \tilde{h}_j^{(t)} \\
h_j^{(t)} &= h(s_j^{(t)}) \cdot o_j^{(t)},
\end{aligned}
\tag{1.51}
$$

kde $\mathbf{x}^{(t)}$ je vstup (buď z předchozí skryté vrstvy nebo přímo ze vstupní vrstvy) v čase t, $\mathbf{h}^{(t-1)}$ je výstupní hodnota z buňky j v čase $t-1$. $\mathbf{W}$ jsou matice vah a index odkazuje na matici vah pro konkrétní bránu, tedy i pro input gate, f pro forget gate a o pro output gate, stejně jak je znázorněno na obrázku 1.6. Pro rozměr váhových matic platí *velikost vstupu z předchozí vrstvy* $\times$ *počet neuronů (buněk) v aktuální vrstvě*. $\mathbf{U}$ jsou čtvercové matice vah pro rekuretní stav LSTM buňky se stejnými indexy pro brány. Stejně $\mathbf{b}$ představují vektory pro vychýlení. Pro aktivační funce platí g je sigmoida a $h = \tanh$, ale prakticky můžou být nahrazeny libovolnou jinou aktivační funkcí, ale nemusí dosáhnout stejné kvality. Tyto LSTM buňky (neurony) budou použity při experimentech a porovnány s GRU, které jsou představeny v následující podsekci.

1.3.1.2 GRU

GRU bylo představeno v roce 2014 v publikaci [32]. GRU je znázorněno na obrázku 1.7. Oproti LSTM neobsahuje output gate. Uzávěrkový mechanismus



Obrázek 1.7: GRU buňka s uzávěrkovým mechanismem [32]

je zprostředkován aktualizační a resetovací bránou. Jak je vidět na obrázku 1.7, GRU obsahuje méně parametrů, ale není schopné překlenout a pracovat s tak dlouhými sekvencemi jako LSTM. GRU se tedy hodí spíš na menší datasety nebo jako alternativa k LSTM, protože je rychleji trénováno a vyhodnocováno právě díky menšímu počtu parametrů. Stejně jako pro LSTM můžeme zapsat rovnice pro vyhodnocení následujícím způsobem [32]:

$$\begin{aligned} r_j^{(t)} &= g\Big(b_{r:j} + [\mathbf{x}^{(t)}\mathbf{W}_r]_{:j} + [\mathbf{h}^{(t-1)}\mathbf{U}_r]_{:j}\Big) \\ z_j^{(t)} &= g\Big(b_{z:j} + [\mathbf{x}^{(t)}\mathbf{W}_z]_{:j} + [\mathbf{h}^{(t-1)}\mathbf{U}_z]_{:j}\Big) \\ \tilde{h}_j^{(t)} &= h\Big(b_{:j} + [\mathbf{x}^{(t)}\mathbf{W}]_{:j} + [(\mathbf{r}^{(t)} \odot \mathbf{h}^{(t-1)})\mathbf{U}]_{:j}\Big) \\ h_j^{(t)} &= z_j^{(t)} \cdot h_j^{(t-1)} + (1 - z_j^{(t)}) \cdot \tilde{h}_j^{(t)}, \end{aligned} \tag{1.52}$$

kde váhové matice $\mathbf{W}$ a $\mathbf{U}$ mají podobné indexy, jak již bylo popsáno v podsekci 1.3.1.1 o LSTM. Odpovídající indexy jsou z pro aktualizační bránu a r pro resetovací bránu, obdobně pro vektory vychýlení. Mizejícímu gradientu je v GRU předcházeno tím, že se samo učí jaké množství informace může pustit z předchozího kroku. To je řízeno aktualizační bránou $z_j^{(t)}$. Pomocí resetovací brány je řízeno množství informace, která má být zapomenuta. Modely založené na GRU a LSTM jsou porovnány dále v této práci.

1.3.2 Trénování RNN

Gradientní sestup popsaný v kapitole 1.3.1 je základní metodou pro trénování dopředné neuronové sítě. Tato metoda ovšem nejde použít pro trénování reku-

rentní neuronové sítě, ale je nutné použít například algoritmus zpětného šíření chyby v čase (BPTT) nebo trénování neuronové sítě v reálném čase (angl. real time recurrent learning, RTRL).

1.3.2.1 Algoritmus zpětného šíření chyby v čase (BPTT)

Algoritmus BPTT využívá rozbalení rekurentní vazby, jak je ukázano na obrázku 1.4. Na rozbalené RNN je proveden téměř totožný postup jako byl ukázán v algoritmu BP v sekci 1.3, ze kterého vycházím a je v něm postup rozepsaný podrobněji. V této podsekci je uvedena část změn, které jsou jiné a podstatné oproti BP. Práce vychází z článku [33], kde je postup rozebraný podrobněji. Značení je použito tak, aby bylo konzistentní se zbytkem práce. Aktualizaci vah je možné zapsat:

$$\Delta \mathbf{W}^{(l)} = -\gamma \frac{\partial E^{(t)}}{\partial \mathbf{W}^{(l)}} \tag{1.53}$$

$$\Delta \mathbf{b}^{(l)} = -\gamma \frac{\partial E^{(t)}}{\partial \mathbf{b}^{(l)}} \tag{1.54}$$

kde (l) značí index rekurentní vrstvy a (t) značí čas. $E^{(t)}$ je hodnota chyby v časovém kroku t. Obdobně lze získat aktualizaci vah pro $\mathbf{W}_h$, $\mathbf{U}_h$ a $\mathbf{b}_h$.

Je tedy nutné spočítat derivace chybové funkce $E^{(t)}$ podle $\mathbf{W}_h$, $\mathbf{U}_h$, $\mathbf{b}_h$, $\mathbf{b}$ a $\mathbf{W}$. Ztrátovou funkci v RNN můžeme zapsat jako $E^{(t)} = \sum_{i=0}^{t} E^{(i)}(\hat{\mathbf{y}}, \mathbf{y})$. Jedná se tedy o sumu přes všechny časové kroky a odpovídá rozbalení sítě z obrázku 1.4. Nejprve je uvedena derivaci podle $\mathbf{W}$, která je rozepsána pomocí řetězového pravidla a vyjde ze závislostí $\mathbf{W}$ ve výrazech 1.50 a vlivu vah na chybu E (obdobně lze zapsat i pro vychýlení $\mathbf{b}$):

$$\frac{\partial E^{(t)}}{\partial \mathbf{W}^{(l)}} = \frac{\partial E^{(t)}}{\partial \hat{\mathbf{y}}^{(t,l)}} \frac{\partial \hat{\mathbf{y}}^{(t,l)}}{\partial \mathbf{s}^{(t,l)} \mathbf{W}^{(l)}} \frac{\partial \mathbf{s}^{(t,l)} \mathbf{W}^{(l)}}{\partial \mathbf{W}^{(l)}}. \tag{1.55}$$

Jak je možné vidět z výrazu 1.55, derivace je téměř totožná s verzí ve standartním algoritmu BP. Změna ovšem nastane pro derivaci podle $\mathbf{W}_h$, $\mathbf{U}_h$ a $\mathbf{b}_h$, protože z výrazu 1.50 je vidět rekurzivní závislost, kde $h^{(t)}$ zavisí na $h^{(t-1)}$ atd. . .Tuto závislost je nutné zapsat a je k tomu využita suma, která je použita i pro chybovou funkci uvedenou výše:

$$\frac{\partial E^{(t)}}{\partial \mathbf{W}_h^{(l)}} = \sum_{i=0}^{t} \frac{\partial E^{(t)}}{\partial \hat{\mathbf{y}}^{(t,l)}} \frac{\partial \hat{\mathbf{y}}^{(t,l)}}{\partial \mathbf{h}^{(t,l)}} \frac{\partial \mathbf{h}^{(t,l)}}{\partial \mathbf{h}^{(i,l)}} \frac{\partial \mathbf{h}^{(i,l)}}{\partial \mathbf{W}_h^{(l)}} \tag{1.56}$$

Obdobně lze zapsat pro $\mathbf{U}_h$ a $\mathbf{b}_h$. Po dopočítání derivací a vytvoření aktualizačních pravidel je možné postup zapsat do algoritmu stejně jako v případě BP v algoritmu 3. Klíčové rozšíření se tedy nachází přímo v součtu přes jednotlivé časové kroky. Jedná se o aplikaci BP na rozbalenou RNN.

Stejným způsobem je možné aplikovat výše zmíněný postup i na rovnice uvedené v sekci 1.3.1.1 pro LSTM a obdobně pro GRU, pouze s tím rozdílem, že je zde nutné provést derivace podle matic vah pro rovnice uzávěrkového mechanismu a vnitřního stavu. Obdobně je potřeba derivovat podle vektorů vychýlení, které existují také pro každou bránu a vnitřní stav.

1.3.2.2 Trénování neuronové sítě v reálném čase (RTRL)

V této metodě je prováděno učení vah v reálném čase, zatímco síť dále zpracovává signál. Metoda předpokládá plně propojenou rekurentní síť. Učení pomocí tohoto algoritmu není lokální, protože je nutné vzít v úvahu účinek změny na jednom místě a jeho vliv na vypočtené hodnoty na jiném místě. RTRL obsahuje více numerických výrazů, které je nutné spočítat a může být náročnější na výpočet než BPTT [25].

Cílem je stále minimalizovat chybovou funkci, která může být součtem okamžitých hodnot. RTRL je také založena na gradientním sestupu. Je použita stejná chybová funkce jako v případě BPTT, ale místo výpočtu přesné derivace pro výrazy 1.53 jsou použity její aktuální odhady [34]. RTRL je možné využít pro nepřetržité trénování bez restartu vnitřních stavů, zatímco BPTT je použito spíš při offline trénování na omezenou délku sekvencí, kdy je síť restartována po sekvenci položek a je možné v paměti provést rozbalení sítě. Tímto může velmi rychle růst paměťová náročnost s velikostí sekvence. Metoda RTRL je popsána v knize [25] a v článku [34], kde je kromě RTRL popsána také BPTT a jsou zdůrazněny společné části obou metod.

1.3.3 Modifikace a optimalizace RNN

Neuronové sítě je možné implementovat a používat podle výše zmíněných postupů, ale pravděpodobně nebude dosaženo tak kvalitních výsledků, proto se v implementacích používají různé modifikace a optimalizace, které pomáhají zpřesnit nebo zrychlit postup učení, či například zlepšit schopnosti generalizace. Nejpoužívanější z nich jsou uvedené v této kapitole a většina není použitelná pouze na RNN, ale také na dopředné neuronové sítě.

1.3.3.1 Dávková aktualizace vah (angl. batch updates)

První z optimalizací je dávkové učení, které se nazývá anglicky *mini batches* a je pod tímto názvem známo. Smyslem této optimalizace je, že nemusí být vždy ideální provádět korekci vah po jednom trénovacím vzorku, ale může být vhodné kumulovat změny pro nějakou dávku vzorků a tyto změny pak aplikovat.

Aplikací této metody dochází k vyhlazování změn parametrů a je snížen rozptyl při aktualizaci trénovaných parametrů. Čím nižší je tedy velikost jedné dávky (standartně se pohybuje mezi 32 a 256), tím je potřeba snížit učící faktor (angl. learning rate), tak aby na učení neměl takový vliv vysoký rozptyl

a naopak [35]. Při trénování po jednom vzorku může docházet k velkému vychýlení parametrů, což může snížit rychlost konvergence. Použití této metody může přinést také nějakou míru regularizace. Další výhodou je možnost paralelizace po jednotlivých dávkách. Tato metoda je v dnešní době velice populární a používána v kombinaci s jinými metodami optimalizace [35].

1.3.3.2 Dávková normalizace (angl. batch normalization)

Dávková normalizace byla představena v roce 2015 v článku [36]. Podle autorů článku tato technika přináší velké zrychlení, vyšší stabilitu a výkon pro trénování neuronových sítí. Myšlenka je taková, že za stejným účelem jako se normalizuje vstupní vrstva, jsou normalizovány všechny vrstvy. NN je rozdělena na vrstvy a na výstup z každé vrstvy se můžeme dívat jako na vstup do vstupní vrstvy a tento vstup normalizovat.

Cílem je tedy normalizovat vstup do vrstvy tak, aby měl nulovou střední hodnotu a směrodatnou odchylku rovnou 1. Na jednotlivé vrstvy nahlížíme jako na nezávislé neuronové sítě, kde výstup z jedné je normalizován a použit jako vstup do další vrstvy. Jak je zmíněno a popsáno v článku [36], tato technika má jít proti internímu kovariátovému posunu. Technika by měla pomoci při inicializaci vah, kde by inicializované váhy neměly mít takový vliv u hlubokých neuronových sítí. Dále by technika měla usnadnit trénování právě hlubokých neuronových sítí.

Dávková normalizace vstupu do sítě pomáhá, ale jsou různé názory, proč tato technika funguje. Autoři této techniky popisují, že jde proti vnitřnímu kovariátovému posunu, ale v publikaci [37] je ukázáno, že metoda nesnižuje tento posun, ale místo toho vyhlazuje objektivní funkci pro zvýšení výkonu. Skutečností je, že pomocí této techniky lze usnadnit učení neuronových sítí, které například u rekurentních LSTM sítí může být náročné.

1.3.3.3 Dropout

Dropout je technika, kdy jsou v neuronové síti vynechány některé neurony, které jsou určeny náhodně faktorem d. Tyto neurony jsou vynechány v trénovací fázi. V testovací fázi při vybavování jsou použity všechny neurony, ale výstup je redukován faktorem d, kvůli chybějícím neuronům během učení. Stejně lze aplikovat dropout i v rámci časových kroků v RNN. Dropout pro rekurentní síť je nastavován stejně pro všechny časové kroky. Síť se tedy učí s menším počtem neuronů. Tato technika zpomaluje učení neuronové sítě, ale předchází přeučování. Síť, která je učena s využitím dropoutu, je schopná větší generalizace a učí se robusnější a kvalitnější vlastnosti řešeného problému.

1.3.3.4 Variabilní učící faktor

Další technikou, která může umožnit vytvoření kvalitnějšího modelu je variabilní učící faktor. V ideálním případě může mít každá váha svůj vlastní učící

faktor, jako je tomu například u optimalizační metody adam, který je popsán níže a zadaný učící faktor využívá k nastavení počáteční hodnoty. Učící faktor je pak upravován dynamicky podle stavu učení daného parametru tak, aby nebyly prováděny příliš velké kroky, které nepovedou ke konvergenci a naopak malé kroky, které neumožní učit velmi sparse parametry nebo i například opustit lokální minimum.

Většinou je použit variabilní učící faktor, který je stejný pro vrstvy nebo pro celou síť. Pro celou síť je většinou nastavena základní hodnota, která je měněna optimalizačním algoritmem. Ze začátku učení může být vhodné mít učící faktor větší pro prozkoumávání prostoru objektivní funkce, ale s přibývajícími epochami a přibližováním se k nějakému minimu, které nemusí být zrovna globální, může být vhodné učící faktor snižovat a tím zmenšit kroky a umožnit modelu tohoto minima dosáhnout.

1.3.3.5 Optimalizační algoritmy

Trénování RNN není většinou prováděno pouze použitím gradientního sestupu v kombinaci s BPTT, jak bylo posáno v sekci 1.3.2.1, ale BPTT může fungovat i s jinými optimalizačními technikami, kde algoritmus BPTT provádí změnu vah ve směru klesající chyby, jak již bylo popsáno, ale samotná hodnota změny vah může být určena i jinou optimalizační technikou, která může urychlit trénování nebo konvergenci.

Běžný gradientní sestup provádí aktualizaci vah s využitím všech vzorků, ale to je velmi výpočetně náročné. Všechny metody jsou ovšem s gradientem nějak spojeny a některé využívají i druhou derivaci. Mezi takové optimalizační metody se kromě klasického gradientního sestupu může řadit stochastický gradientní sestup (SGD), momentová metoda, metoda adam, adagrad nebo adadelta.

- **Stochastický gradientní sestup** byl již popsán v 1.2.2 v rámci sekce MF 1.2. Jeho princip je stejný jako běžný gradientní postup jen s tím rozdílem, že je aktualizace vah prováděna po jednom trénovacím vzorku. Tato metoda byla popsána v algoritmu 3, kde byl náhodně vybrán jeden trénovací vzorek. Pokud trénování a aktualizace vah probíhá po jednom trénovacím vzorku, mohou mít optimalizované váhy veliký rozptyl, a proto je místo jednoho vzorku využita technika mini batches popsaná v 1.3.3.1. Počítání aktualizací je prováděno způsobem, jak bylo pro RNN popsáno v sekci 1.3.2.1.
- **Momentová metoda** je vylepšením metody SGD. Metoda SGD má problém s vysokým rozptylem hodnot trénovaných parametrů, což zpomaluje konvergence. Kromě použití metody malých dávkových aktualizací, je možné přidat k aktuální změně váhy i hodnotu z předchozí aktualizace, což může pomoci vyhladit velký rozptyl hodnot parametrů a je možné jej kombinovat i s technikou malých dávkových aktualizací

hodnot. Podle publikace [25] můžeme tento člen přidat do aktualizace váhy a zapsat následujícím způsobem:

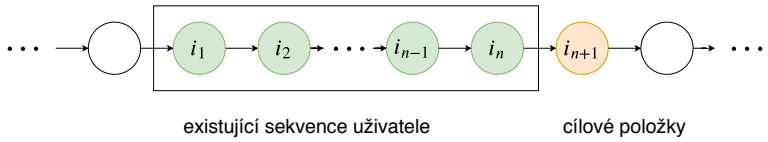
$$\begin{aligned} \Delta w_{i,j}^{(l)} &\leftarrow \eta \Delta w_{i,j}^{(l)} - \gamma \frac{\partial E^{(t)}}{\partial w_{i,j}^{(l)}} \\ w_{i,j}^{(l)} &\leftarrow w_{i,j}^{(l)} + \Delta w_{i,j}^{(l)}, \end{aligned} \tag{1.57}$$

kde index i odpovídá indexu vstupu a j indexu v aktualizované vrstvě, t je časový index pro rekurentní neuronovou síť a η je faktor míry vyhlazování, který je v intervalu $(0, 1)$ a jedná se o parametr určující změnu váhy z předchozího kroku.

- **Adam** je poslední a hojně používanou optimalizační metodou. Název tohoto optimalizačního algoritmu je odvozen od adaptivních momentů (angl. adaptive moments), které využívá. Metoda byla představená v publikaci [38] a počítá různé faktory učení pro různé parametry. Algoritmus používá odhady prvního a druhého momentu gradientu k přizpůsobení rychlosti učení pro každý učený parametr. Hlavní kroky algoritmu lze podle publikace [38] zapsat způsobem níže, kde index t zde neznamená krok v RNN, ale jeden krok v učení. Krok v RNN zde ve značení kvůli přehlednosti uveden nebude, ale lze tento postup využít pro RNN nebo na LSTM aplikací na všechny parametry, které jsou učeny:

$$\begin{aligned} \mathbf{m}_t &\leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} E \\ \hat{\mathbf{m}}_t &\leftarrow \frac{\mathbf{m}_t}{(1 - \beta_1^t)} \\ \mathbf{v}_t &\leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_1) \left(\nabla_{\boldsymbol{\theta}} E \right)^2 \\ \hat{\mathbf{v}}_t &\leftarrow \frac{\mathbf{v}_t}{(1 - \beta_2^t)} \\ \boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\hat{\mathbf{m}}_t}{(\sqrt{\hat{\mathbf{v}}_t} + \epsilon)} \end{aligned} \tag{1.58}$$

kde θ je aktualizovaný parametr, který může představovat vychýlení nebo matici vah pro různé vazby v LSTM, α je velikost kroku při učení, β_1 a β_2 jsou parametry exponenciálního snižování pro odhady momentů, kde β^t představuje mocninu s exponentem t. Všechny operace na vektorech jsou bodové a platí $\left(\nabla_{\boldsymbol{\theta}} E \right)^2 = \nabla_{\boldsymbol{\theta}} E \odot \nabla_{\boldsymbol{\theta}} E$. Jak je zmíněno v publikaci [38], vhodnými hodnotami pro parametry můžou být $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ a $\epsilon = 10^{-8}$. Tato metoda je v dnešní době velice populární díky rychlé konvergenci oproti SVG.



Obrázek 1.8: Sekvence interakcí uživatele s položkami při použití plovoucího okna je nejčasteji využíváný princip u tohoto typu dat [39]

1.3.4 RNN v doporučovacích systémech

Jak bylo zmíněno v sekci 1.3.1 v této práci, RNN jsou sítě, které díky vnitřnímu stavu (cyklus v orientovaném grafu) mají nějakou paměť a jsou vyhodnocovány s ohledem na časové kroky. Jejich použití je tedy vhodné na zpracování sekvenčních dat popsaných v sekci 1.1.2.2. V této sekci jsou uvedena a popsána vybraná existující řešení, ve kterých jsou RNN využity pro generování doporučování nebo predikci další uživatelské akce v závislosti na aktuálně provedených akcích.

Společným rysem všech uvedených řešení je nabídnout obsah na základě předchozího chování uživatele, které představuje sekvence interakcí. Jeden uživatel může být reprezentován i jednou relací, protože uživatelé mohou být anonymní. Tyto relace mohou být v různých datasetech jinak dlouhé podle implementace trasování relací na dané doméně, ze které data pocházejí. Na některých doménách mohou být sekvence velmi dlouhé a obsahovat i stovky položek a na druhou stranu na doméně s pouze anonymními relacemi mohou být velmi krátké.

Základní princip pohledu na data pro vstup do rekurentní neuronové sítě je ukázán na obrázku 1.8, kde je možné vidět využití plovoucího okna pro určený počet interakcí. Interakce v plovoucím okně slouží jako vstup do RNN. Pomocí RNN je cíl predikovat další položku v této sekvenci. V případě další interakce od uživatele je toto plovoucí okno posunuto. Je tedy pevně daná délka sekvence a pro učení je vhodné využít algoritmus BPTT popsaný v 1.3.2.1 a princip rozbalení celé rekurentní vazby v paměti pro rychlé trénování.

Druhou možností je neomezování délky sekvence, protože RNN lze vyhodnocovat i bez restartu a omezení délky sekvence a trénovat například s využitím RTRL z 1.3.2.2. V prvním doporučovacím systému popsaném v publikaci [40], který využívá RNN s uzávěrkovým mechanismem, není délka sekvence omezována. V této publikaci jsou použity GRU rekurentní jednotky. Představený model se skládá ze vstupní vrstvy, která má počet neuronů roven počtu položek a je použito kódování 1 z N, kde 1 je pro položku v sekvenci na daném místě a ostatní jsou 0. Další vrstva je rekurentní GRU vrstva. Po rekurentní GRU vrstvě je výstupní vrstva, kde je opět použita reprezentace pomocí 1 z N a predikuje se skóre.

Doporučeny jsou položky s největší hodnotou predikovaného skóre. V publikaci je také zmíněn problém tohoto přístupu, který představuje možné velké

množství položek a tím velké množství parametrů a paměťovou náročnost. Problém je řešen přidáním embedding vrstvy mezi vstupní vrstvu a GRU vrstvu. Dále je zmíněno, že tímto přístupem nebylo dosaženo lepších výsledků než při použití kódování 1 z N. Optimalizace pro velkou výstupní vrstvu je řešena výběrem podmnožiny položek, které jsou predikovány. Tímto ovšem může být omezena možnost predikce všech položek.

V publikaci [41] používají také GRU jednotky jako v předchozím případě a stejná je i architektura s tím, že první vrstva představuje embedding vrstvu pro vytvoření husté reprezentace položek z reprezentace 1 z N. V této publikaci je použita pevná délka sekvence. Dále je v této publikaci představen dropout, který je aplikován na vstupní sekvenci při učení. Tento dropout znamená, že jsou vynechány některé položky náhodně pro větší robusnost při učení. V této práci je také představena metoda pro řešení problému s velkým množstvím položek, kterých mohou být v doporučovacích systémech až desítky miliónů. Za vstupní vrstvou se nachází vrstva pro generování embeddingů, které jsou trénovatelné. Model predikuje přímo embeddingy jako výstup. Je předpokládáno, že položky, na které uživatel klikne společně po nějaké sekvenci, jsou podobné i v případě podobnosti embeddingů. K této podobnosti je využita kosinová podobnost.

V dalším přístupu v publikaci [42] vytvářejí a testují několik architektur. Využívají obrázkový embedding společně s 1 z N kódování pro položky. Výstup je pokaždé v kódování 1 z N. V představených architekturách používají GRU jednotky. Odlišný je přístup ke vstupu, kdy jednou je použit vstup jako položky v kódování 1 z N nebo obrázkové embeddingy a tyto způsoby odpovídají výše zmíněným modelům. Jak je v publikaci uváděno, samostatně modely pouze s jedním typem vstupu nedosáhly takových výsledků jako v případě jejich kombinace. Kombinovány jsou buď spojením a použitím jedné GRU vrstvy nebo odlišných GRU vrstev pro různé reprezentace položek. Výstupy z jednotlivých GRU bloků jsou váženy a mají společnou výstupní vrstvu.

V článku [43] jsou využity GRU jednotky a je představen celý framework i s doménou. V tomto modelu není využita výstupní dopředná vrstva, ale přímo výstup z rekurentních vrstev. Dále jsou využity embeddingy v prostoru euklidovské vzdálenosti, do kterého jsou mapováni jak uživatelé, tak položky. Předpokládá se, že podobní uživatelé a položky se v tomto prostoru nacházejí blízko sebe.

V posledním představeném přístupu v článku [17] jsou pro predikci využity LSTM jednotky místo GRU jednotek používaných v předchozích implementacích. Pro vstup i výstup používají kódování 1 z N. V práci jsou představeny různé možnosti předzpracování sekvencí, kdy první z nich je vynechávání některých položek. Tento postup byl už představen v druhé publikaci. Další uvedenou možností předzpracování je změna pořadí položek v sekvenci, kdy jsou některé položky náhodně prohozeny.

Kapitola 2

Návrh

V této kapitole je popsán návrh algoritmu pro tvorbu modelů a generování doporučování, který využívá sekvenční data a staví na teoretických základech z předchozí kapitoly 1. Také je představen model maticové faktorizace pro generování latentních příznaků jako reprezentaci položek pro rekurentní model, ale také pro samotný referenční model MF.

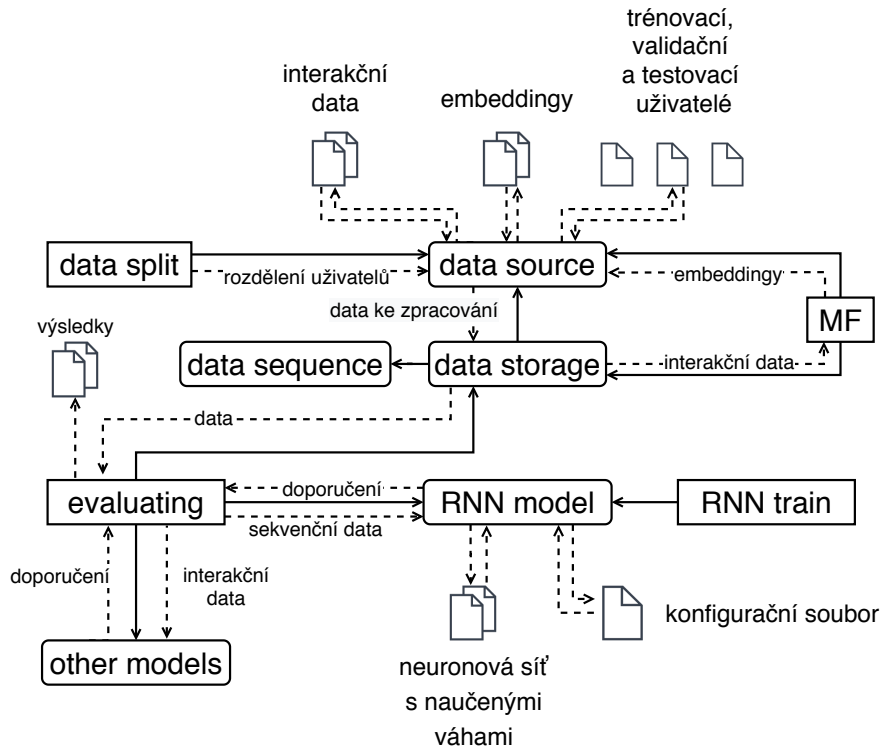
Nejprve je popsána obecná struktura na abstraktní úrovni a poté jednotlivé části (předzpracování dat pro modely, samotné modely, metodika trénování a vybavování modelů). Dále je v této kapitole uvedeno zpracování některých kontextových dat a jejich použití v neuronové síti. Je představena architektura rekurentní neuronové sítě pro vytvořený model, principy tvorby a možnosti vybavování vytvořených modelů složených z rekurentních sítí. Také je navržen princip testování modelů, který bere v úvahu sekvenční povahu dat a vychází z existujících metrik.

2.1 Obecná struktura

Na obrázku 2.1 je schematicky zobrazena struktura celého frameworku. Je možné vidět jednotlivé moduly, které jsou ve čtvercích a zajišťují vybrané funkcionality. Tyto moduly představují hlavní části, které jsou spouštěny. Zaoblenými čtverci jsou zobrazeny moduly, které slouží jako komponenty, ale nejsou spustitelné samostatně. V následujícím seznamu jsou popsány hlavní funkcionality modulů:

Data source je modul sloužící k načítání interakčních dat, k serializaci latentních příznaků (embeddingů) pro položky a k práci se seznamy s trénovacími, validačními a testovacími uživateli. V této práci bude použit modul pro serializaci do csv souborů z důvodu jednoduchosti a přenositelnosti mezi stroji. S využitím stejného rozhraní je možné využít téměř jakékoliv uložistě.

Data split je prvním spustitelným modulem a je také nejjednodušší. Tento modul načítá interakční data a vytváří množiny uživatelů pro trénování, va-



Obrázek 2.1: Obecná struktura zobrazená pomocí modulů, do které jsou zasazeny navržené algoritmy a je zobrazen tok a uložení nejstěžejnějších dat u jednotlivých modulů

lidaci a testování pro ostatní modely. Velikosti těchto množin jsou nastaveny procentuálně a lze omezit i celkový počet uživatelů (také procentuálně). Uživatelé jsou do jednotlivých množin vybíráni náhodně bez opakování. Další funkcionalitou modulu je filtrování uživatelů s málo interakcemi i filtrování robotů s mnoha interakcemi. Tímto lze předejít situaci, kde by se v menší testovací sadě nacházelo příliš mnoho uživatelů jen s několika interakcemi, kteří by testování vychylovali. Data jsou vždy generována pro konkrétní experiment, takže je možné nagenerovat více rozdělení dat. Tímto je například možné zjistit, zdali je velikost testovací množiny dostatečná.

Data storage modul zpracovává a ukládá data. Provádí předzpracování interakčních dat pro ostatní modely a moduly. Dále načítá rozdělení uživatelů pro konkrétní experiment, které je aplikováno na načítaná data. Poslední načítanými jsou embeddingy vytvořené pomocí maticové faktorizace. Jelikož není pokaždé potřeba načítat všechna data, tak jsou data načítána a předzpracována v moment, kdy jsou požadována některým modelem poprvé.

MF modul slouží k vytvoření embeddingů pro položky. Maticová faktorizace využívá ratingovou matici popsanou v 1.1.2.1, do které jsou agregovány implicitní interakce s přiřazenými váhami. Embeddingy jsou vytvořeny pouze pro položky obsažené v trénovací množině uživatelů.

Data sequence slouží pro přípravu dat při trénování rekurentní neuronové sítě. Jedná se o modul vytvařející mini batches popsané v 1.3.3.1 a jejich přípravu pro jednotlivé epochy. Implementační detaily jsou zmíněny v kapitole 3.

RNN model reprezentuje samotnou neuronovou síť, která je konfigurována pomocí yaml konfiguračního souboru. Jednotlivé natrénované epochy jsou ukládány jako binární soubory vytvořených sítí a je k nim přiřazen aktuální konfigurační soubor pro pozdější evaluaci. Stejně tento modul umí načíst vytvořenou síť a provádět vybavování. Oproti ukládání samotných vah je binární soubor vybrán z důvodu možností změny architektury sítě). Pokud by se ukládaly pouze váhy, bylo by nutné vytvořit stejnou síť (např. konfiguračním souborem. S binárním souborem lze vyhodnocovat modely s libovolnou architekturou, které mají stejný formát vstupu a výstupu.

Evaluating je modul, který provádí vyhodnocování podle konfiguračního souboru, kde jeho nejdůležitější části jsou popsány v kapitole 3. Modul pracuje s obecným rozhraním, které může implementovat libovolný model a může být jednoduše pomocí tohoto modulu vyhodnocován. Výsledky jsou ukládány do csv souborů v dané souborové struktuře.

RNN train je modul, který řídí trénování neuronové sítě – jedná se pouze o spuštění načtení dat, připravení struktur a spuštění trénování.

Other models není pouze jeden modul, ale pod tento modul spadají další modely, se kterými budu svůj model porovnávat. Mezi tyto modely patří item-knn a to jak interakční, tak i nad embeddingy z maticové faktorizace. Dále sem patří user-knn, popularity model a na závěr jednoduchý opakovací model, tzv. reminder.

2.2 Využití maticové faktorizace – generování reprezentace položek (embeddings)

Teorie ohledně maticové faktorizace byla popsána v sekci 1.2. Pro generování embeddingů je využit model, který pracuje s implicitními interakcemi a je popsán na závěr maticové faktorizace v podsekci 1.2.5. Data, která vstupují do maticové faktorizace, jsou reprezentována ratingovou maticí. Základní rozklad, který je maticovou faktorizací prováděn, je zobrazen na obrázku 1.1.

Data z implicitních interakcí je nutné předzpracovat do ratingové matice. V práci počítám s třemi základními interakcemi, kterými jsou zobrazení detailu produktu, přidání do košíku nebo zakoupení produktu. Těmto interakcím jsou přiřazeny váhy, kde zobrazení produktu je určitě méně důležité než přidání do košíku nebo koupení položky. Pro zobrazení přiřadím váhu 0,25 a pro přidání do košíku a nákup např. 0,75, ale tyto hodnoty je možné nastavit v konfiguračním souboru pro experimentování s těmito hodnotami. Hodnoty pro interakce se stejnou položkou jsou do ratingové matice sčítány do nějaké zvolené maximální hodnoty.

Takto vytvořenou matici je nutné před použitím maticové faktorizace dále předzpracovat, protože se v datech mohou vyskytovat položky, na které kliknou všichni uživatelé tzv. bestsellery nebo určitě můžou existovat uživatelé, kteří systematicky klikají na velké množství položek.

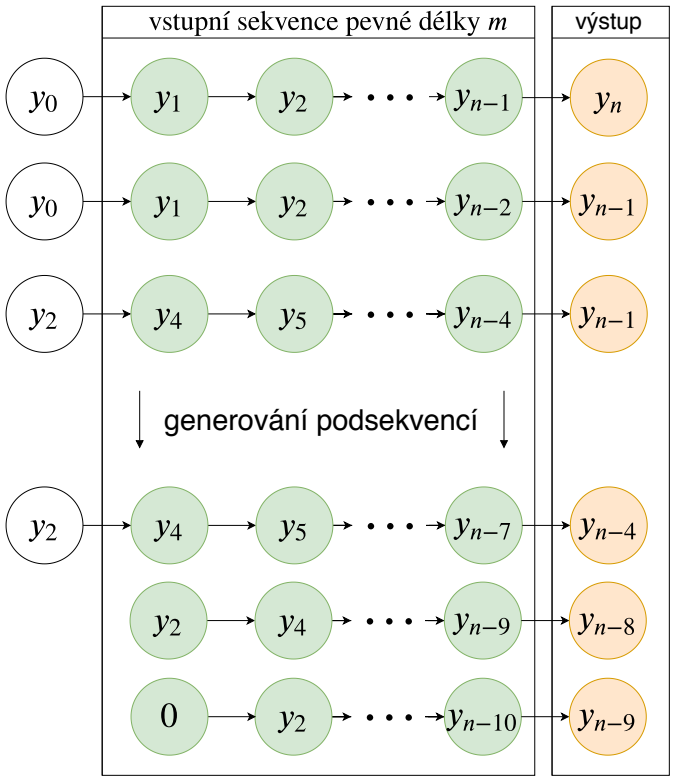
Je použito pravděpodobnostní vážící schéma BM25, které je podobné například známému schématu tf-idf, ale s využitím normy délky vektoru s parametrem b. S větším b se zesilují účinky této normy, kterou je možné vidět ve výrazu 2.1. Dále oproti tf-idf mají menší vliv vysoké hodnoty ve frekvenci termů nebo použité hodnoty vah z implicitních interakcí, jako v případě této práce. Pro zvyšující se hodnotu této složky se hodnota blíží limitně k nějaké hodnotě a neroste neustále.

V práci mi jde především o získání vektorů latentních příznaků pro položky, latentní příznaky pro uživatele jsou ignorovány. Proto je vážení provedeno pro vektory položek. Pro ratingovou matici můžu vážení přes položky zapsat:

$$
\begin{aligned}
r_{u,i} &= idf(r_{u,i}) \cdot \frac{r_{u,i} \cdot (k_1 + 1)}{r_{u,i} + k_1 \cdot \left(1 - b + b \cdot length\_norm\right)} \\
idf(r_{u,i}) &= ln\left(\frac{\#(r_{:i})}{\#\limits_{r_{u,i} \neq\, ?}(r_{:i})}\right) \\
length\_norm &= \frac{\sum\limits_{\substack{k \in U \\ r_{k,i} \neq\, ?}} r_{k,i}}{\frac{1}{|I|} \cdot \sum\limits_{j \in I} \sum\limits_{\substack{k \in U \\ r_{u,j} \neq\, ?}} r_{k,j}},
\end{aligned}
\tag{2.1}
$$

pro hodnoty $i \in I$, $u \in U$, kde $r_{u,i} \neq ?$.

Poslední úpravou, kterou se pokusím otestovat je přenásobení prvků matice nějakým koeficientem. Tento koeficient je nazván m a je to další hyperparametr maticové faktorizace. Jeho použití zvětší hodnoty v matici a tedy i rozdíly mezi nimi, proto může pozitivně ovlivnit MF, protože se tím i zvětší velikost chyby. Dále parametr může přinést vyšší stabilitu při počítání v plouvoucí desetinné čárce.



Obrázek 2.2: Ukázka zpracování sekvencí interakcí. Možnosti jsou odřezávání interakcí, odstraňování podobných a generování podsekvencí ze sekvencí

Nad takto upravenou maticí je provedena maticová faktorizace podle 1.2.5. Získané latentní faktory představují reprezentaci položek a slouží jako vstup do neuronové sítě.

Pro podobnost vektorů v algoritmech user-knn a item-knn se používá kosinová podobnost, která byla zmíněna v 1.7. Tuto podobnost lze využít i nad vytvořenými embeddingy. Model s embeddingy jde vybavovat stejným způsobem jako item-knn a tvoří jeden z referenčních modelů, pomocí kterého je možné měřit kvalitu embeddingů. Tohoto faktu, že se embeddingy nacházejí v prostoru kosinové podobnosti, bude využito při konstrukci ztrátové funkce pro učení neuronové sítě v podsekci 2.5.

2.3 Předzpracování a integrace sekvenčních interakčních dat

Oproti agregaci do ratingové matice je předzpracování na sekvenční data přímočařejší, protože mnohem lépe odráží samotná data, jak jsou získávána

například z log souborů. Základ je přiřazení různých interakcí uživateli a setřídit je v čase.

RNN je typicky trénována tak, že první část sekvence tvoří data, která slouží jako vstup. Výstup je poté srovnán s poslední položkou, kterou se síť snaží doporučit. Používá se učení s učitelem, jak bylo zmíněno v sekci 1.3.

Předzpracování je zobrazeno na obrázku 2.2. Sekvence jsou předzpracovány do sekvencí pevné délky, jak je ukázáno na obrázku s požadovaným výstupem. Nemusí být vždy vhodné končit sekvenci položkou, kterou si uživatel pouze zobrazil, ale pokud jsou k dispozici i interakce, které mají typicky vyšší váhu, tak sekvenci oříznout tak, aby končila například nákupem položky. Tato data pak představují přesně cíl doporučování. Tedy koupi položky a historii, která jí předcházela. Tento typ předzpracování je zobrazen na poslední položce a prvních dvou řádcích na obrázku 2.2.

Další možností, jak lze sekvenci předzpracovat, je vynechání některých položek z historie uživatele. Pokud existuje například mnoho stejných zobrazení téhož produktu v krátkém časovém horizontu, může být výhodné tyto interakce sloučit nebo sloučit zobrazení produktu a jeho koupi, pokud se nacházejí opět v nějakém časovém okně a ponechat pouze tu významější interakci.

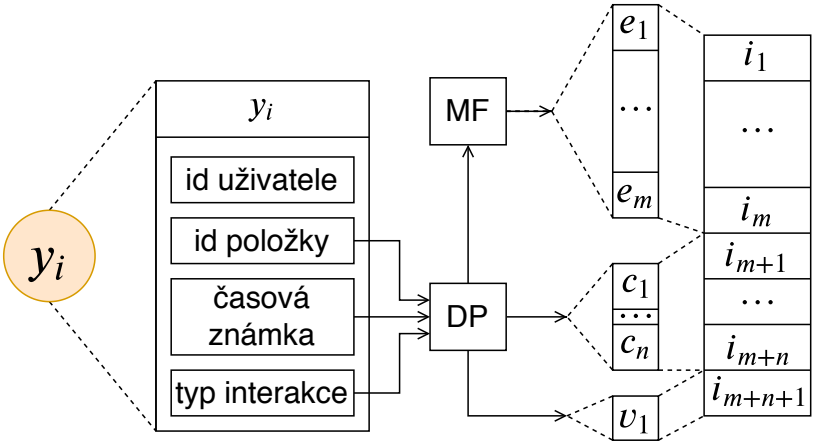
Posledním prováděným předzpracováním je možnost generování podsekvencí, pokud je sekvence dostatečně dlouhá. Interakce je možné odřezávat od těch nejnovějších pro dosažení různorodějších trénovacích dat a jejich většího počtu. Pokud je sekvence odříznuta podle času, tak interakce představují chování uživatele před tím, než si prohlédl nebo koupil danou položku.

2.4 Rozšíření vstupu o kontextová data a jejich zpracování

Získané vektory embeddingů jsou základním vstupem do rekurentní neuronové sítě. Vstupem do neuronové sítě ovšem nemusejí být pouze embeddingy, ale vstup je možné rozšířit o různá kontextová data. Jako první se nabízí rozšíření o časovou známku interakce nebo její typ. Dalšími údaji můžou být atributová data o položkách převedená do vhodné reprezentace.

V této práci je rozšířen vstup o typ interakce a také o časovou známku, kde časový údaj je nutné vhodně zpracovat, aby mohl být použit v neuronové síti. Základní princip tohoto rozšíření je zobrazen na obrázku 2.3, kde je vidět, že se jedná o rozšíření původního embeddingu o další data. Embedding je vytvořen pomocí maticové faktorizace. Časový údaj a váha jsou předzpracovány z interakčních dat v data storage.

Typ interakce bude pouze zaměněn za váhu, které jsou nastaveny v rozsahu $[0, 1]$ a není nutné je dále zpracovávat. Váhy je možné nastavit stejně, jak bylo popsáno v sekci 2.2. Časový údaj není vhodné přidávat jako datum v roce, i když i to v sobě může obsahovat nějakou informaci (roční období). V práci



Obrázek 2.3: Zpracování interakce pro rekurentní neuronovou síť s využitím kontextových dat interakce

jsou použity časové rozdíly mezi interakcemi. První interakce v sekvenci bude mít nulový čas a ostatní budou mít rozdíl času k předchozí interakci. Tento rozdíl je jako desetinné číslo v sekundách a může nabývat i vysokých hodnot, které by nebyly vhodné jako vstup do neuronové sítě, proto jsou dále zpracovány.

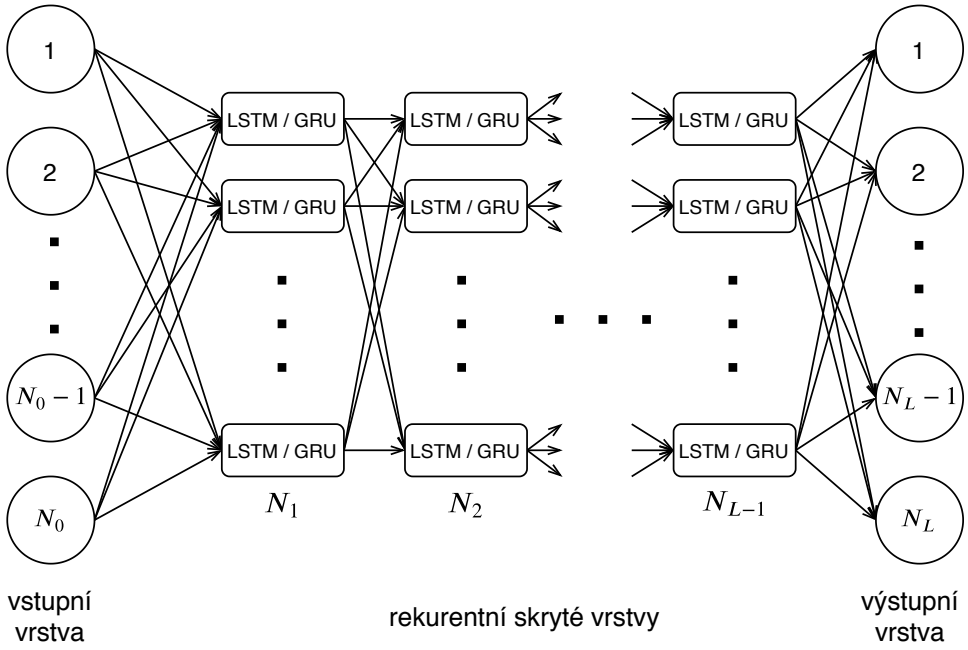
Prvním možným způsobem zpracování je použití min-max normalizace na hodnoty zapsané do vektoru $\mathbf{x}$ v intervalu $[-1, 1]$:

$$y_i = 2 \cdot \frac{x_i - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} - 1, \tag{2.2}$$

kde $\max(\mathbf{x})$ znamená maximální hodnotu z vektoru $\mathbf{x}$ a pro minimum obdobně. Z výsledné hodnoty lze převést zpět na původní hodnotu:

$$x_i = \frac{(y_i + 1) \cdot (\max(\mathbf{x}) - \min(\mathbf{x})) + 2 \cdot \min(\mathbf{x})}{2} \tag{2.3}$$

Dalším způsobem je vytvoření diskretizace pro tyto hodnoty, kde počet binů je 2^x, tak aby biny bylo možné reprezentovat číslem v binární podobě a byly využity všechny jeho bity maximálně. Číslo binu v binárním zápisu slouží jako další část vektoru, jak je znázorněno na obrázku 2.3. Délka části vektoru, která reprezentuje čas, může být tedy variabilní. Diskretizaci je možné provést, jak v rámci celého datasetu, tak pouze pro danou trénovací sekvenci. Prováděna je diskretizace s pevnou velikostí binů, protože na rozdíl od frekvenční, lze pomocí této zachytit časovou prodlevu mezi dvěma interakcemi, kde určitý bin může zůstat prázdný.



Obrázek 2.4: Architektura neuronové sítě, kde nejsou zobrazeny rekurentní vazby, ale LSTM nebo GRU buňky představují rekurentní buňku popsanou v 1.3.1.1 nebo 1.3.1.2 se zpětnými vazbami

2.5 Návrh modelu a architektury využívající RNN při doporučování

Základní architektura je zobrazena na obrázku 2.4. Jak je na něm vidět, síť se skládá ze vstupní vrstvy, rekurentních vrstev a poté výstupní plně propojenou vrstvou. Toto je zakládní charakteristika. Většinu zbylých hyperparametrů lze nastavit v konfiguračním souboru, který je popsán v 3. V této části jsou dále popsány některé zvolené hodnoty a ztrátová funkce, pomocí které je neuronová síť trénována.

Vstup do sítě je reprezentován tak, jak bylo popsáno v sekci 2.4. Kontextová data nemusí být ve vstupu přítomna, ale embedding položky tvoří vstup vždy. Výstupní vrstva je stejné velikosti jako vstupní, protože je představována také embeddingem reprezentující položku interakce podobně jako u vstupu.

Zásadní při učení neuronové sítě je ztrátová funkce, která určuje chybu sítě proti požadovanému výstupu. Jak bylo zmíněno v sekci 2.2, embeddingy jsou v prostoru kosinové podobnosti, a proto první návrh ztrátové funkce je využití kosinové podobnosti, která je ovšem upravená tak, aby hodnota -1 vyšla pro stejné vektory, 0 pro kolmé a 1 pro opačné. Toto je z důvodu konvence, aby bylo možné využít existují algoritmy učení, které se snaží hodnotu ztrátové funkce minimalizovat. Tato kosinová podobnost byla využita i ve třetím popsaném

řešení v sekci 1.3.4. Takto upravenou kosinovou podobnost označím *cosp*:

$$cosp\left(\mathbf{y}, \hat{\mathbf{y}}\right) = -\frac{\mathbf{y} \cdot \hat{\mathbf{y}}}{\|\mathbf{y}\|\|\hat{\mathbf{y}}\|}, \tag{2.4}$$

ze které je pak jednoduše vytvořená ztrátová funkce, která splňuje podmínku, aby byla derivovatelná.

$$\mathcal{L} = cosp\left(\mathbf{y}, \hat{\mathbf{y}}\right). \tag{2.5}$$

Tato ztrátová funkce sama o sobě má jeden poblém, že síť má ve výsledné vrstvě velké hodnoty, které jsou sice blízko v kosinové podobnosti a lze nalézt v prostoru embeddingů nejbližší, které mají být doporučeny. Tyto vysoké hodnoty však znemožnují jejich opětovné použití v síti při jiných způsobech vybavování, které jsou popsány v podsekci 2.5.2. K řešení tohoto problému je v práci využita kombinace ztrátových funkcí s určenými váhami. Jako druhou ztrátovou funkci použiji MSE:

$$MSE\left(\mathbf{y}, \hat{\mathbf{y}}\right) = E\Big[\left(\hat{\mathbf{y}} - \mathbf{y}\right)^2\Big] \tag{2.6}$$

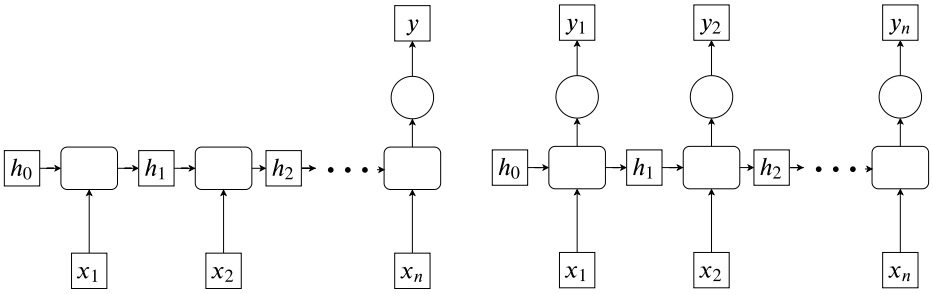
a po přidání vah ω_1 a ω_2 vznikne následující ztrátová funce:

$$\mathcal{L} = \omega_1 \cdot cosp\left(\mathbf{y}, \hat{\mathbf{y}}\right) + \omega_2 \cdot MSE\left(\mathbf{y}, \hat{\mathbf{y}}\right). \tag{2.7}$$

S touto ztrátovou funkcí lze řídit i velké odchylky hodnot, které jsou pomocí MSE složky penalizovány. Při použití kontextových dat ze sekce 2.4 je potřeba tuto funkci ještě upravit. Pro časovou a váhovou složku není použita kosinova podobnost, ale pouze MSE. I těmto částem lze přiřadit konkrétní váhy. Na jednotlivé složky výstupu je možno aplikovat příslušné funkce a výslednou ztrátovou funkci zapsat:

$$\begin{aligned}\mathcal{L} = &-\omega_1 \cdot \frac{\sum_{i=1}^{m} \mathbf{y}_i \hat{\mathbf{y}}_i}{\sqrt{\sum_{i=1}^{m} \mathbf{y}_i^2}\sqrt{\sum_{i=1}^{m} \hat{\mathbf{y}}_i^2}} + \omega_2 \cdot \frac{1}{m}\sum_{i=1}^{m}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \\ &+\omega_3 \cdot \frac{1}{n}\sum_{i=m+1}^{m+n}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \\ &+\omega_4 \cdot (\mathbf{y}_{m+n+1} - \hat{\mathbf{y}}_{m+n+1})^2.\end{aligned} \tag{2.8}$$

Ztrátová funkce je základní kámen, pomocí kterého je definována chyba a je možné učit neuronovou síť. Sekvence jsou pevné délky a pokud je sekvence kratší než požadovaná délka, je doplněna nulami, jak bylo ukázáno na



Obrázek 2.5: Zobrazeny dva nejčastější modely běhu RNN, vlevo *many-to-one* a vpravo *many-to-many*

obrázku 2.2. Pro trénování je použit algoritmus BPTT popsaný v 1.3.2.1 s optimalizátorem adam, který byl popsán v sekci 1.3.3.5.

Další použitou optimalizací je použití normalizační vrstvy (batch normalization) popsané v 1.3.3.2. Normalizační vrstva je vložena mezi poslední rekurentní vrstvu a výstupní vrstvu. Při trénování jsou také využity dávkové aktualizace vah (mini batches) z 1.3.3.1. Jiné optimalizační algoritmy, jako naříklad momentová metoda nebo SGD, budou otestovány v experimentech.

Důležitým parametrem jsou také aktivační funkce neuronů. S těmi nebude experimentováno z důvodu problému s mizejícím a explodujícím gradientem zmíněným v sekci 1.3. Pro výstupní vrstu bude použita lineární aktivační funkce, protože výsledné embeddingy mohou nabývat i jiných hodnot než pouze v rozsahu $[0, 1]$ nebo $[-1, 1]$. Pro rekurentní vrstvy budou použity aktivační funkce vhodné pro LSTM a GRU popsané v sekci 1.3.1.1 a 1.3.1.2.

Poslední optimalizací, která bude využita, je dropout ze sekce 1.3.3.3. Dropout je možné opět aplikovat několika způsoby a to mezi vstupní a rekurentní vrstvu, mezi rekurentní vrstvy nebo před výstupní vrstvu. Další možná aplikace je mezi rekurentní vazby mezi jednotlivými časovými kroky.

2.5.1 Modely běhu RNN

Při bližším zkoumání obrázku 1.4 je možné najít několik možných způsobů, jak model vybavovat a také odpovídajícím způsobem trénovat. Tyto způsoby lze nazvat podle vstupu a výstupu, one nebo many. One pro jednu položku a many pro nějakou sekvenci položek. Prvním základním modelem je tedy *one-to-one* model, do kterého lze zařadit i například dopřednou neuronovou síť, která dostane vstup a je spočítán odpovídající výstup. Vstup si lze představit jako sekvenci délky jedna, ale tento způsob by nevyužil potenciál rekurentních neuronových sítí.

Druhý způsob, který vychází přímo z popisu RNN, je zobrazen vlevo na obrázku 2.5. Model je možné nazvat *many-to-one*. Máme vstupní sekvenci a cílem je predikovat jednu položku, jak je i vidět na obrázku. Z modelu je

tedy využit poslední výstup z rekurentní vrstvy. Tento způsob bude primární v této práci.

Dalším otestovaným způsobem v této práci je vybavování *many-to-many*, kde síť jako odpovídající výstup dostane posunutou vstupní sekvenci o jednu položku. Cílem je předpovědět další položku v každém kroku vybavování a podle toho je síť také trénována. Výsledkem je sekvence jako u vstupu a musí být použity odpovídající struktury, aby mohla být v modelu využita poslední dopředná vrstva.

Způsob je ukázaný vpravo na obrázku 2.5. Modifikací zobrazeného přístupu může být vynechání některých kroků a nechat vybavování pouze na vnitřním stavu. Tento zbůsob je však náročný na trénování.

Posledním způsobem, který vůbec není uvažován, je *one-to-many*, kde je vstupem jedna položka a výstupem je sekvence. Rekurentní síť je vybavována opakovaně pouze s prvním vstupem a bere se výstup z každého časového kroku. Je využíváno vnitřního stavu.

2.5.2 Metody vybavování a generování doporučení

Vybavování sítě a generování doporučení je možné provádět několika způsoby při použití výše navrženého modelu. Prvním metodou je vygenerování embeddingu při vybavování, ke kterému jsou nalezeni nejbližší sousedé v prostoru těchto embeddingů a položky jim odpovídající jsou využity jako doporučení.

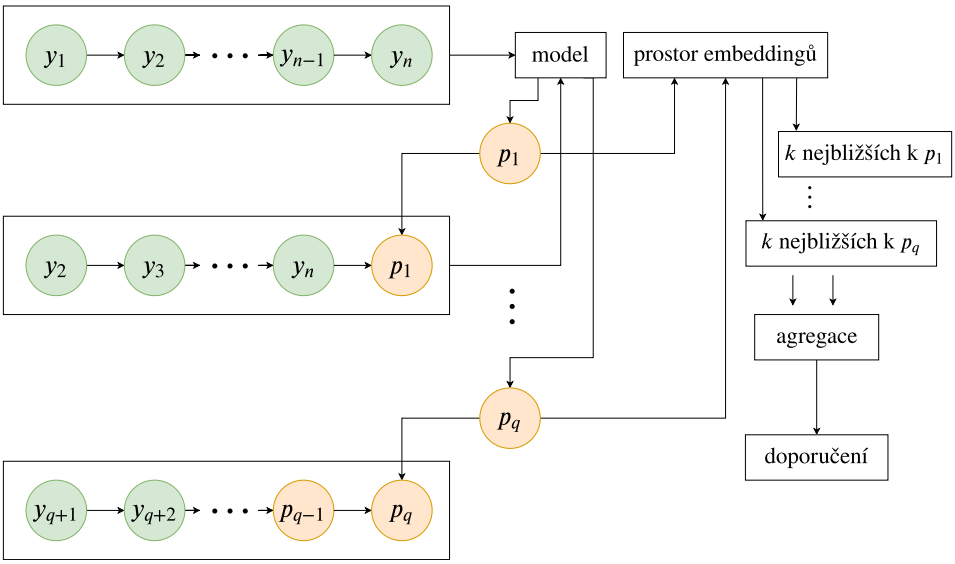
Druhou metodou, která je ukázána na obrázku 2.6, se pomocí vygenerovaného embeddingu rozšiřuje vstupní sekvence a ta je znovu použita jako vstup do modelu. Je tedy předpokládáno, že je to přesně ten embedding, který je následující v sekvenci a uživatel by udělal tuto interakci.

Posledním metodou je využití vygenerovaného embeddingu, nalezení nejbližší položky k takovému vektoru v kosinové podobnosti a využití embeddingu jako další vstup do neuronové sítě. Všechny tyto tři metody budou otestovány v kapitole 4.

2.6 Vyhodnocování úspěšnosti

Pro sekvenční modely nelze použít leave-one-out způsob testování, kde je vynechána jedna položka z vektoru uživatele a testovat, zdali ji model nedoporučí. V takovém způsobu se dotazujeme na doporučení z minulosti na základě aktuálních dat. To již bylo zmíněno v části 1.1.3.3. K testování však bude stále využita křížová validace, kde je vynechána jedna položka, kterou by měl model doporučit. Vynechaná položka ovšem bude vybírána konkrétním způsobem.

Vynechána je poslední položka a způsob lze tedy nazvat leave-last-one-out, ale nejedná se pokaždé pouze o poslední položku, protože sekvence může být předzpracována. Předzpracování bylo ukázáno na obrázku 2.2 v sekci 2.3. Vy-



Obrázek 2.6: Schematicky zobrazené vybavování modelu a generování doporučení

nechané položky tedy budou například nákupy nebo přidání do košíku (obecně tedy interakce, která má větší váhu), protože ty je cílem predikovat.

Tento způsob je náročný na množství dat, protože nelze provést takové množství testů z několika vektorů jako u leave-one-out testování, ale na druhou stranu mnohem lépe odráží chování uživatelů, kde z posledních interakcí je zbytečné se snažit předpovědět nějakou před rokem vynechanou historickou interakci. Model má za cíl předpovědět takto vynechanou položku ze sekvence. Doporučení může mít různou velikost a podle toho se také budou lišit hodnoty. Využito je především doporučování pěti položek (top-5 doporučovací úloha), protože to je počet, který lze jednoduše doporučit a zobrazit uživateli, aby si mohl vybrat.

Hodnoty budou zapisovány do matice záměn, ze které lze spočítat recall, který pro mě bude představovat základní metriku. Druhou metrikou bude catalog coverage popsaný v 1.1.3.3, který měří procento doporučených položek a může ukazovat, jak diverzifikované doporučení model poskytuje. Model, který doporučuje pouze nejpopulárnější položky, může mít vysoký recall, ale velmi nízkou catalog coverage a neumožnuje uživatelům prozkoumávat nějaký další obsah na dané doméně. Výběr takového modelu pro doporučovací systém proto nemusí být ideální, i když by měl nejvyšší recall.

Kapitola 3

Realizace

V této kapitole jsou uvedeny implementační detaily, které nebyly zmíněné v předchozí kapitole. Na začátku kapitoly jsou zmíněny použité technologie a knihovny, na které je navázáno optimalizacemi při trénování neuronové sítě. Poté jsou stručně popsány a uvedeny implementované modely, se kterými je model porovnáván. Na závěr je v této kapitole zmíněn formát souborů pro uládání dat a nejdůležitější části konfigurace jednotlivých algoritmů a vyhodnocování.

3.1 Použité technologie a knihovny

Všechny algoritmy byly implementované v programovacím jazyce Python. Jedním z důvodů výběru Pythonu jako programovacího jazyku byla efektivita a rychlost tvorby v tomto jazyce. Druhým a důležitějším faktorem byla existence knihoven pro strojové učení, které jsou pro Python jednoduše k dispozici.

Knihovny poskytují algoritmy, které byly popsány v kapitole 1. Tyto algoritmy jsou většinou tvořeny v jiném programovacím jazyce s API v Pythonu z důvodu rychlosti provádění operací. V práci jsou využity následující knihovny:

Numpy je knihovna pro práci s hustými vektory a maticemi [44]. Manipulace s vektory a maticemi je pomocí této knihovny velice rychlá. V diplomové práci je knihovna využita pro uložení embeddingů pro položky a také pro vstupní data k trénování neuronové sítě.

Scipy je knihovna, ve které je implementováno mnoho algoritmů a také je zde implementace uložení řidkých matic a vektorů [45]. V práci byla z knihovny využita reprezentace matic v řidké formě pro maticovou faktorizaci, protože husté matice těchto rozměrů by nebylo možné vůbec vytvořit nebo dokonce udržovat v operační paměti.

Dále z této knihovny byl využit algoritmus pro počítání kosinové podobnosti, a to jak pro vektory, tak i pro matici, kde jsou vektory uloženy v řádcích a podobnost je potřeba spočítat pro každý vektor s každým pro následné rychlé nalezení těch nejbližších k jednomu z nich v prostoru embeddingů.

Implicit je knihovna pro kolaborativní filtrování nad datasety s implicitní zpětnou vazbou. Data do této knihovny jsou reprezentována řidkou formou pomocí scipy [24]. Tato knihovna je v práci využita pro maticovou faktorizaci. Všechny metody včetně MF jsou v této knihovně implementované v Cythonu a OpenMP pro maximální paralelizovatelnost a využití všech jader procesoru. Dále je možné některé aloritmy pouštět také na GPU. Mezi tyto algoritmy patří i maticová faktorizace, kterou je možné tímto způsobem mnohonásobně zrychlit.

Tensorflow (1.14) je knihovna používaná pro tvorbu a trénování neuronových sítí [46]. Knihovna tensorflow umožnuje vytváření výpočetního grafu reprezentujícího neuronovou síť a takto vytvořenou síť je možné pomocí této knihovny učit. Učení je možné provádět na CPU, ale také na GPU pro vysokou efektivitu, a nebo dokonce distribuovaně na více GPU.

Nevýhodou této knihovny může být občas problemnatická kompatibilita různých verzí a to především kompatibilita s novou verzí 2.0 a příslušných CUDA knihoven pro využití GPU. Pro částečné řešení tohoto problému je proto využito následující knihovny.

Keras (2.3.1) je API pro tvorbu modelů neuronových sítí na vyšší úrovni [47]. Modely popsané v tomto API jsou částečně nezávislé na konkrétní použité implementaci. Knihovna pro tvorbu a trénování sítě využívá v této práci jako implementaci knihovnu tensorflow. Keras jako implementaci nízkoúrovňových metod umí využít i jinou implementaci než pouze tensorflow bez nutnosti větších zásahů do modelu neuronové sítě.

Keras tedy nepracuje přímo s tensory reprezentující data a jejich tokem a derivacemi, ale k těmto operacím využívá již zmíněné implementace. Modely jsou tedy vytvářeny skládáním připravených vrstev a nelze v této knihovně vytvořit libovolný výpočetní graf. To ale není cílem této práce a ve většině případů se vždy pracuje s neuronovými sítěmi jako s propojenými vrstvami neuronů.

3.2 Optimalizace přípravy dat pro RNN

Výše zmíněné knihovny poskytují implementaci, kterou již není ve většině případů nutné nějak výrazně optimalizovat z důvodu výpočetních prostředků. Jedinou otimalizací, která byla pro model vytvořena, je příprava dat pro neuronovou síť. Pro trénování jsou využity mini batche 1.3.3.1 zvoleného rozměru, počet kroků (délka sekvence) je také pevně zvolen a velikost embeddingu je

také stejná. Data pro jednu dávku jsou potom rozměru $x \times n \times N_0$, kde N_0 je velikost vstupní vrstvy, n je délka sekvence a x je velikost mini batche.

Výhodou je, že na jednotlivých dávkách se může síť učit paralelně. Nevýhodou je vysoká paměťová náročnost, protože těchto dávek může být k trénování mnoho a to až v řádech tisíců. Tento proces lze optimalizovat tím, že nemusí být uchovány sekvence nahrazené embeddingy, ale pouze sekvence identifikátorů položek popsané v 1.1.2.2.

Data jsou tedy ukládána pouze jako sekvence, jejichž počet je předem známý. V knihovně keras existuje možnost trénování modelu s využitím struktury, která implementuje dvě metody. První medota vrací celkový počet dávek. Druhá metoda vrátí konrétní dávku podle indexu. Data je tedy možné připravovat až v moment vyžádání učícím algoritmem. Pro trénování je nutné zaručit změnu pořadí dávek, aby se výsledná síť nepřeučovala na jedno pořadí vzorků. Tuto funkcionalitu zajišťuje keras náhodně, ale lze ji provádět i systematicky a implementovat libovolně.

Při paralelním trénování je síť trénována na několika dávkách na GPU, zatímco jiná vlákna využila vytvořenou strukturu k získání dávky s konkrétním indexem. Jednotlivé sekvence v dávce jsou předzpracovány do husté matice a id položek je nahrazeno jejich embeddingy a zpracovanými kontextovými daty. Toto předzpracování probíhá na CPU a lze tedy efektivně využít plný potenciál hardwaru a zefektivnit trénování.

Pro opravdu velká data mohou být i sekvence načítány například ze souboru nebo databáze. Tento způsob může zpomalit trénování, pokud načítání trvá delší dobu než trénování a není k dispozici příslušný počet jader CPU pro načítání dat oproti rychlosti trénování.

3.3 Implementované modely ve frameworku pro vyhodnocování

Vedle algoritmu pro tvorbu rekurentních neuronových modelů a jejich vyhodnocování byly implementovány následující algoritmy pro porovnání vysledných modelů (jedná se především o metody kolaborativního filtrování):

- **user-knn** – Standartní algoritmus kolaborativního filtrování využívající podobnost uživatelů, pro kterou je nejčastěji použita kosinová podobnost. V algoritmu je implementováno prořezávání možných kandidátů. Z podobných uživatelů je potom sestaveno doporučení z položek, se kterými uživatel neinteragoval.

- **item-knn** – Další příklad kolaborativního filtrování, kde jsou hledány podobné položky k těm, se kterými uživatel interagoval. Mezi položkami je spočítáno skóre, které může představovat jejich podobnost. Položky s největší hodnotou jsou uživateli doporučeny. Pro počítání podobnosti

dvou stejných položek je implementovaná cache, aby nebylo nutné stejné podobnosti počítat opakovaně.

- **precomputed-item-knn** – Stejná varianta jako výše zmíněné item-knn jen s tím rozdílem, že jsou v souboru uloženy ke každé položce nejbližší položky s hodnotou skóre. To je v práci využito pro item-knn na embeddingy z MF, protože počítání opakovaných podobností nad hustými vektory je výpočetně náročné.
- **popularity** – Jednoduchý model doporučující pouze nejpopulárnější položky.
- **reminder** – Model opakující obsah, který již uživatel viděl.

3.4 Struktura a formát souborů k ukládání dat

Jak je možné vidět z obecné struktury algoritmu na obrázku 2.1, k ukládání dat je využito několik souborů. Některé z těchto souborů mohou být vyměněny za libovolný jiný datový zdroj (např. databázi nebo nějaké distribuované uložiště pro velká data).

Jediným souborem, který nelze vyměnit, je serializace natrénovaných neuronových sítí podle standardu v knihovně keras v souboru ve formátu hdf5. Tento soubor obsahuje kromě vah i další užitečné údaje jako jsou architektura neuronové sítě, detaily kompilace (ztrátová funkce a metriky) a stav optimalizátoru. Data jsou využita například k tomu, že lze tento soubor načíst a pokračovat v trénování nebo síť vyhodnocovat.

Dalšími soubory jsou ty s interakcemi, které tvoří trojice *(id uživatele, id položky, časová známka, typ interakce)*. V souboru s rozdělením uživatelů jsou uvedeni pouze uživatelé jako jednotlivé řádky. Pro embeddingy je využit formát, kde je uveden identifikátor položky a poté jsou čárkou oddělené hodnoty embeddingů.

Pro uložení podobností pro embeddingy je v řádku opět uveden identifikátor položky a poté je text ve formátu JSON: $[\text{id} : 384, \text{rating} : 0.87, \ldots]$. Pro konfigurační soubory je zvolen strukturovaný formát yaml a ukázka celého konfiguračního souboru s výchozími hodnotami je uvedena v příloze C.

3.5 Základní konfigurace běhu frameworku

Ke konfiguraci běhu je využit yaml soubor, který obsahuje jednotlivé sekce zajišťující konfiguraci jednotlivých modelů. Zde jsou uvedeny pouze základní parametry a zbývající lze vidět v konfiguračním souboru v příloze C. V první části jsou obecná data, která určují elementární parametry běhu a hlavně strukturu, ve které budou ukládána všechna data od rozdělení po výsledky:

- **experiment-name** – Název sloužící především pro strukturalizaci dat v souborovém systému, např. pro více různých rozdělení uživatelů na trénovací, validační a testovací množinu.
- **dataset** – Název datasetu použitého pro experimenty.
- **interaction-data-file** – Názvy souborů s interakčními daty, které jsou odděleny čárkou.

Další částí je konfigurace načítání a předzpracování dat.

- **days** – Určuje maximální počet dní, jak může být stará interakce a představuje vertikální škálování pro velké datasety, kde je k dispozici i více jak rok dat.
- **number-of-train-users** – Počet trénovacích uživatelů, kteří jsou zvoleni náhodně z celé množiny pro trénování. Tento parametr představuje horizontální škálování pro velké datasety, které může být u navržené RNN vhodnější, protože RNN potřebuje celé sekvence uživatelů a mnoho krátkých sekvencí by nemuselo být vhodné.
- **data-prec** – Předzpracování trénovacích a validačních dat. Jedná se o metody předzpracování, které byly popsány v sekci 2.3.
- **min-ratings-per-user** – Minimální počet interakcí uživatele. Parametr lze specifikovat při vytváření rozdělení uživatelů nebo až při učení modelu.
- **max-ratings-per-user** – Obdoba předchozího bodu určující maximální počet interakcí.

Poslední ze základních sekcí se týká evaluace algorimtů. V této sekci jsou popsány nejdůležitější parametry pro nastavení vyhodnocování.

- **algorithms** Algoritmy oddělené čárkou, které mají být vyhodnocovány.
- **n-test-users** – Počet testovacích uživatelů.
- **top-n** – Počet doporučení, které mají být generované.
- **data-prec** – Předzpracování testovacích dat. Toto je vhodné především pro volbu testování doporučení položky, kterou si uživatel koupí, místo pouhého zobrazení.

3.6 Konfigurace algoritmu generující rekurentní modely

Zde jsou uvedeny nejdůležitější parametry, které lze nastavit při konfiguraci trénování RNN. Všechny zbývající parametry lze nalézt v příloze C.

- **add-weight** – Určuje, zdali má být model rozšířen o váhu dané interakce. Jedná se o přidání kontextových dat. Jsou vhodné pouze tehdy, pokud je v datasetu více rozdílných typů interakcí.
- **add-timestamps** – Pokud nabývá hodnoty *true*, je do modelu přidáno rošíření o časový údaj, který je dále zpracován.
- **use-bins** – Signalizuje použití diskretizace časových údajů využívající binů pevné šířky. Jinak je použita min-max normalizace na časový údaj.
- **per-seq-bins** – Specifikuje způsob diskretizace časového údaje – zdali má být vytvořena přes celý dataset, nebo má být spočítána pro každou sekvenci.
- **bins-exponent** – Exponent ve výrazu 2^x, který určuje počet binů v diskretizaci. Představuje počet neuronů, o které je rozšířena vstupní vrstva.
- **cosine-weight** – Váha kosinové ztrátové funkce, která bude typicky nejvyšší, protože zásadně ovlivňuje přesnost modelu.
- **mse-weight** – Váha MSE na části vektoru, která je představována embeddingem. Může předcházet velkému přeučení pro kosinovou podobnost.
- **timestamp-weight** – Váha chyby na časovém údaji v přidaných kontextových datech.
- **weight-weight** – Váha chyby při využití typu interakce ve ztrátové funci.
- **model-type** – Model běhu popsaný v sekci 2.5.1. Volba tohoto modelu je zásadní i pro vybavování sítě. Při vybavování *many-to-many* jsou vidět výsledky pro jednotlivé kroky.
- **n-rec-cycle** – Počet cyklů při vyhodnocování sítě. Postup byl popsán v sekci 2.5.2 a ukázán na obrázku 2.6.
- **nearest-item-cycle** – Zdali má být embedding vygenerovaný sítí nahrazen embeddingem nejbližší položky.
- **embeddings** – Soubor s uloženými embeddingy, které jsou použity pro daný model. Pro natrénovaný model je vhodné použít vždy stejnou sadu embeddingů, na které byl model trénován.

- **layer-size** – Velikost rekurentních srytých vrstev.
- **n-recurent-layers** – Počet rekurentních strytých vrstev.
- **batch-size** – Velikost mini batche pro trénování NN.

Kapitola 4

Experimenty

V této kapitole jsou představeny provedené experimenty a jejich výsledky. Většina experimentů je prováděna systematicky s cílem ukázat chování modelu v závislosti na jeho architektuře, hyperparametrech nebo způsobech vyhodnocování.

Algoritmy jsou testovány podle způsobu, který by představen v sekci 2.6. Vyhodnocování je prováděno na 5 doporučených položkách (top-5 doporučovací úloha). Metriky vyhodnocování byly popsány v 1.1.3.3. Z nich je využit především recall a catalog coverage.

Výchozí hodnoty pro nastavení algoritmů jsou uvedeny v konfiguračním souboru v příloze C. Hyperparametry, které jsou změněny, jsou uvedeny v jednotlivých sekcích u experimentů, nebo se jedná o hodnoty testovaných parametrů u jednotlivých grafů. Embeddingy jsou vždy generovány na dané datové sadě s příslušným rozměrem a parametry, které byly naměřeny jako optimální.

Nejprve jsou představeny datové sady, na které je navázáno experimenty s MF, která je využita pro tvorbu embeddingů. Poté je pokračováno otestováním zvolené metriky a ztrátové funkce, zdali je možné pomocí navržené ztrátové funkce model trénovat. Následují experimenty zaměřené na architekturu neuronové sítě a její hyperparametry. Dále budou ukázány výsledky při použití jiných optimalizačních metod nebo přidání kontextových dat s jejich vlivem na úspěšnost modelu. V neposlední řadě budou výsledné modely porovnány s ostatními modely zmíněnými v části 3.3.

4.1 Datové sady

Pro experimenty jsou využity tři datové sady. Dvě z nich obsahují pouze anonymní data o interakcích. Popsány jsou pouze jejich základní statistiky na základě interakcí. Třetí datová sada je veřejně dostupná. Datové sady jsou poměrně odlišné, což je možné vidět ze základních statistik z tabulek 4.1 a 4.2.

První datová sada v této práci, označovaná jako D1, je internetový obchod se zařízením do domácnosti. V této datové sadě se mohou vyskytovat relace

Tabulka 4.1: Základní údaje o počtu interakcí, uživatelů a položek pro datové sady a jejich podmnožiny a také hustota interakční matice z nich vytvořená

datová sada	# interakcí	# uživatelů	# položek	hustota
D1	15 711 063	3 428 484	29 761	0.015 %
trénovací	8 095 319	496 751	22 456	0.073 %
validační	441 874	27 674	19 116	0.084 %
testovací	441 730	27 414	18 900	0.085 %
D2	344 809 934	3 625 903	4238	2.182 %
trénovací	209 476 412	1 188 215	1873	9.412 %
validační	11 728 102	66 255	1727	10.250 %
testovací	11 559 430	66 184	1713	10.196 %
yoochoose	34 149 631	9 248 226	52 736	0.007 %
trénovací	19 841 142	3 134 516	46 174	0.014 %
validační	1 098 413	174 138	26 623	0.024 %
testovací	1 104 839	174 381	26 710	0.024 %

anonymních uživatelů, kde jednomu uživateli může odpovídat i více relací. Druhá datová sada D2 představuje obchod s potravinami a je pro tuto práci prioritní. V D2 se nevyskytují případy více relací na jednoho uživatele. Třetí datová sada je veřejně dostupná a byla zveřejněna v rámci RecSys Challenge 2015 [48]. Jedná se o datovou sadu yoochoose, která představuje blíže nespecifikovaný internetový obchod. V yoochoose jsou data v rámci relací anonymních uživatelů. V této práci jsou sekvence mapovány na uživatelské vektory, kde každý takový vektor je představován pouze jednou relací. Relace mohou mít tedy společného reálného uživatele.

V tabulce 4.1 jsou uvedeny počty interakcí, uživatelů, položek a hustota ratingové matice. Data jsou uvedena, jak pro celou datovou sadu, tak i pro vytvořené rozdělení na trénovací, validační a testovací množinu, kde jsou oříznuti uživatelé, kteří mají méně než 3 interakce a také více jak 25 % zobrazených položek, ti jsou uvažováni jako roboti.

Z uvedené tabulky 4.1 jsou patrné informace, ze kterých lze stanovit předpoklady pro jednotlivé datové sady. Tyto předpoklady je možné využít i při návrhu modelů. Vidíme, že součet uživatelů z rozdělených dat netvoří původní velikost datového souboru. Toto je způsobeno vyřazením uživatelů zmíněných výše. Dále je patrné, že byly vyřazeni především uživatelé s málo interakcemi, protože se zvýšila hustota dat ve vytvořených podmnožinách.

Počet položek by měl být ideálně neměnný, ale toho by bylo možné docílit až stejně velkými množinami rozdělení. Přesto je možné vidět rozdíly mezi vybranými datovými sadami. Především mezi D1, D2 a veřejným yoochoose, kde rozdíl počtu položek v trénovacích a testovacích datech je poměrně velký. Tento rozdíl může signalizovat, že se zde vyskytuje mnoho položek s málo interakcemi. Bude zde problém se studeným startem u doporučovacího systému,

Tabulka 4.2: Statistika délek sekvencí uživatelů pro jednotlivé datasety

datová sada	průměr	$P_{25\%}$	$P_{50\%}$	$P_{75\%}$	$P_{90\%}$	$P_{95\%}$	$P_{99\%}$
D1	16.39	5	8	15	31	51	140
D2	176.24	85	148	234	342	425	632
youchoose	6.32	3	5	7	11	15	28

který v této práci není řešen. Mohlo by být vhodné použít nějaké techniky na řešení tohoto problému.

Další užitečnou statistikou může být počet interakcí uživatelů. Tato statistika je využita při volbě délky sekvence pro rekurentní neuronovou síť. Průměrná hodnota a jednotlivé percentily jsou zobrazeny v tabulce 4.2.

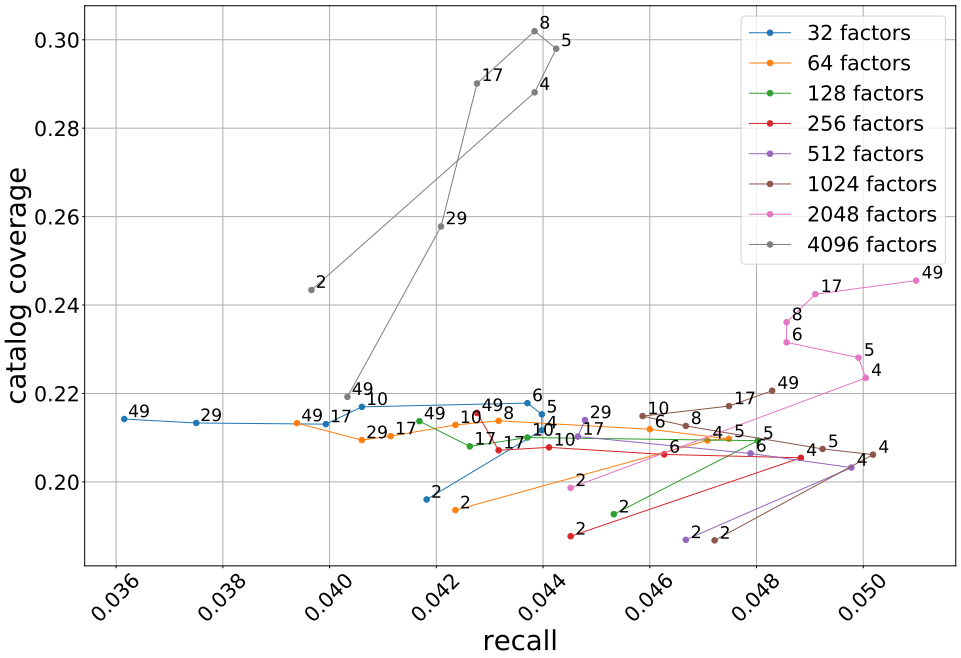
Z tabulky 4.2 je patrné, že pro datovou sadu D2 existuje mnoho uživatelů, kteří se pravidelně vrací a nakupují větší množství produktů. Na této datové sadě budou prováděny experimenty s měnící se délkou sekvence pro neuronovou síť a pozorovat její vliv na skutečnost, jak dlouhé sekvence se vyplatí využívat při trénování RNN. Očekáváním je, že čím delší budou existující sekvence, tím lépe bude možné neuronovou síť naučit.

Na veřejné datové sadě yoochoose je vidět, že jsou sekvence velice krátké. Zde by mohl být ve velké výhodě opakovací (reminder) model, protože bude možné v průměru použít téměř celou historii při top-5 doporučování. Sekvence představují relace v nějakém e-shoopu a pravděpodoně podle její délky se jedná vždy o jeden přístup na danou doménu. Uživatelé nemají déle trvající relace. Na této datové sadě by také mohl být problém s vytvořením kvalitních embeddingů, protože MF, jako algoritmus standartního kolaborativního filtrování, pravděpodobně nedá tak dobré výsledky.

4.2 Maticová faktorizace a kvalita embeddingů

První experimenty jsou zaměřeny na maticovou faktorizaci. K jejímu posouzení jsou použity všechny datové sady, protože představuje klíčovou část pro vytvoření reprezentace položek pro neuronovou síť. Pro MF je použit model vyhodnocování jako v případě algoritmu item-knn. Vektory latentních příznaků jsou použity jako vektory, které reprezentují položky. Tento model je popsán mezi ostatními implementovanými modely v sekci 3.3.

MF je vyhodnocována pomocí leave-last-one-out techniky a metrik recall a catalog coverage. Pro datasety D1 a D2 je cíl předpovědět následující položku, kterou si uživatel koupí. U veřejného yoochoose datasetu je pro porovnání modelů vynechána podmínka predikce nákupu a předpovídána následující položka představovaná libovolnou interakcí. V tomto datasetu je velmi málo nákupů, krátká historie uživatelů, a proto úspěšnost při predikci následujícího nákupu pomocí MF nebyla téměř měřitelná. Na kvalitu embeddingů by



Obrázek 4.1: Porovnání výkonnosti modelů s různým počtem latentních příznaků. Ukázán vliv na chování modelu při změnách počtu nejbližších sousedů. Provedeno na databázi D1

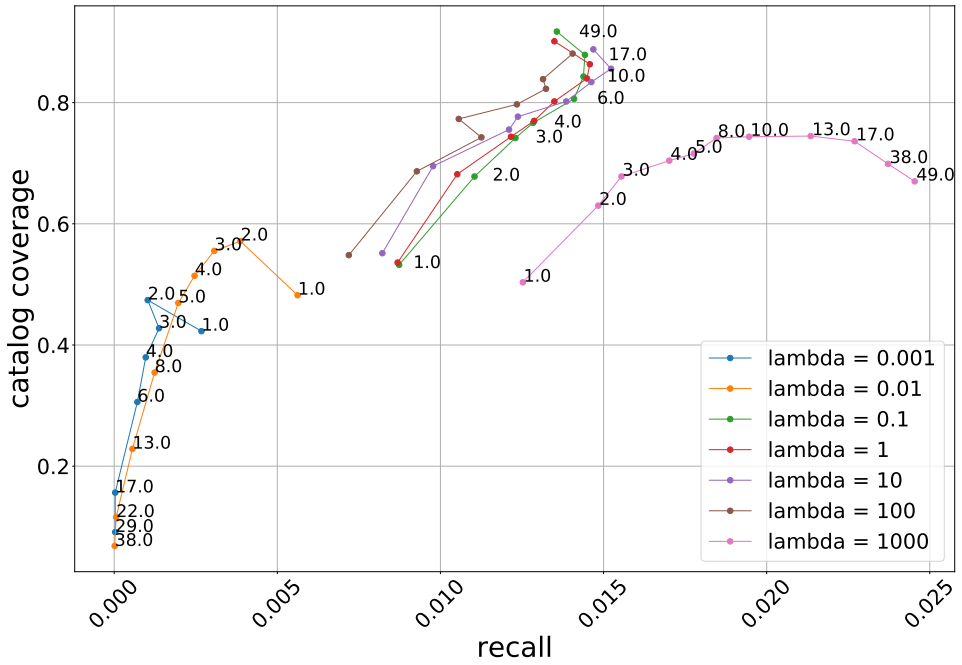
tato skutečnost neměla mít žádný vliv. Cílem je měřitelným způsobem embeddingy porovnat a vybrat alespoň nějaké použitelné pro neuronovou síť.

Z tohoto důvodu jsou u D1 a D2 mnohem nižší hodnoty recallu, protože je predikce následujícího nákupu mnohem náročnější úlohou než predikce dalšího zobrazeného produktu. V závěrečném porovnání ale výhoda není ponechána a cílem je měřit úspěšnost při doporučování následujícího nákupu, proto model MF není na grafu uveden.

4.2.1 Chování MF vůči počtu uvažovaných sousedů

U MF jsou nejprve prozkoumány závislosti na hodnotách k a na hodnotách některých hyperparametrů. Sledován je jejich vliv na výkonnost. Na prvním grafu 4.1 jsou zobrazeny hodnoty k pro jednotlivé modely s různým počtem latentních příznaků pro datovou sadu D1. Viditelný je trend, že maximum recallu je pro $k = 4$. Dále s rostoucím počtem faktorů se také zvyšuje recall při přibližně stejné hodnotě k.

S roustoucím počtem faktorů, kde je jejich počet 1024 a více, se situace mění. Pro model 1024 sice výkonnost klesá, ale pro vyšší k opět roste a obdobně pro 2048 příznaků. Pro 4096 příznaků už je vidět výrazný pokles výkonu a s rostoucím k klesá různorodost doporučovaných položek. Toto chování může



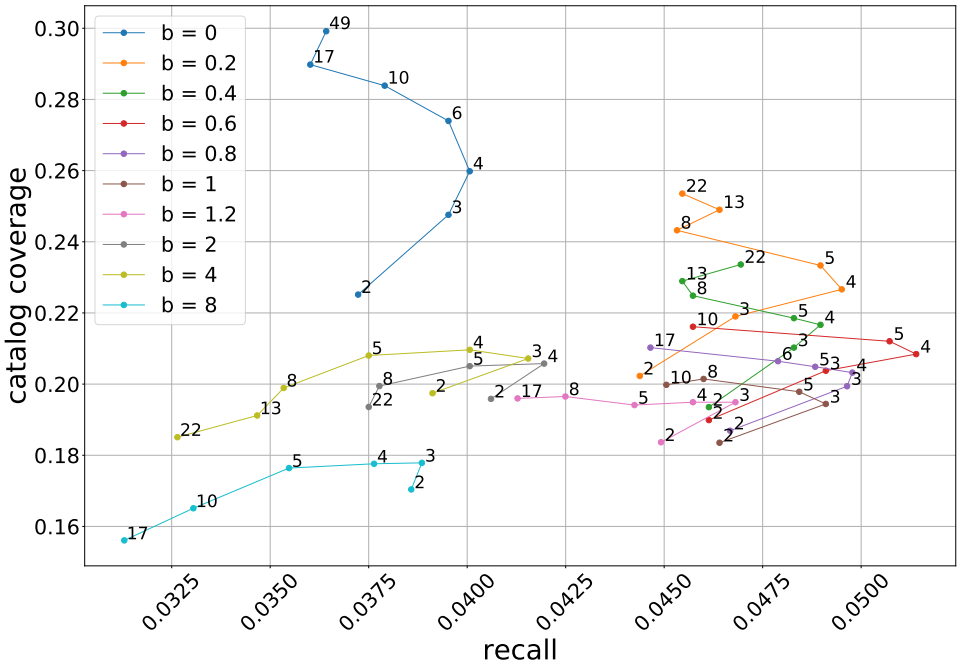
Obrázek 4.2: Vliv hodnoty regularizace na chování modelu vůči uvažovanému počtu nejbližších sousedů. Zobrazené měření je provedeno na databázi D2. Parametr regularizace je v grafu označen jako lambda

ukazovat na přeučení modelu s mnoha faktory. Tento experiment byl proveden na datové sadě D1 a predikci následujícího nákupu.

Na grafu 4.2 je vykresleno chování modelů na různou hodnotu regularizace pro měnící se k. Tento experiment byl proveden pro dataset $D2$, protože na tomto datasetu má regularizace největší vliv. Je možné vidět zajímavé chování pro nízké hodnoty, kde velice rychle klesá úspěšnost. S rostoucím k tyto modely nejsou úspěšné.

Pro většinu hodnot regularizace (0.1–100) jsou hodnoty velice blízko sebe. Toto chování ukazuje na nějaký existující práh regularizace, který je nutný pro úspěšný model a v těchto hodnotách se tento model chová vůči k jako typický algoritmus pro hledání k nejbližších položek. Nejzajímavější chování je pro vysokou hodnotu regularizace, kde s rostoucím k klesá coverage a roste recall a to poměrně znatelně. Toto chování může být způsobeno vysokou hustotou datové matice, kde i s vysokou regularizací se model učí a dosahuje lepších výsledků. Pro takto vysokou hodnotu regularizace MF pro datové sady D1 a yoochoose vůbec nefungovala.

Na posledním grafu 4.3 je ukázán vliv parametru b z vážení BM25 popsaném v sekci 2.2. Vliv tohoto parametru souvisí s velikostí vektorů a jejich vlivu na úpravu datové matice. Nízké hodnoty nějak výrazně neupravují



Obrázek 4.3: Porovnání modelů v rovině recall-coverage s různou hodnotou b v BM25 vážení a vliv parametru na změnu chování vůči uvažovanému počtu nejbližších sousedů. Měření je provedeno na databázi D1

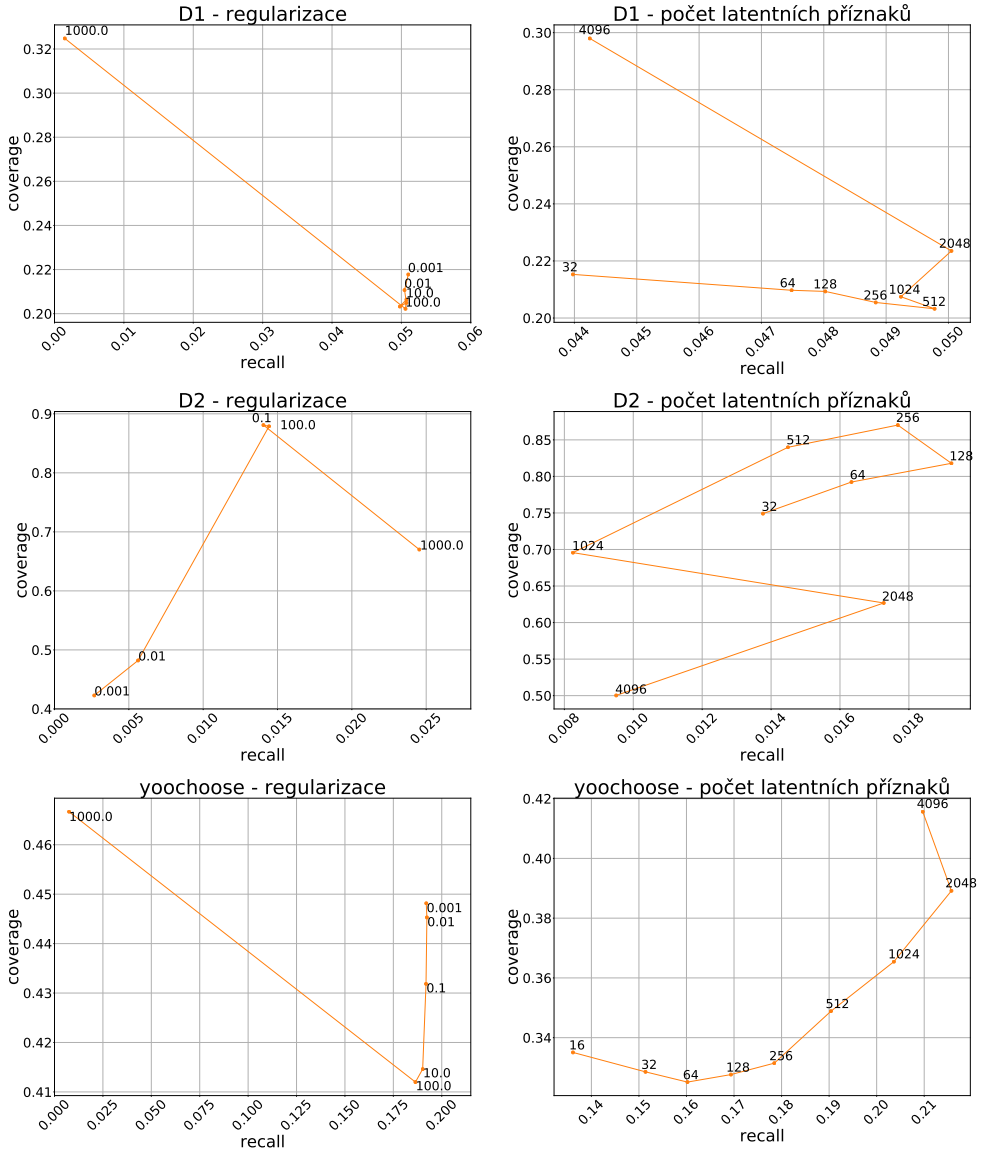
chování algoritmu a s rostoucím k roste podle očekávání i catalog coverage. Zajímavé je, že s rostoucí hodnotou b se toto chování mění a volba velkého k může vést k horším výsledkům pravděpodobně z důvodu přeučení.

Parametr m a prořezávání není ukázano, protože zde hodnota výrazně neovlivňuje chování modulu vůči hodnotám k, ale ovlivňuje výkonnost pouze celkově. Vliv bude sledován v následující podkapitole.

4.2.2 Vyhodnocení chování MF při změně klíčových hyperparametrů

Na obrázku 4.4 jsou zobrazeny grafy pro počet latentních příznaků a regularizaci, jejíž vliv je popsán jako první. Pro datasety D1 a yoochoose je vidět, že vyšší hodnoty regularizace mají spíše negativní vliv a pro vysoké hodnoty (~ 1000) je faktorizace téměř s nulovým recallem. Zvětšující se hodnoty regularizace snižovaly coverage modelů. Na datasetu D2 je chování odlišné. Je patrné, že zde regularizace má pozitivní vliv a je vhodné ji využít s vyššími hodnotami. Chování je pravděpodobně způsobeno vysokou hustotou dat.

Na grafech v pravém sloupci je ukázáno chování vůči počtu latentní příznaků. U všech grafů je vidět, že pro nějakou hodnotu příznaků se model začíná



Obrázek 4.4: Porovnání vlivu parametrů regularizace a počtu latentních příznaků pro všechny tři zvolené datové sady na výkonnost v rovině recall-coverage

přeučovat a jeho výkonnost klesá. Pro dataset D1 je patrné, že se optimální hodnoty pohybují mezi 512 a 2048. Pro dataset D2 bych za optimální hodnotu označil 128. Na grafu je ovšem vidět vyšší hodnota pro 2048 faktorů, která může představovat vychýlenou hodnotu, které nemusí být dosaženo při opakovaném trénování. Opakované výpočty pro tuto hodnotu nebyly prováděny pro vysokou náročnost v případě velkého počtu latentních příznaků a velikost datové sady D2.

Na grafu 4.5 jsou vykresleny zbylé parametry. Jak je možné vidět, parametr m popsaný v 2.2 nemá příliš velký vliv na výkonnost, ale jeho vyšší hodnoty $m = 5$ nějaké zlepšení přinášejí.

Další parametr b má poměrně vysoký vliv na výkon a z grafů vyplývá, že pro něj existuje volba optimální hodnoty. Z datasetů je patrné, že čím menší je průměrná délka vektorů v datové sadě, tím menší je optimální hodnota parametru a naopak.

Posledním testovaným parametrem je hodnota prořezávání. To je minimální velikost vektoru a to jak řádkového (pro uživatele) tak i sloupcového (pro položky). Kratší vektory jsou odstraněny. Chování je očekávané z popisu dat. U datasetu D1 a yoochoose je vidět optimální hodnota nižší. U D2 výkonnost modelu s prořezem roste, protože průměrná délka vektoru je zde vyšší a odřezáním kratších vektorů se nesnižuje množství obsažené informace.

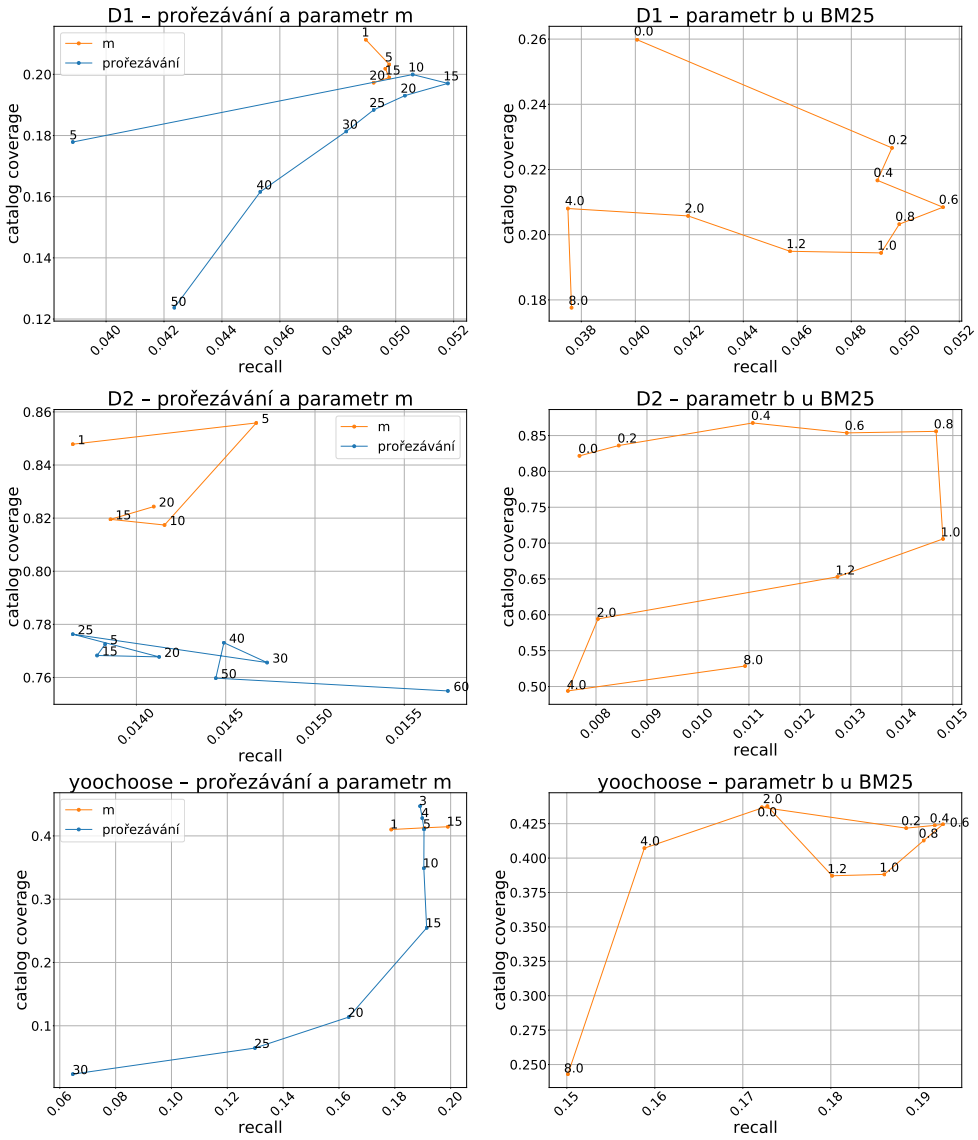
Zajímavý rozdíl je mezi datasetem D1 a yoochoose, kde pro yoochoose se do hodnoty 15 téměř nemění recall, ale pouze klesá coverage, což může ukazovat na mnoho cold-start items. Na uživatelské vektory větší prořezání vliv nemá, protože stále nebyla prořezána podstatná část informace.

4.3 Vliv změn hodnot ztrátové funkce na metriku úpěšnosti

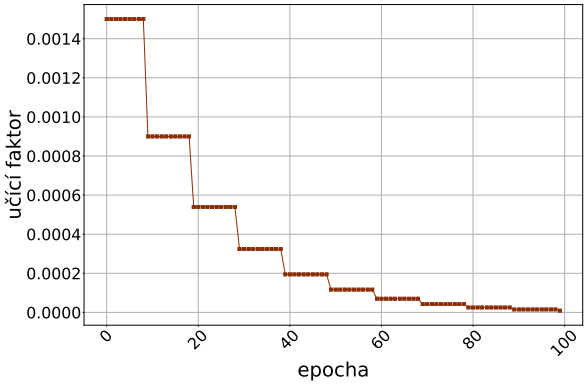
Další experimenty jsou zaměřeny na hodnoty recallu a catalog coverage při učení pomocí zvolené ztrátové funkce, která byla popsána v sekci 2.5. Na těchto experimentech je patrná korelace mezi hodnotami ztrátové funkce a hodnotou recall. Tyto experimenty byly důležité pro ověření ztrátové funkce a možnosti její použití při trénování.

Pro všechny experimenty je použit průběh učícího faktoru zobrazený na obrázku 4.6. Pro datovou sadu D2 a yoochoose je pouze změněn celkový počet epoch, ale průběh učícího faktoru je zachován. Hodnoty na křivce jsou pro jednotlivé epochy, kde snižováním učícího faktoru jsou změnšovány kroky v sestupu pro dohledávání optima. Pomocí takto zvoleného průběhu učícího faktoru v kombinaci s optimalizátorem adam bylo dosahováno tížených výsledků.

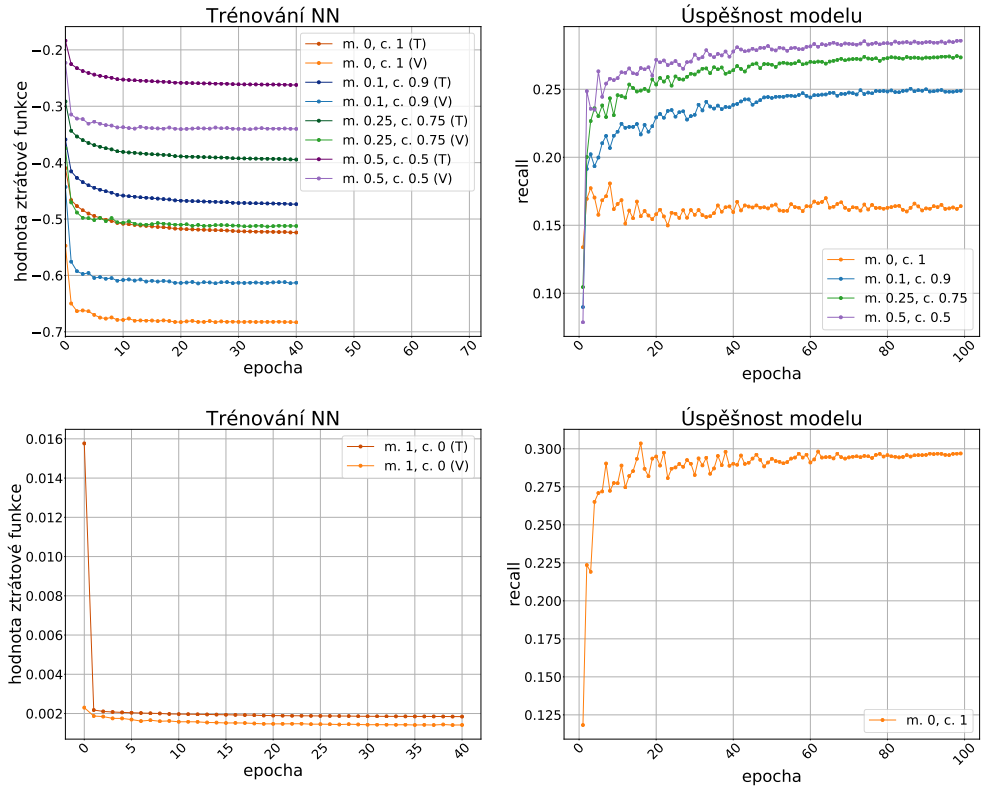
Na obrázku 4.7 jsou zobrazeny grafy pro datovou sadu D1 a vliv hodnot ztrátové funkce. Pro model byla použita velikost jediné skryté vrstvy 256 a velikost embeddingu shodně 256. Dále maximální délka sekvence pro model je zvolena 10, kde vycházím z dat tabulky 4.2.



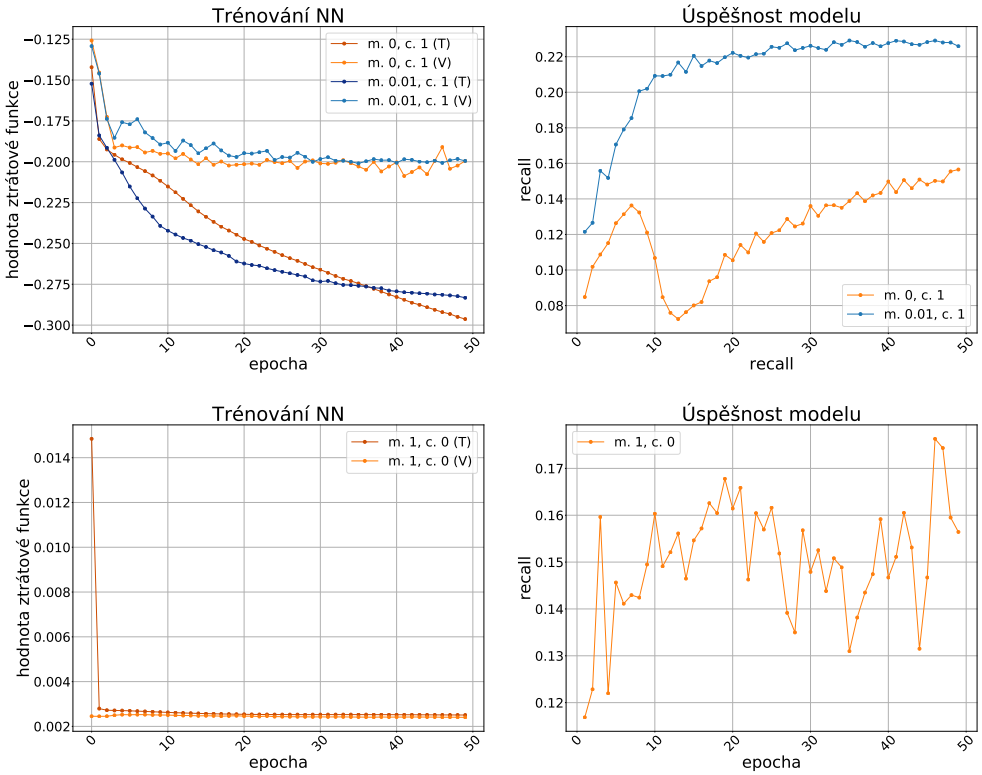
Obrázek 4.5: Porovnání vlivu parametru b u BM25, parametru m, který zvětšuje hodnoty v ratingové matici, a porovnání vlivu velikosti prořezávání na výkonnost modelu pro všechny zvolené datové sady



Obrázek 4.6: Učící faktor v průběhu učení



Obrázek 4.7: Vliv změny vah ve ztrátové funkci na učení a úspěšnost modelu, kde m. znamená hodnotu pro váhu MSE složky ztrátové funkce a c. je část obsahující kosinovou podobnost. Testováno na datasetu D1



Obrázek 4.8: Vliv změny vah ve ztrátové funkci na učení a úspěšnost modelu, kde m. znamená hodnotu pro váhu MSE složky ztrátové funkce a c. je část obsahující kosinovou podobnost. Testováno na datasetu D2

Vlevo na obrázku jsou zobrazeny grafy pro trénovací a validační chybu. Hodnoty, kde je uvažována pouze MSE, jsou zobrazeny na grafu níže z důvodu veliké odlišnosti velikosti hodnot pro MSE a pro kosinovou ztrátovou funkci.

Na grafu 4.7 vlevo nahoře je patrné, že hodnota složky MSE nemá příliš vliv na výsledné hodnoty ztrátové funkce, ale hlavní složku tvoří kosinová podobnost. Průběh učení je velice podobný pro všechny modely obsahující kosinovu podobnost. Hodnoty se snižují se snižující se vahou kosinové části, což je očekávané, ale na průběh učení velikost hodnot nemá vliv.

U výsledků je to ovšem jiné a je zde vidět, že model trénovaný bez použití MSE složky má tendenci k přeučení a použití MSE pomáhá k zlepšení úspěšnosti modelu. Při pohledu na graf, kde je použito pouze MSE je vidět, že bylo dosaženo nejlepších výsledků a použití kosinové podobnosti je na tomto datasetu dokonce nevýhodné. Toto ale neplatí pro všechny datasety. Při učení pomocí MSE je také patrné, že se model učil velice rychle a během několika epoch již dosáhl vysoké hodnoty recallu.

Na obrázku 4.8 jsou zobrazeny závislosti na hodnotách ztrátové funkce pro dataset D2. Na datasetu D2 je použita velikost jediné skryté vrstvy 512, velikost embeddingu 256 a maximální délka sekvence je omezena na 50. Kratší sekvence jsou doplňovány nulami.

Na tomto datasetu není otestováno tolik různých hodnot z důvodu jeho velikosti. Na dolních grafech pro MSE je vidět, že pouze pomocí této ztrátové funkce se model nedaří dostatečně úspěšně trénovat. Toto chování může být způsobeno délkou sekvencí uživatelů, která je ukázána v tabulce 4.2 v sekci 4.1. Model uvažuje sekvence délky 50 oproti 10 v datové sadě D1.

Při pohledu na graf, kde je použita pouze kosinova ztrátová funkce je vidět určitá nestabilita při učení, která může být také způsobena volbou optimalizátoru adam, který učení velice urychluje, ale někdy to může být na úkor úspěšnosti a přesnosti. Při kombinaci obou částí ztrátové funkce se podařilo dosáhnout nejlepších výsledků a tato kombinace pro tento dataset funguje.

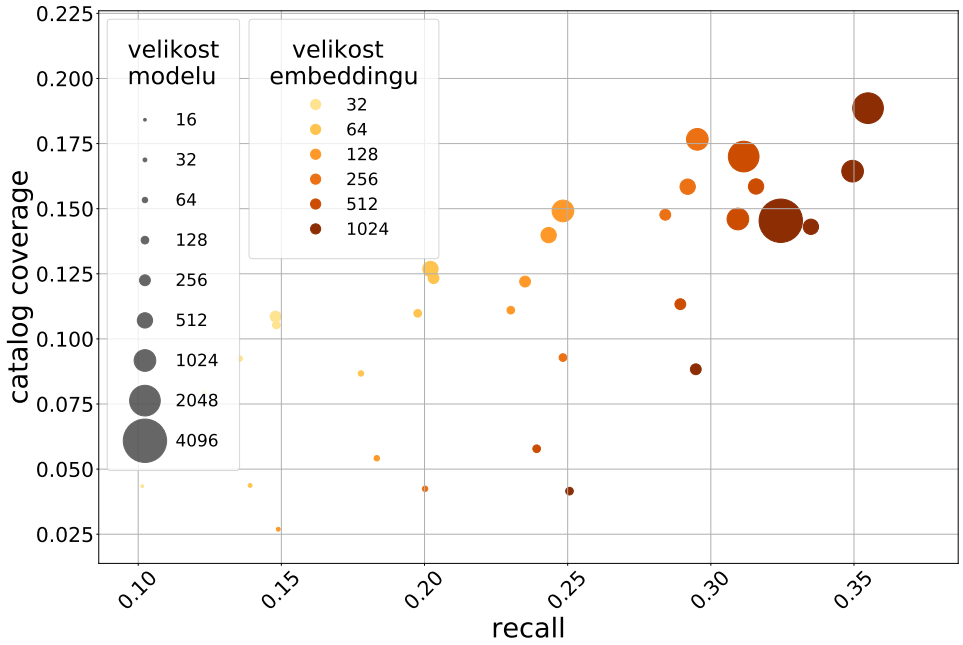
Toto chování se děje pravděpodobně z důvodu struktury dat a délky sekvence, kde pro datovou sadu D1 se model může učit precizně hodnoty embeddingů, protože je zde více penalizována větší odchylka a průměrná délka sekvence na datasetu je $16,39$. Pro velkou datovou sadu D2 (průměrná délka sekvence $176,24$) může kosinová část ztrátové funkce pomáhat modelu generalizovat v prostoru embeddingů, které byly vytvořeny pomocí MF. MSE část potom může pomáhat při udržení nižších hodnot predikovaných embeddingů, které lze v síti opětovně použít při vybavování.

4.4 Vliv velikosti NN a počtu faktorů u embeddingů

V předchozí sekci bylo ukázáno, že ztrátová funkce koreluje s použitou metrikou a lze tedy pomocí této metriky učit modely. Tato sekce je zaměřená na experimenty s vlivem velikosti modelu a velikosti embeddingu na výkonnost modelu.

Očekávaný průběhem je, že s rostoucí velikostí sítě a embeddingu poroste také úspěšnost modelu, ale ne vždy může být výhodné mít co největší model. S rostoucí velikostí embeddingu roste také velikost vstupní vrstvy a počet učených parametrů, které rostou i při zvyšující se velikosti modelu (skryté LSTM vrstvy) a je zpomalováno učení. Rostou nároky na velikost operační a grafické paměti. Také se zpomaluje rychlost vybavování výsledného modelu. Velký model také může mít tendence se přeučovat a nedostatečně generalizovat.

Cílem by bylo najít způsob, jak volit velikost embeddingů a velikost modelu a tím dosáhnout odpovídající úspěšnosti s dostatečnou generalizací tak, aby model byl reálně upočitatelný. Zjistit tedy zdali zvolit větší embedding k menší velikosti sítě nebo naopak.

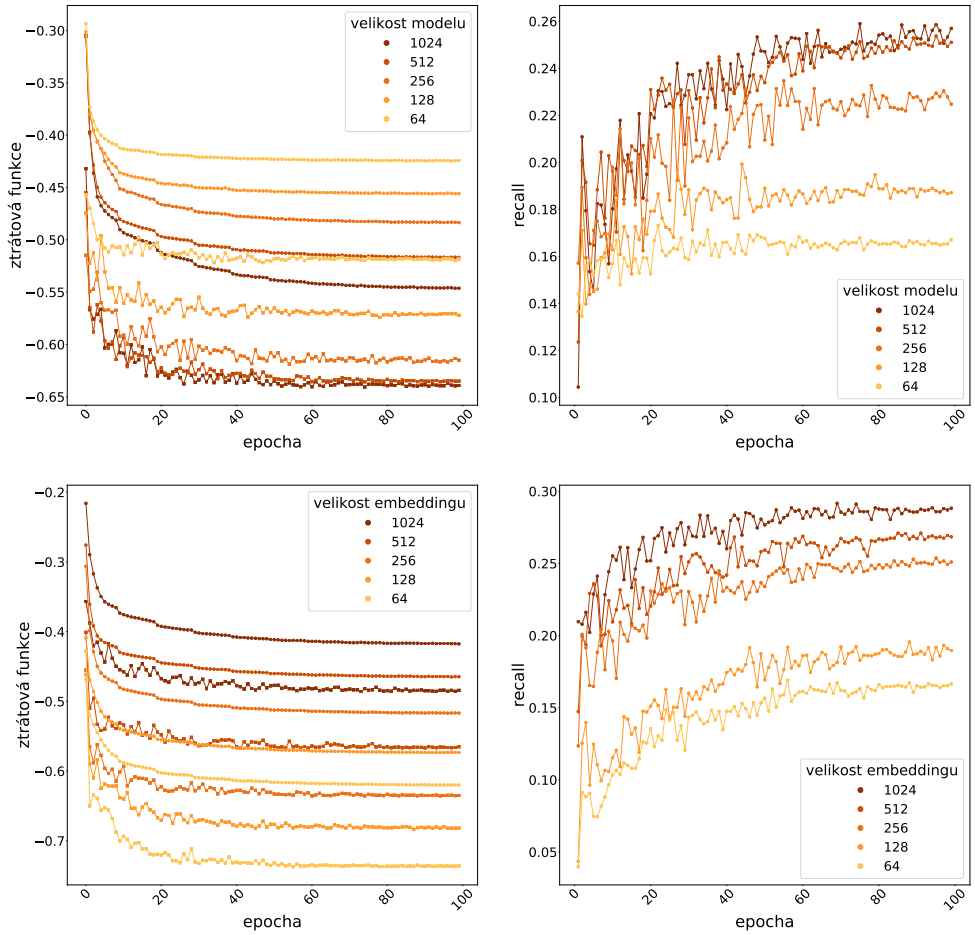


Obrázek 4.9: Graf vlivu velikosti embeddingu a velikosti modelu na jeho úspěšnost v rovině recall-coverage

Počet učených parametrů, od kterých se také odvíjí velikost sítě, počet nutných operací, atd. . . , lze podle velikosti sítě a embeddingu spočítat a vyjít přitom z teorie popsané v sekci 1.3.1.1. Je možné vidět, že jsou zde matice vah **W**, **U** pro rekurentní vazby a **b** pro vychýlení. Tyto matice jsou potřeba pro každou zmíněnou uzávěrkovou bránu a vnitřní stav. Pokud označíme velikost vstupu (embeddingu) jako m a počet neuronů LSTM vrstvy (velikost modelu) n, tak počet učených parametrů je dán vztahem $4(mn + n^2 + n)$. K tomu je nutné připočíst ještě váhy z poslední dopředné vrstvy nm, protože počet neuronů v poslední vrstvě se rovná m. Nárůst učených parametrů sice není exponenciální, ale polynom má vysoký exponent. Pro porovnání počtu učených parametrů modelu s velikostí embeddingu 128 a skryté vrstvy 128 proti velikostem 1024: 147 968 vs. 9 441 280.

Závislosti velikostí a úspěšnosti jsou zobrazeny na grafu 4.9. Jasně jsou zde patrné některé závislosti. Je možné si všimnout, že sledované parametry modelu mají odlišný vliv na výsledky modelů. Pokud se zaměříme na barvu (velikost embeddingu) pro stejně velké modely, je vidět, že se zvyšuje recall při téměř stejné coverage. Coverage někdy dokonce klesá při vyšších hodnotách, kde embedding je dvakrát větší než model a ani přírůstek recallu již není takový.

Z výše zmíněné závislosti je patrné, že velikost embeddingu musí být odpovídající velikosti sítě. Pokud se podíváme na závislost velikosti modelu a em-



Obrázek 4.10: Zobrazení průběhu ztrátové funkce (levé grafy) a recallu (grafy vpravo) v závislosti na velikosti modelu (horní grafy) a velikosti embeddingu (grafy dole). Na grafech vlevo vždy odpovídají dvě křivky jednomu modelu. Křivka s nižší hodnotou, která není tolik vyhlazená, odpovídá chybě na validačních datech. Naopak vyhlazená křivka odpovídá trénovací chybě

beddingu, je možné vidět, že s roustoucí velikostí modelu roste jak recall tak i coverage, ale pouze do nějaké velikosti. Pokles je patrný pro velké hodnoty obou parametrů. Úspěšnost takových modelů poté klesá nebo stagnuje při několikanásobně vyšším počtu učených parametrů.

Proto je vhodné pro konkrétní velikost embeddingu zvolit i odpovídající velikost sítě a najít tím tížené optimum v rovině recall-coverage pro tyto parametry s ohledem na výpočetní prostředky.

Na obrázku 4.10 jsou ukázány učící křivky, kde jsou zobrazeny hodnoty ztrátové funkce (grafy vlevo) a na grafech vpravo je pro každou epochu spočítán recall. Jak již bylo ukázáno v sekci 4.3 výše, hodnoty recall korelují

se ztrátovou funkcí. Hodnoty recall jsou měřeny na testovacích datech, validační chyba je spočítána na validačních datech bez aplikovaného dropoutu a trénovací chyba je měřena i s vynechanými neurony pomocí techniky dropout na trénovacích datech.

Na grafech lze vidět vliv na učení. Pro velikost modelu je patrné, že není ideální zvětšovat model neustále, protože při zdvojnásobení modelu z 512 na 1024 výrazně vzroste složitost, ale již téměř vůbec neroste recall pro danou velikost embeddingu a ani již není nižší hodnota chyby na validačních datech. Pouze se snížila hodnota chyby na trénovacích datech, takže model se snáze přeučuje.

Pro velikost embeddingů je chování trochu odlišné a je zde vidět jedna zvláštní vlastnost ztrátové funkce, která ovšem mohla být očekávána. Na grafu 4.10 vídíme, že s rostoucím embeddingem roste i chyba (respektive se nesnižuje na tak nízkou hodnotu), ale zvyšuje se recall. Hodnoty ztrátové funkce tedy nelze použít přímo pro porovnání modelů. Stále je ale možné využít hodnoty ztrátové funkce k ukončení trénování, pokud již chyba neklesá tak rychle a již se nevyplatí NN dále trénovat. Učení přímo na hodnotách recall by ovšem nebylo upočitatelné.

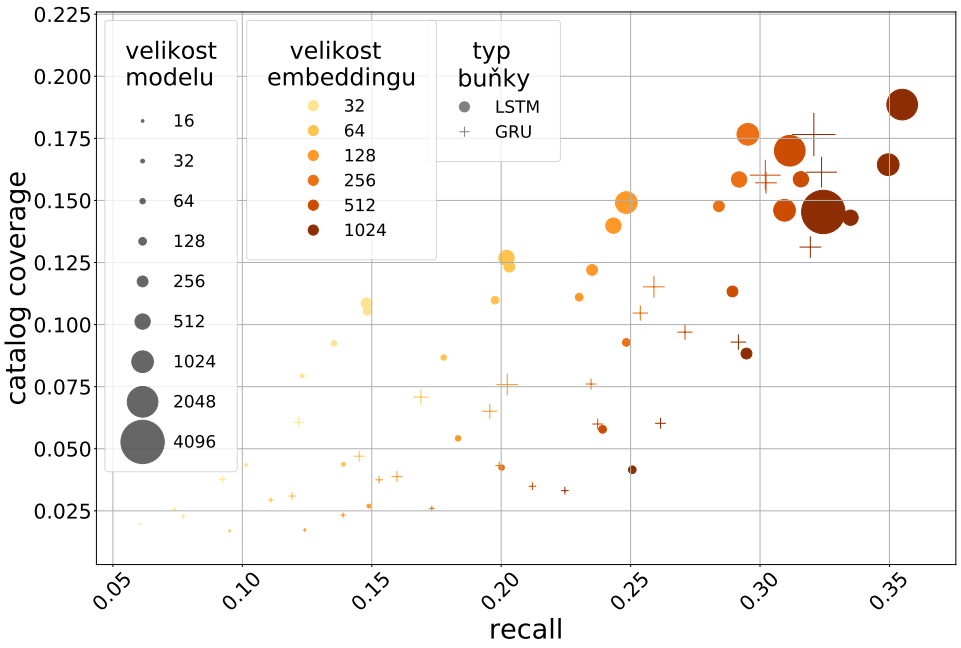
Nárůst chyby pro ztrátovou funkci je pravděpodobně způsoben tím, že je tato chyba počítána vždy v prostoru, který má dvojnásobný počet dimenzí. Neuronová síť se nenaučí tak přesné výsledné hodnoty embeddingů, které se ovšem neprojeví na výsledném recallu, protože velikost prostoru se mnohonásobně zvětšila. Recall pak pravděpodobně reflektuje především rozdělení embeddingů v prostoru, kde více dimenzí může pomoci k vyššímu recallu díky nárůstu vzdálenosti mezi embeddingy.

4.5 Porovnání modelů s LSTM a GRU

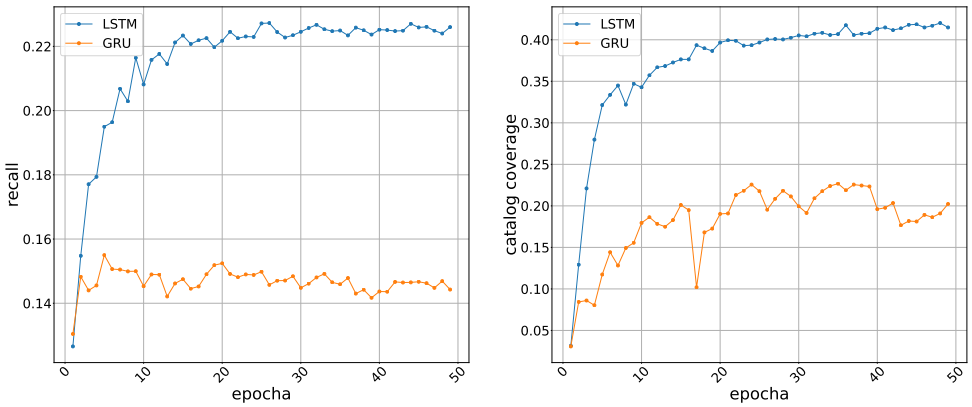
V předchozí sekci 4.4 na obrázku 4.10 byly zobrazeny závislosti úspěšnosti na velikosti modelu a embeddingů. Do tohoto grafu jsou na obrázku 4.11 přidány hodnoty pro modely, kde jsou použity buňky GRU. Buňky GRU jsou otestovány, protože byly využívány a zmiňovány v literatuře a v ní uvedených existujících modelech. Tyto přístupy jsou popsány v sekci 1.3.4.

Na grafu 4.11 je patrné, že modely s GRU nedosahují takových výsledků jako LSTM, ale rozdíl pro datovou sadu D1 není tak výrazný. Modely s GRU by mohli být využity a ztráta na úspěšnosti by nebyla tak velká. Modely, které využívaly GRU, navíc zabíraly méně místa na disku (obsahují méně učených parametrů) a jejich trénování probíhá rychleji.

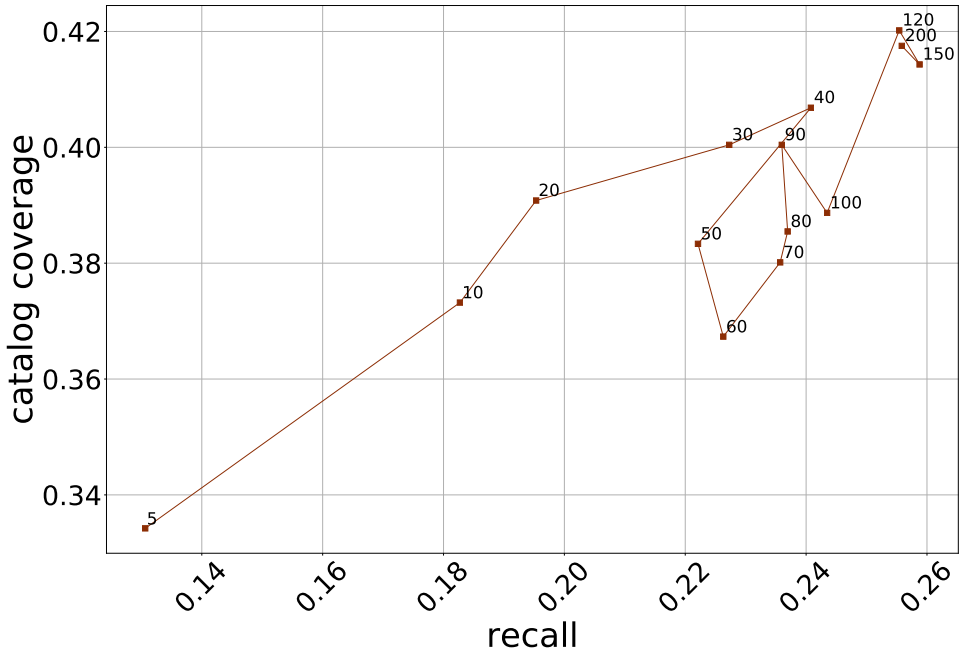
Na obrázku 4.12 jsou naměřeny rozdíly pro datovou sadu D2 pro délku sekvence 70. Na grafech je vývoj metrik při trénování. Na levém jsou hodnoty recall a na pravém hodnoty catalog coverage. Na této datové sadě jsou již patrné větší rozdíly. Je vidět, že GRU má problém s delšími sekvencemi a nedokáže z dat extrahovat příznaky důležité pro doporučování.



Obrázek 4.11: Na obrázku se nachází rozšíření grafu 4.10, kde byly přidány naměřené hodnoty pro modely, kde je použito GRU místo LSTM na datové sadě D1 a GRU je porovnáno s LSTM



Obrázek 4.12: Porovnání modelů s GRU a LSTM buňkami na datové sadě D2, kde byla délka sekvence zvolena 70



Obrázek 4.13: Vývoj úspěšnosti modelu s rostoucí maximální délkou sekvence

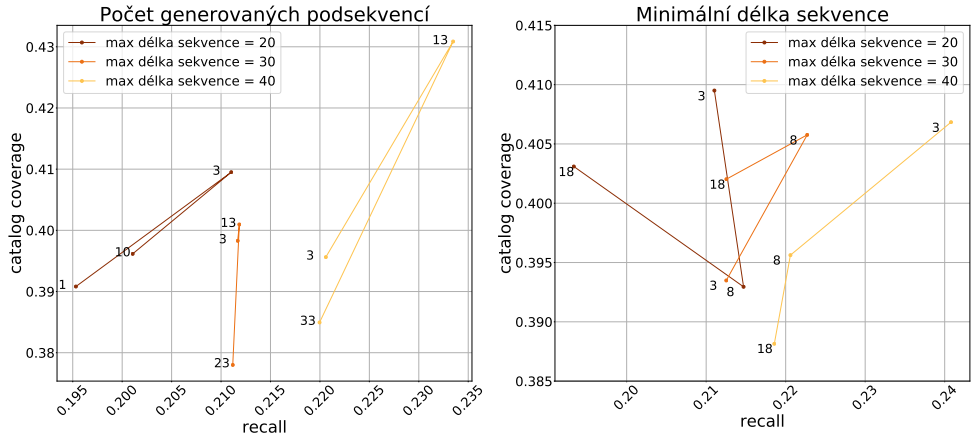
Z obou příkladů je vidět, že použití GRU není vhodné a LSTM dosahuje lepších výsledků. Pokud tedy není podmínkou šetřit nějaké jednotky procent místa na disku nebo času při trénování, tak není nutné GRU testovat.

4.6 Délka sekvence

Délka uvažované sekvence byla testována na datové sadě D2. Na obrázku 4.13 je vidět, že s rostoucí délkou sekvence roste i úspěšnost modelu téměř pro všechny hodnoty. Výchýlení, které je vidět pro hodnoty 50–100 je pravděpodobně způsobeno sezónním zbožím. V datasetu pro takovou délku sekvence mohlo mít obsažené sezónní zboží vliv na úspěšnost. Mohlo se jednat také například o Vánoce nebo akce na černý pátek. Úspěšnost opět vzrostla při dostatečné dlouhé sekvenci a potlačení této informace.

Na grafech na obrázku 4.14 jsou porovnávány hodnoty minimální délky sekvence a počtu generovaných trénovacích podsekvencí z jedné sekvence v datové sadě. Na levém obrázku je počet generovaných podsekvencí, kde se jedná o počet položek skrytých z posledního či posledních nákupů. Tyto položky je cíl predikovat. Je možné vidět, že je výhodné použít hodnotu, která je přiměřená i délce dané sekvence.

Na pravém grafu je omezena minimální délka sekvence. Je patrné, že omezování minimální délky sekvence má spíš negativní vliv na úspěšnost. Pro RNN



Obrázek 4.14: Na levém obrázku je zobrazen vývoj úspěšnosti pro počet generovaných sekvencí (počet skrytých posledních položek). Na pravém obrázku je omezována minimální délka sekvence

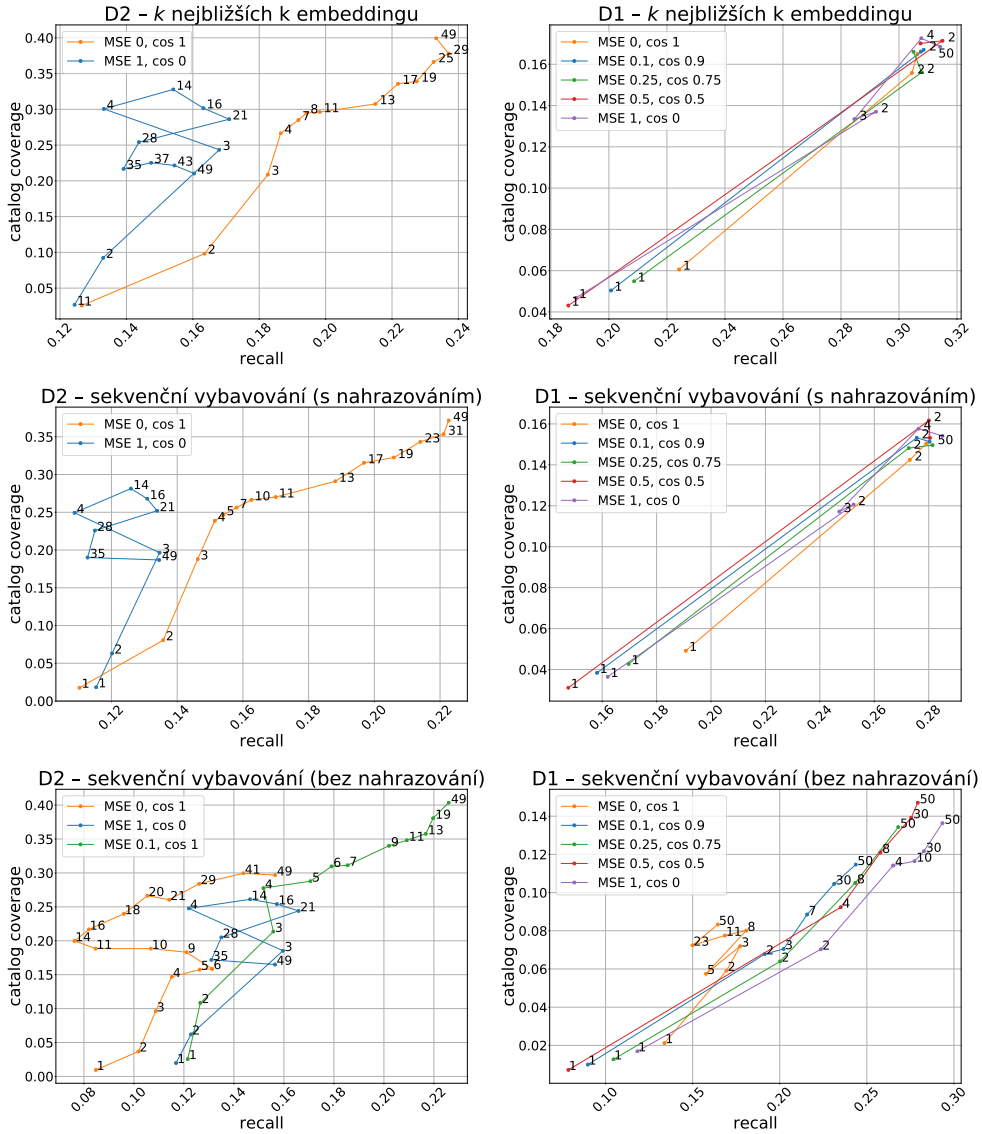
je tedy výhodné takové sekvence doplnit nulami, jak je i v práci použito pro kratší sekvence. Model si s doplněnými sekvencemi umí dobře poradit a využít informaci v nich obsaženou, jak je z grafu 4.14 patrné.

4.7 Způsob vybavování

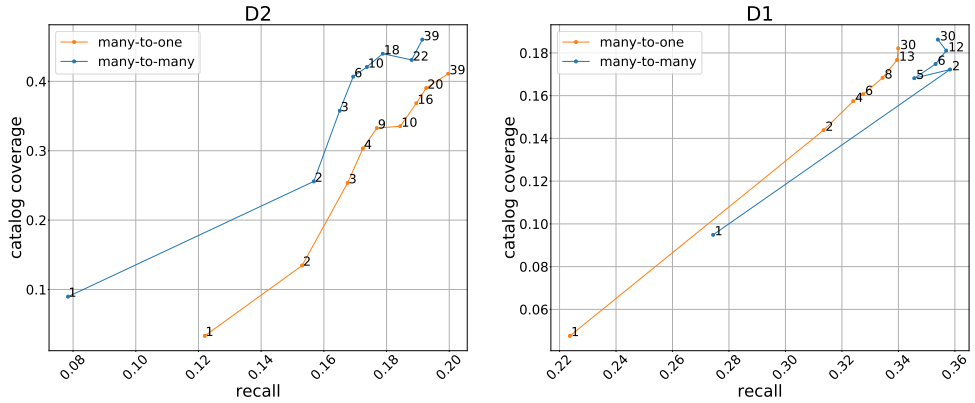
Navržené způsoby vybavování byly představeny v kapitole 2.5.2. Uvedeny byly tři způsoby vybavování, které se liší zpracováním výsledku sítě a také způsobem, jak je generováno doporučení. Na grafu 4.15 jednotlivé body představují epochy při učení, kde jsou hodnotami naznačeny první epochy pro lepší orientaci v grafu.

V první řadě grafů se jedná o způsob, kde je nalezeno n nejbližších položek k embeddingu, který je vygenerován sítí. V těchto experimentech tento způsob vyšel jako nejúspěšnější, i když rozdíly nejsou tak veliké. Úspěch tohoto modelu je zřejmě zajištěna schopností NN sítě hledat patřičný embedding (bod) v prostoru embeddingů, které jsou vygenerovány pomocí MF. Tento způsob by mohl být méně úspěšný při menším počtu latentních příznaků u reprezentace položek.

U druhého způsobu je vygenerovaný embedding znovu vložen do RNN jako pokračování sekvence a opět je generováno doporučení. Takto je vygenerováno celé doporučení. Rozdíl oproti poslednímu způsobu vybavování je především v embeddingu, který je vkládán opět do NN. V tomto způsobu je nahrazen již existujícím nejbližším embeddingem, na kterém byla NN učena. Výhodou přístupu je, že nemá problém s vysokými hodnotami embeddingu, se kterými



Obrázek 4.15: Srovnání způsobů vypavování RNN na datových sadách D1 a D2 pro modely s různými váhami ztrátové funkce



Obrázek 4.16: Porovnání modelů běhu RNN (*many-to-one* a *many-to-many*) na datasetech D1 a D2

měl model problémy v případě učení pouze podle ztrátové funkce obsahující kosinovou podobnost.

Posledním způsobem je nenahrazování výsledného embeddingu existujícím a jeho opětovného využití přímo do NN. Tento model má velice podobnou úspěšnost s výše zmíněným přístupem. Jediným rozdílem je použitá ztrátová funkce. Tento způsob vybavování je citlivý na velikost hodnot embeddingů. S těmito vysokými hodnotami byl problém v případě nevyužití složky MSE. Toto chování je patrné z grafů 4.15.

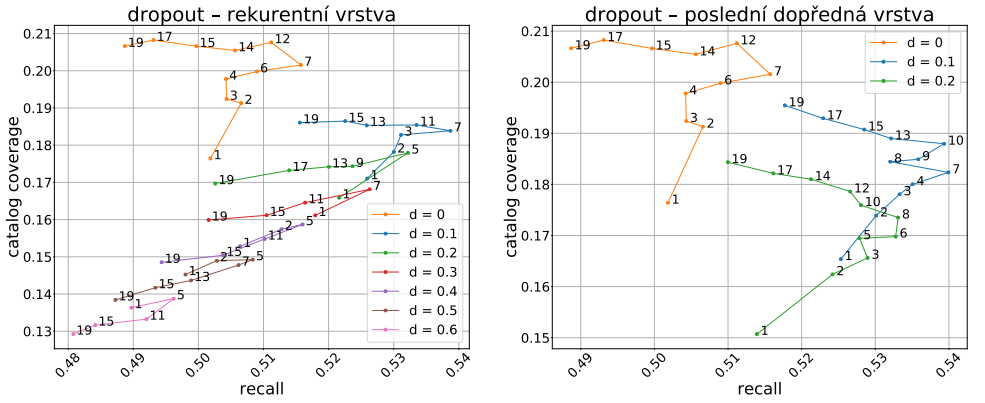
V použité metrice vychází první zmíněný způsob nejlépe, ale při pohledu na generovaná doporučení je patrné, že se doporučení z modelů liší a bylo by zajímavé způsoby porovnat na jiných metrikách, kterými lze porovnávat úspěšnost doporučování.

4.8 Modely běhu (many-to-one, many-to-many)

V této části jsou porovnány modely běhu popsané v sekci 2.5.1. Jedná se o dva způsoby trénování a vybavování neuronové sítě. Způsob *many-to-one* je výrazně rychlejší při trénování i při vybavování. Na druhou stranu model *many-to-many* by mohl pomoci modelu se lépe učit závislosti následujících položek v sekvenci.

Jak je patrné z grafu 4.16, nárůst výkonu není tak výrazný a není stejný na obou testovaných datových sadách D1 a D2. Pro D1 má metoda pozitivní vliv a roste úspěšnost. Tento nárůst může být způsoben délkou uživatelských sekvencí v datasetu, kde na datasetu D1 je následující položka mnohem více závislá na blízké historii, protože jsou sekvence krátké.

Snížení recallu na datové sadě D2 může být způsobeno dlouhými sekvencemi, kde učení modelu na predikci následující položky v každém kroku může být nevýhodné. Toto chování bylo možné vidět i v experimentech v sekci 4.5,



Obrázek 4.17: Graf vlivu hodnoty hyperparametru dropout na úspěšnost modelu pro datovou sadu yoochoose

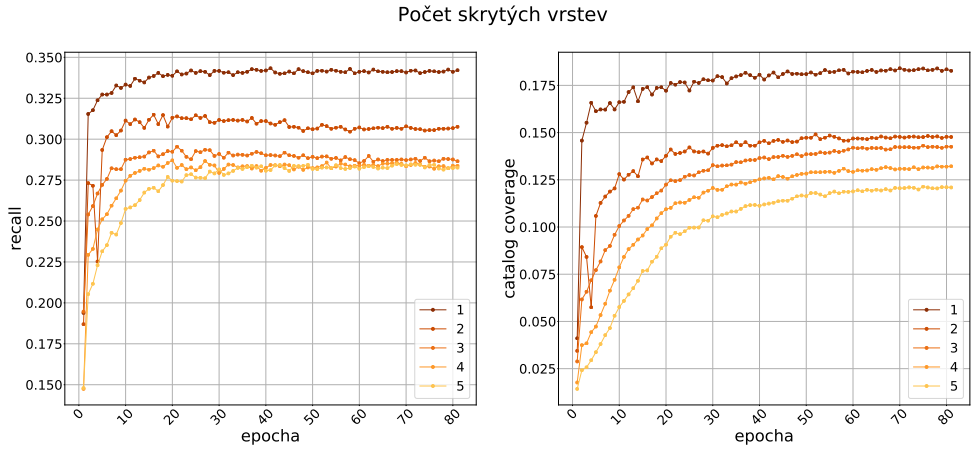
kde se GRU nebylo schopné dostatečně učit na dlouhých sekvencích. Na těchto sekvencích může být složitější identifikovat tu část historie, která je důležitá pro predikci. Také se mohou měnit preference uživatelů. Model běhu *many-to-one* nedává takový důraz na historii, protože je při učení propagována chyba velkým množstvím časových kroků a její vliv tím klesá. Dále také není poslední dense vrstva trénována v každém kroku jako v případě modelu *many-to-many*. Chyba počítána v každém kroku může dávat příliš velký důraz na historické části sekvence.

4.9 Dropout

Jak bylo zmíněno v sekci 1.3.3.3, dropout je velice často používanou metodou, která modelu umožnuje dosáhnout lepší generalizace a tím i lepších výsledků na testovacích datech. Síť je nucena se učit s nižší výpočetní kapacitou. Tento hyperparametr je představen na databázi yoochoose, ale využíván je na všech datových sadách.

Na grafech na obrázku 4.17 jsou vidět výsledky z měření na databázi yoochoose. Měřen je vliv hodnoty dropout pro skrytou vrstvu a také pro poslední dobřednou vrstvu. Pro skrytou vrstvu jsou výsledky na levém grafu a je vidět, že zavedení hodnoty vedlo ke skokovému nárůstu úspěšnosti. Se zvyšující se hodnotou dropoutu úspěšnost modelu klesala. Na toto chování může mít opět vliv struktura a velikost dat. Je vhodné najít správnou hodnotu pro konkrétní datové sady.

Pro tento dataset je hodnota menší, ale pro datasety D1 a D2 bylo nejlepších výsledků dosaženo s hondotami 0.3 – 0.4 pro skrytou vrstvu. Datové sady D1 i D2 mají k dispozici mnohem delší sekvence k učení.



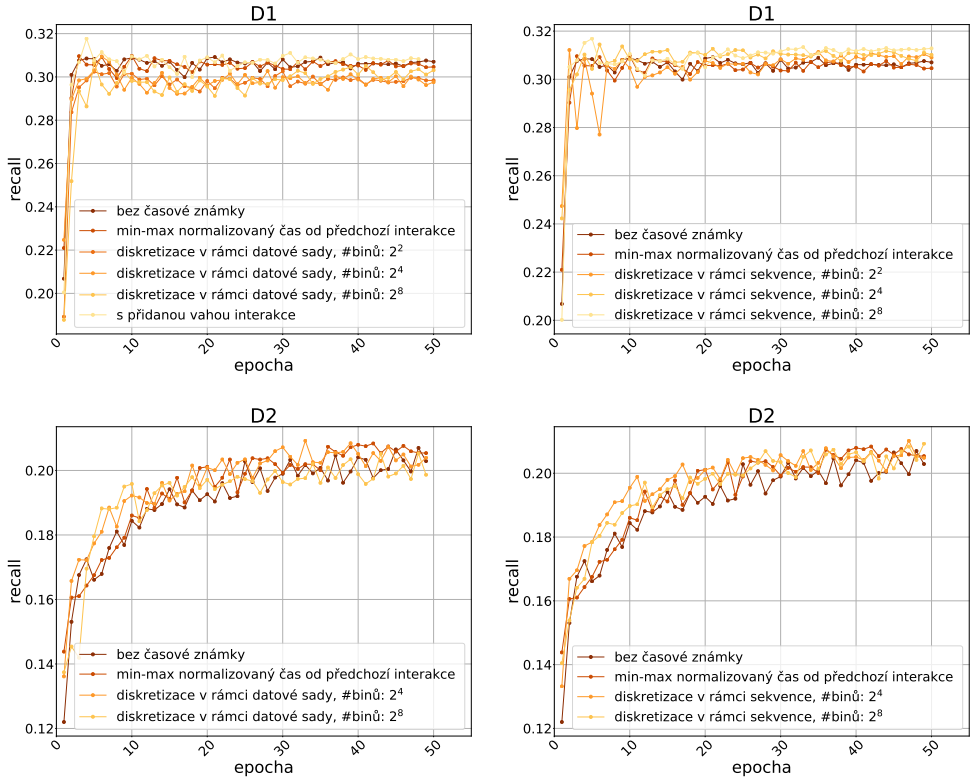
Obrázek 4.18: Graf vlivu počtu skrytých rekurentních vrstev na kvalitu modelu pro datovou sadu D1

Na grafu vpravo na obrázku 1.3.3.3 jsou zobrazeny hodnoty pro poslední dopřednou vrstvu, kde je dropout implementován pomocí skrývání vah. Je patrné, že hodnoty okolo 0.1 jsou ideální a stejné hodnoty jsou použity pro všechny datasety, protože v poslední vrstvě je způsob vybavování nezávislý na délce sekvence.

4.10 Počet skrytých rekurentních vrstev neuronové sítě

V této sekci je prozkoumána závislost úspěšnosti modelu na počtu skrytých rekurentních vrstev v modelu. V tomto parametru nebylo naplněno očekávání, kde s přidáváním vrstev by měla růst úspěšnost modelu. Z levého grafu na obrázku 4.18 je patrné, že s počtem skrytých vrstev modelu se hodnota recallu nezvyšuje. Nezvyšuje se ani hodnota metriky catalog coverage, která je zobrazena na grafu vpravo. Vzrostla pouze výpočetní náročnost modelu.

Toto může být zapříčiněno způsobem učení RNN a neschopností dostatečně propagovat chybu při učení skrz velké množství vrstev. Pro D1 a yoochoose je délka sekvence omezena na 10 a pro D2 se pohybovala délka sekvence až k 200. Při učení pomocí BPTT, popsaného v sekci 1.3.2.1, vede na učení NN s velkým počtem skrytých vrstev. Další možností může být přeučení modelu a bylo by nutné s modelem dále pracovat a pokusit se přeučení při použití více vrstev zabránit.



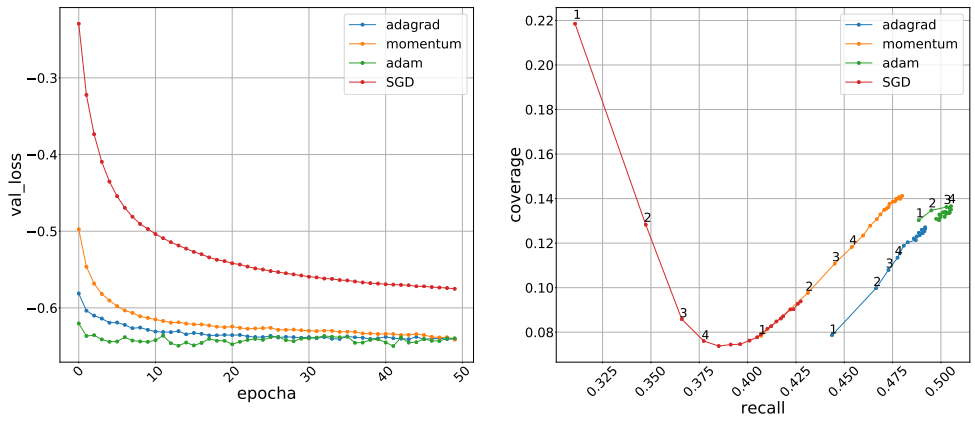
Obrázek 4.19: Úspěšnost modelu při použití kontextových dat (časové známky a váhy interakce) na datové sadě D1 a D2

4.11 Přidání kontextových dat do NN

V diplomové práci byla také implementována možnost přidání některých kontextových dat. Konkrétně pak přídání časové známky a váhy interakce. Váha byla otestována na datové sadě D1 a časová známka na datových sadách D1 a D2. Na datové sadě D2 nebyla váha testována z důvodu existence pouze nákupů položek.

Váha interakce je přidána jako jeden neuron ve vstupu. Časovou známku je možné přidat několika způsoby. Prvním je použití časového rozdílu od poslední interakce. Tato hodnota je ještě normalizována min-max normalizací, jak bylo uvedeno v části 2.4. Dalším způsobem je použití diskretizace s využitím binů. Diskretizaci je možné využít v rámci celé datové sady nebo také vždy v rámci jedné trénovací sekvence. Přidání kontextových dat je podrobněji popsáno v sekci 2.4.

Přidání váhy interakce je možné vidět na levém horním grafu na obrázku 4.19. Na grafu je patrné, že přidání váhy nepřináší téměř žádné zlepšení úspěšnosti nebo pouze minimání.



Obrázek 4.20: Vliv použitého optimalizačního algoritmu při učení NN sítě na úspěšnost modelu

Výsledky experimentů, kdy byla použita časová data, je možné vidět na grafu 4.19, kde na horních grafech jsou výsledky pro datovou sadu D1 a na spodních grafech pro D2. Z výsledků je patrné, že použití časové známky nepřináší velké výrazné zlepšení.

Na levém grafu pro D1 je vidět, že použití časové známky jako normalizovaného čísla nemá téměř žádný vliv. Při použití diskretizace v rámci celé datové sady má tato metoda dokonce negativní vliv na úspěšnost a recall klesá. Jiné je to na pravém grafu, kde je opět uveden základní graf pro srovnání. Modely s diskretizací ji používají v rámci jedné sekvence. Tato metoda na datové sadě D1 přináší zlepšení o několik jednotek procent.

Na spodních grafech pro datovou sadu D2 již není patrné žádné výraznější zlepšení ani pro jednu metodu. Tento stav může být zapříčiněn délkou sekvence, kde je použita délka 50 namísto 10 pro datovou sadu D1. Časový údaj nemusí mít tak výrazný vliv na predikci další interakce.

4.12 Vliv použité optimalizační metody

V sekci 1.3.3.5 jsou popsány možné optimalizační algoritmy, které lze využít při učení neuronové sítě a úpravě vah v NN. V této kapitole jsou tyto metody otestovány na datové sadě yoochoose. Výsledky experimentů je možné vidět na grafech na obrázku 4.20. Na levém je vývoj validační chyby a na grafu vpravo poté vliv na úspěšnost v rovině recall-coverage, kde jednotlivé body odpovídají epochám při učení. Pro lepší přehlednost jsou první epochy označeny.

Je patrné, že použitím vhodného algoritmu lze výrazně vylepšit model. Pro metodu SGD je vidět veliká nevýhoda nevyužití jiných hodnot než aktuální chyby, která pro jednotlivé trénovací vzorky může oscilovat a zpomalit učení. Tento problém je řešen metodou momentů a výsledek je i z grafů patrný.

Metoda adagrad nebyla popsaná v 1.3.3.5, ale zavádí různé učící faktory pro různé učené parametry . Zadaný učící faktor představuje pouze výchozí bod. Tato metoda se ukázala úspěšnou, protože modelu umožnuje rychleji učit váhy, které je nutné upravovat výrazněji. Naopak pro váhy, kde by vysoké změny vedly k vysokému rozptylu, je učící faktor snížen.

Poslední metodou je metoda adam, která vychází právě ze zmíněné myšlenky u adagrad. Využívá různé učící faktory pro různé učené parametry. Je zde patrný přínos využití prvního a druhého momentu. Přínos je vidět na grafu proti algoritmu adagrad, a proto bylo měření tohoto algoritmu uvedeno na grafu pro porovnání. Metoda adam velice rychle konverguje a již během několika epoch je dosaženo dobrých výsledků a učení může být ukončeno. Toto chování je možné vidět na grafech 4.20. Z tohoto důvodu byla metoda adam využívána i během experimentů.

4.13 Porovnání výsledků s jinými modely

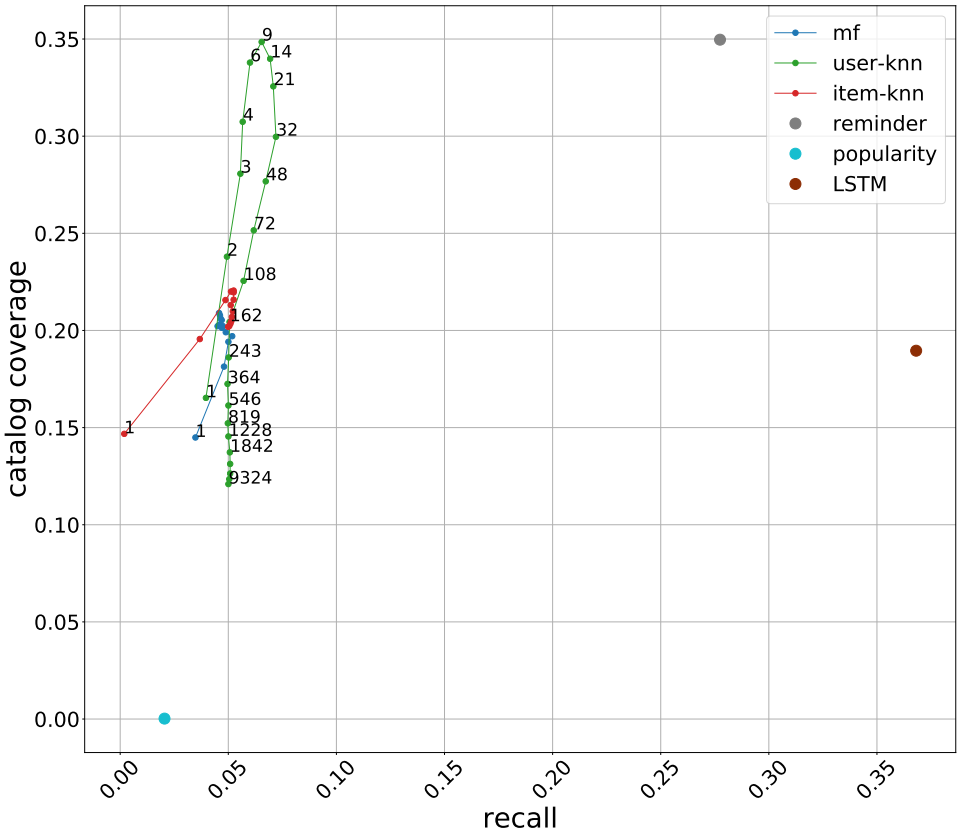
Na závěr experimetů je uvedeno porovnání modelů využívající RNN s ostatními modely zmíněnými v sekci 3.3. Porovnání bude provedeno na predikci dalšího nákupu, jak je využíváno v rámci celé práce, pokud není zmíněno jinak. Výsledky jsou rozděleny do podsekcí pro jednotlivé datové sady.

4.13.1 Datová sada D1

Na datové sadě D1 je možné vidět porovnání algoritmů kolaborativního filtrování a porovnání popularity, reminder a také LSTM modelů. Pro modely kolaborativního filtrování jsou na grafu 4.21 zobrazena některá k. V práci je na sekvenčích datech testována predikce dalšího nákupu. Z toho vyplývá, že se interakce již mohou opakovat. Pokud s položkou uživatel interagoval vícekrát, tak model standartního kolaborativního filtrování není schopen položku ze sekvence opět doporučit, protože pomocí něho lze doporučit pouze položky, které uživatel ještě neviděl.

Pokud se v datové sadě nachází např. koupě po zobrazení produktu v nějakém časovém okně (v práci zvoleno 10 minut), jsou interakce spojeny do nákupu. Tento přístup je využit i při trénování neuronové sítě, protože předpokládám, že zobrazení produktu následované jeho koupí, je jedna interakce uživatele s položkou. Přístup pomáhá modelům předcházet učení se opakování zobrazeného produktu, protože zobrazením se zvyšuje šance, že si produkt uživatel koupí. Každý uživatel si produkt nejdříve pravděpodobně prohlédne než si ho koupí, ale nemusí to vždy platit pro opakované nákupy.

Na grafu 4.21 je patrná výhoda modelů, které umí uživateli nabídnout již viděnou položku, kterou by pravděpodobně uživatel chtěl opět zkonzumovat. Modely LSTM a reminder dosahují výrazně vyšších hodnot recallu. LSTM model oproti reminder modelu doporučuje méně různých položek (má nižší



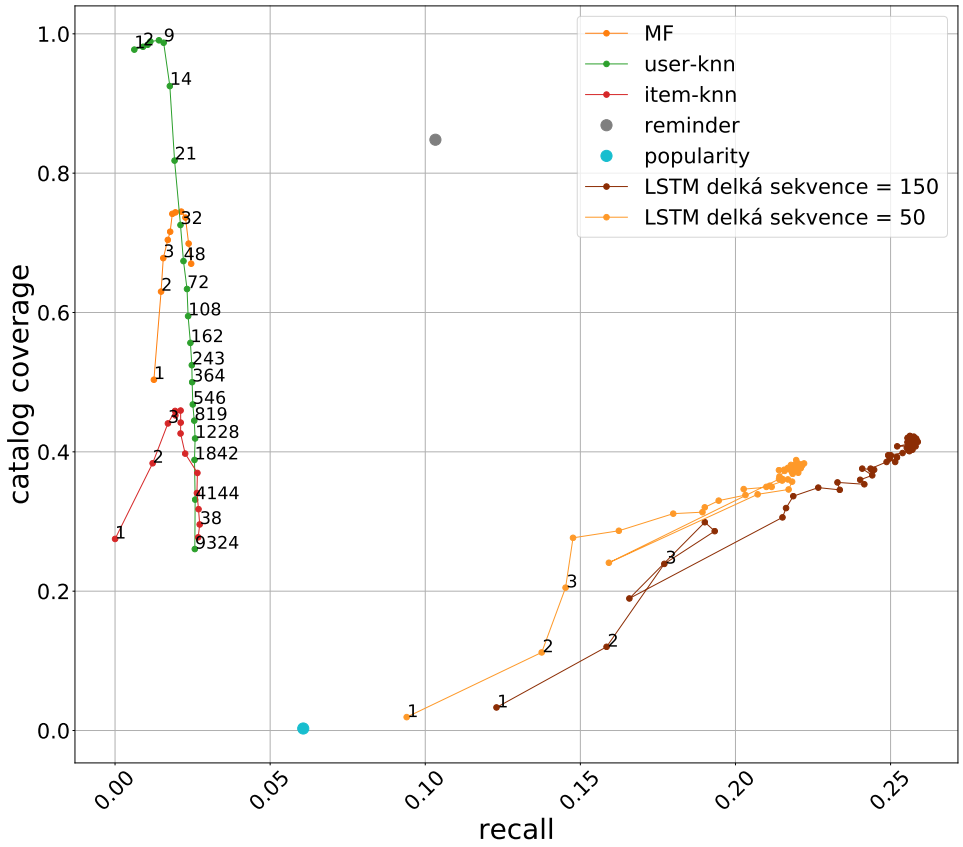
Obrázek 4.21: Porovnání vytvořeného modelu s dalšími algoritmy využívanými v doporučovacích systémech na datové sadě D1

catalog coverage), protože má pravděpodobně vyšší sklon se učit populární položky pro dané skupiny uživatelů.

Toto chování bude více patrné v kapitole 4.13.2, kde je zobrazena změna hodnot během celého učení. Na této datové sadě není učení modelu uváděno, protože model je schopen se během první epochy posunout velice výrazně a z grafu poté není nic patrné.

4.13.2 Datová sada D2

Na datové sadě D2 je úspěšnost modelu a jeho využití nejvíce patrné. Jak bylo vidět z tabulky 4.2, datová sada obsahuje dlouhé sekvence, na kterých je možné LSTM síť dobře naučit. Na grafu 4.22 je možné vidět srovnání s ostatními modely zmíněnými v předchozí kapitole. Pro modely kolaborativního filtrování (MF, item-knn, user-knn) představují hodnoty v grafu parametr k. Pro LSTM sítě jsou zde uvedeny hondoty pro jednotlivé epochy během učení. Z vývoje

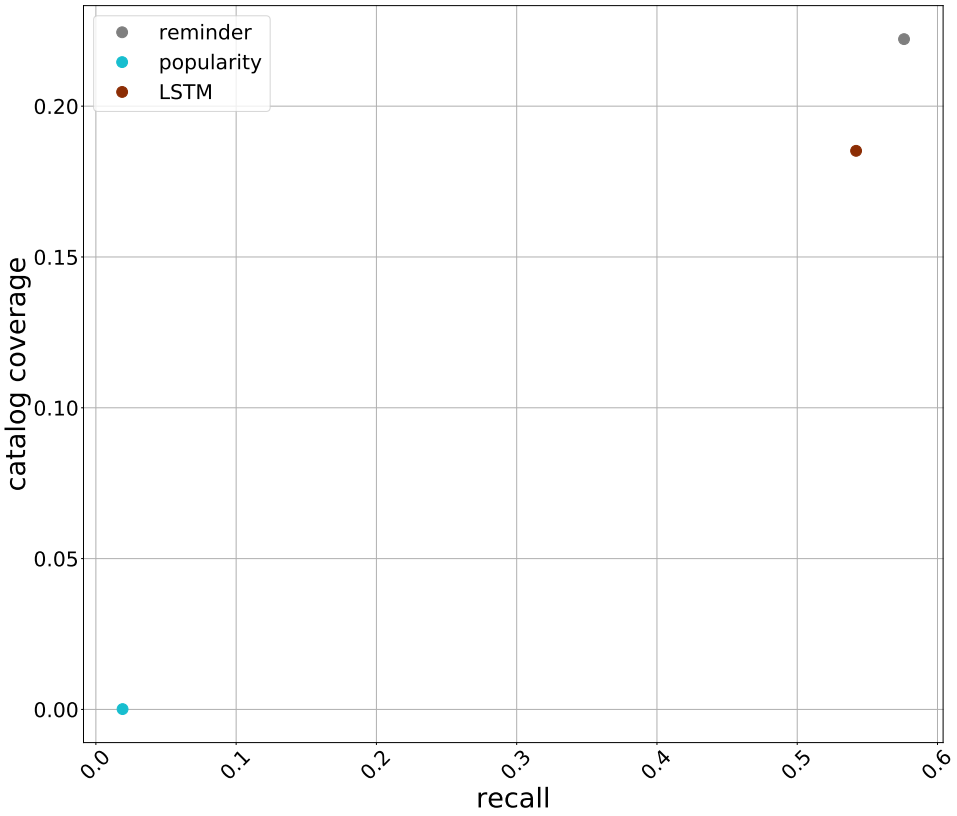


Obrázek 4.22: Porovnání vytvořeného modelu s dalšími algoritmy využívanými v doporučovacích systémech na datové sadě D2

je vidět, že v prvních epochách je model velice blízko modelu doporučujícímu populární položky.

Na tomto datasetu také model doporučující 5 nejpopulárnějších položek má poměrně vysoký recall. Model LSTM může být kombinací několika modelů. V doporučení jsou často patrné populární položky. Počet položek v doporučení, které již uživatel koupil, se pohybuje okolo 70 %. Model doporučuje i položky podobné těm, které již uživatel viděl, což je způsobeno vybavováním a využíváním embeddingů vytvořených pomocí MF. Při generování doporučení by mohly být některé položky při hledání odpovídajícího embeddingu penalizovány (potlačení opakování) a tím může být doporučení generované pomocí LSTM upraveno.

V hodnotách recall, na které byl model především optimalizován, se podařilo dosánout více než dvojnásobného zlepšení, které ukazuje na správný směr využití RNN při tvorbě doporučení na datových sadách s mnoha interakcemi.



Obrázek 4.23: Porovnání vytvořeného modelu s dalšími algoritmy využívanými v doporučovacích systémech na datové sadě yoochoose

4.13.3 Datová sada yoochoose

Na datové sadě yoochoose nejsou zobrazeny modely kolaborativního filtrování, protože při predikci dalšího nákupu jako interakce se jejich recall pohyboval jen málo nad hodnotou 0. Z úspěšnosti modelu doporučujícího populární položky je vidět, že se v datasetu nenachází položky, které by byly tzv. bestsellery. Naopak má velikou úspěšnost opakovací model, který se LSTM modelem nepodařilo překonat.

Trochu to naplnilo očekávání, které bylo zmíněno na začátku kapitoly při popisu datových sad. Tento dataset má velice krátké sekvence a mnoho cold-start položek. MF byla na tak řídkém datasetu velice obtížná, protože pro hodnoty prořezávání, kdy bývá MF úspěšná, byla odstraněna většina datové matice. Embeddingy proto nebyly dostatečně kvalitní pro trénování modelu LSTM. Pravděpodobně by bylo nutné zvolit pro tvorbu embeddingů nějaký jiný způsob nebo se pokusit využít segmentaci položek a použít kódování 1 z N.

Kapitola 5

Diskuze

V této kapitole jsou diskutovány zjištěné poznatky, možná zlepšení jednotlivých modelů, dále hyperparametrizace a celkové dosažené výsledky. Nejprve se zaměřím na embeddingy, poté na kvalitu samotného modelu. Na tyto sekce naváži diskuzí nad některými výsledky a z nich plynoucí vylepšení předzpracovávání dat. Na závěr budu diskutovat systém vyhodnocování.

5.1 Reprezentace položek – embeddingy

V této práci byla pro tvorbu embeddingů zvolena maticová faktorizace, pomocí které lze vytvářet velice kvalitní embeddingy. MF se však potýká s některými problémy. Prvním takovým problémem mohou být řidká data. Čím hustší je matice, tím kvalitnější embeddingy lze vytvořit. Z tohoto důvodu je využíváno prořezávání, aby nedocházelo k přeučování na řidká data. Nějvětší rozdíl je patrný mezi datovými sadami D2 a yoochoose. Druhý problém souvisí také s prořezáváním tím, že jsou odřezány položky bez interakcí (angl. cold-start items), pro které poté neexistuje embedding. Pro trénování NN nelze interakce obsahující tyto položky využít.

Pro yoochoose, kde jsou data velmi řidká, jak je možné vidět v tabulce 4.1, se během experimentů nepodařilo za pomoci MF vůbec predikovat následující nákup. Z tabulky 4.1 je také patrný úbytek položek obsažených v testovací a validační datové sadě, který naznačuje mnoho položek s málo interakcemi a problém se studeným startem, se kterým se také kolaboritivní filtrování potýká. Zbylé datasety takovým problémem netrpí. Řešením pro tento problém by mohla být tvorba embeddingů s využitím atributových nebo obrázkových dat.

Oproti youchoose pro datovou sadu D2 bylo možné využít vysokou hodnotu prořezávání a regularizace, čímž je možné vytvořit ještě hustší matici. MF na této sadě poskytla velmi kvalitní a stabilní embeddingy, které byly vhodným vstupem pro vytvořený rekurentní model.

Na D1 se také podařilo vytvořit kvalitní embeddingy, stále však byly některé položky s málo interakcemi odstraněny. Řešení se nabízí stejné jako bylo naznačeno u yoochoose.

5.2 Vylepšení modelů a jejich hyperparametrizací

Možným zlepšením modelu, které by mohlo být velice účinné, vychází z modelů běhů představených v sekci 2.5.1 a otestovaných v části 4.8. Model many-to-many zde nedosáhl uspokojivých výsledků. Nevýhodou modelu je snaha predikovat následující položku na každou v sekvenci. Poslední dopředná vrstva je učena i na historických datech, což může snižovat úspěšnost.

Řešením by mohlo být využití upraveného způsobu many-to-many. Nebylo by tedy cílem predikovat po každé položce následující, ale predikce by zahrnovala pouze posledních několik nákupů. Ty by mohly být omezeny například maximálním stářím. V případě této úpravy by bylo nutné změnit ztrátovou funkci a celý proces učení sítě a vytvořit způsob, který by s tímto uměl nějak dynamicky pracovat.

Z hyperparametrů, které by mohly vést k lepším výsledkům a jejich testování by bylo velmi výpočetně i časově náročné, se jedná o kombinace učícího faktoru a dalších možných hyperparametrů optimalizátoru v kombinaci s hodnotou dropout. Během experimentů se ukázalo, že tyto dva parametry spolu při učení úzce souvisí. Průběh učícího faktoru lze velice flexibilně měnit a tak měnit míru učení sítě.

Rekurentní neuronové sítě, konkrétně pak sestavené z buněk LSTM, umožňují trénování s vyššími hodnotami dropoutu, který se může pohybovat okolo hodnoty 0,4, ale s úpravou průběhu a hodnot učícího faktoru by se tato hodnota mohla ještě zvýšit a modely by umožnily lepší generalizaci. Takto vysoké hodnoty platí pro datovou sadu D2. Pro řidší datové sady byly hodnoty menší, ale pořád by bylo možné s využitím učícího faktoru tuto hodnotu zvýšit.

5.3 Předzpracování dat – sezónní data

Během experimentů na datové sadě D2 jsem při testování délky sekvence v sekci 4.6 na obrázku 4.13 pozoroval zajímavý propad úspěšnosti modelů pro nějaké délky sekvence. Tento propad byl pozorován pouze na datové sadě D2, protože zde byly k dispozici dlouhé sekvence, kde byla omezována jejich délka a některá historická data byla odříznuta. Na zbylých datových sadách se toto chování neprojevovalo, což mohla způsobit například krátká historie uživatelů.

Tento propad přisuzuji sezónní povaze dat, kdy mohla být využita k trénování data, která pro aktuální období (např. zima nebo léto) neměla význam a mohla doporučení vychylovat nesprávným směrem a tím způsobit výsledný propad. Při dalším prodloužení sekvence již byla tato sezonní data znovu kompenzována.

Tento problém by bylo možné řešit úpravou předzpracování dat. Trénovací sekvence by nebyly tvořeny pouze z posledních obsažených položek, ale dlouhé sekvence by byly využity tak, aby trénovací sekvence pokrývaly rovnoměrně celý rok. Dalším krokem by mohlo být kromě omezení délky sekvence i omezení časové, kde by se jednalo například o celé roky.

5.4 Vyhodnocování zohledňující sekvenční povahu dat

Sekvenční vyhodnocování navržené v sekci 2.6 je upraveno tak, že cílem je predikce následujícího nákupu. Toto přímo reflektuje požadavek doporučit uživateli produkt, který si koupí. Jak je možné vidět z výsledků v části 4.13, opakovací model reminder dosahuje vysoké úspěšnosti. Uživatelé si pravděpodobně koupí položku, kterou již v minulosti viděli nebo koupili.

Navržený model využívající rekurentní neuronové sítě se toto chování může také učit. Proto jsem v předzpracování v sekci 2.3 navrhnul způsob úpravy sekvencí, kde kromě nákupů tvořících cíl predikce, jsou spojovány některé interakce nad stejnými položkami v blízkém čase nebo za sebou. Tímto je ze sekvence odstraněna velká část opakujících se položek, které měl model tendenci se učit doporučit.

I tak zůstává několik desítek procent zopakovaných položek, které ovšem nebyl cíl úplně odstranit a můžou představovat výhodu modelu k dosažení vyšších hodnot recallu. Model by například mohl mít parametr na řízení počtu opakovaných položek. Tato omezení by však pravděpodoně měla velký vliv na metriku recall a to především v negativním smyslu. Tyto úpravy by musely pak být spíše požadavkem a omezením na vytvářené doporučení.

Validace těchto úprav by byla možná použitím nějaké online metriky jako je například míra prokliku (angl. click-through rate, CTR) nebo konverzní poměr (angl. conversion rate, CR). Pomocí těchto metrik by také mohlo být zvalidováno, zdali opakování modelu a jeho pozitivní vliv na recall má také stejně pozitivní vliv na online metriku, která může lépe reflektovat chování uživatelů.

Závěr

V této práci jsem se zabýval doporučovacími systémy založenými na rekurentních neuronových sítích, které zohledňují sekvenční povahu dat vstupujících do tohoto systému. Cílem práce byl návrh a implementace algoritmu pro tvorbu modelů, které generují doporučování na základě těchto sekvenčních dat.

Algoritmus implementovaný v rámci práce umožnuje trénováním vytvářet modely generující doporučení využívající rekurentní neuronové sítě. Algoritmus lze také využít k vybavování vytvořeného modelu pro tvorbu doporučení, na kterém je model testován. Kromě algoritmu pro práci s rekurentními modely byl vytvořen způsob testování modelů zohledňující sekvenční povahu dat. Zmíněné části spolu s modulem, který umožnuje tvorbu embeddingů pomocí maticové faktorizace, a dalšími modely tvoří framework pro ladění modelů využívající rekurentní neuronové sítě, jehož běh je řízen jedním strukturovaným konfiguračním souborem.

Během experimentů bylo otestováno několik možností, které spadají především do architektury modelu. Byly vybrány ty úspěšnější modely a přístupy. Dále bylo otestováno velké množství hyperparametrizací a ukázán jejich vliv na chování vygenerovaných modelů a na jejich úspěšnost. Modely byly testovány pomocí způsobu, který byl v této práci navržen a diskutován. Způsob vychází ze standartních existujících metod.

Celkově se pomocí modelů založených na rekurentních neuronových sítích podařilo dosáhnout uspokojivých výsledků a to především na primárně sledované největší datové sadě D2, kde se povedlo pomocí vytvořených modelů zdvojnásobit hodnoty recallu proti opakovacímu reminder modelu. V porovnání s kolaborativním filtrováním byly rozdíly ještě výraznější z důvodu toho, že tyto algoritmy ve standartní verzi neumožňují doporučení položky, kterou již uživatel viděl. Hodnota recallu pro vytvořené modely byla přibližně 7–9 krát vyšší než pro standartní kolaborativní filtrování.

Využití embeddingů generovaných pomocí maticové faktorizace limituje vytvářené embeddingy, které slouží jako vstup do sítě. Především položky

mající málo interakcí mohou mít nekvalitní embedding nebo mohou být při technice prořezávání odebrány. Zde se nabízí první možnost navázání na tuto práci. Zaměřit se na embeddingy a vytvářet je nejen pomocí interakcí, ale využít například atributová data nebo obrázky pro jejich zkvalitnění a vytvořit embeddingy i pro položky s málo interakcemi, aby nebyly v sekvenci odstraněny, ale využity.

Předchozí způsob uplatnění atributých dat není možné využít pouze při tvorbě embeddingů, ale tato data by mohla být rozšířením vstupu rekurentního neuronového modelu, tak jak bylo navrženo přidání kontextových dat v této práci. Kromě atributových dat by mohl být vstup rozšířen vhodným obrázkovým embeddingem s příslušným doplněním ztrátové funkce.

Posledním možným rozšířením by mohla být úprava navrženého modelu pro predikci následujícího košíku, kde by model musel predikovat i jeho velikost, která by mohla být určována částí neuronové sítě nebo dalším modelem. Druhou možností je přidání míry důvěry k doporučení jako její součást, kterou by bylo ovšem nutné nějak vhodně trénovat. Velikost košíku by byla určována nějakou prahovou hodnotou této důvěry.

Literatura

[1] Martínek, L.: *Evaluace algoritmů lokálně senzitivního hashování (LSH) v doporučovacích systémech.* Bakalářská práce, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2018. Dostupné z: https://dspace.cvut.cz/handle/10467/76825

[2] Ricci, F.; Rokach, L.; Shapira, B.; aj.: *Recommender systems handbook.* Springer New York Dordrecht Heidelberg London, 2011, ISBN 978-0-387-85819-7.

[3] Jawaheer, G.; Szomszor, M.; Kostkova, P.: Comparison of implicit and explicit feedback from an online music recommendation service. [online], 2010, doi:https://doi.org/10.1145/1869446.1869453, [cit. 2020-02-21].

[4] Jannach, D.; Zanker, M.; Felfernig, A.; aj.: *Recommender systems: an introduction.* 32 Avenue of the Americas, New York, NY 10012-2473, USA: Cambridge University Press, 2010, ISBN 978-0-521-49336-9.

[5] Řehořek, T.: *Manipulating the Capacity of Recommendation Models in Recall-Coverage Optimization.* Dizertační práce, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2019. Dostupné z: https://dspace.cvut.cz/handle/10467/81823

[6] Jawaheer, G.; Weller, P.; Kostkova, P.: Modeling user preferences in recommender systems: A classification framework for explicit and implicit user feedback. [online], 2014, doi:https://doi.org/10.1145/2512208, [cit. 2020-02-21].

[7] Núñez Valdez, E.; Cueva Lovelle, J.; Sanjuán, O.; aj.: Implicit feedback techniques on recommender systems applied to electronic books. [online], 07 2012, doi:10.1016/j.chb.2012.02.001, [cit. 2020-02-21].

[8] Bennett, J.; Lanning, S.; aj.: The netflix prize. In *Proceedings of KDD cup and workshop*, ročník 2007, Citeseer, 2007, str. 35, [cit. 2020-02-23].

[9] Koren, Y.; Bell, R.; Volinsky, C.: Matrix factorization techniques for recommender systems. *Computer*, ročník 42, č. 8, 2009: s. 30–37, doi: 10.1109/MC.2009.263, [cit. 2020-02-27].

[10] Hu, Y.; Koren, Y.; Volinsky, C.: Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, Ieee, 2008, s. 263–272, doi:10.1109/ICDM.2008.22, [cit. 2020-02-27].

[11] Parra, D.; Karatzoglou, A.; Amatriain, X.; aj.: Implicit feedback recommendation via implicit-to-explicit ordinal logistic regression mapping. *Proceedings of the CARS-2011*, ročník 5, 2011, [cit. 2020-02-23].

[12] Han, J.; Pei, J.; Kamber, M.: *Data mining: concepts and techniques*. Elsevier, 2011, ISBN 978-0-12-381479-1.

[13] Johnson, C. C.: Logistic matrix factorization for implicit feedback data. *Advances in Neural Information Processing Systems*, ročník 27, 2014, [cit. 2020-02-27].

[14] Adomavicius, G.; Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, ročník 17, č. 6, 2005: s. 734–749, doi:10.1109/TKDE.2005.99, [cit. 2020-02-27].

[15] Yu, H.-F.; Hsieh, C.-J.; Si, S.; aj.: Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *2012 IEEE 12th International Conference on Data Mining*, IEEE, 2012, s. 765–774, doi:10.1109/ICDM.2012.168, [cit. 2020-02-27].

[16] Agrawal, R.; Srikant, R.: Mining sequential patterns. In *Proceedings of the eleventh international conference on data engineering*, IEEE, 1995, s. 3–14, doi:10.1109/ICDE.1995.380415, [cit. 2020-02-27].

[17] Devooght, R.; Bersini, H.: Long and short-term recommendations with recurrent neural networks. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, 2017, s. 13–21, doi:https://doi.org/10.1145/3079628.3079670, [cit. 2020-02-27].

[18] Takács, G.; Pilászy, I.; Németh, B.; aj.: Scalable collaborative filtering approaches for large recommender systems. *Journal of machine learning research*, ročník 10, č. Mar, 2009: s. 623–656, [cit. 2020-02-27].

[19] Goldberg, D.; Nichols, D.; Oki, B. M.; aj.: Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, ročník 35, č. 12, 1992: s. 61–70, doi:https://doi.org/10.1145/138859.138867, [cit. 2020-02-28].

[20] Sarwar, B.; Karypis, G.; Konstan, J.; aj.: Application of dimensionality reduction in recommender system-a case study. [online], 2000, [cit. 2020-02-28]. Dostupné z: http://files.grouplens.org/papers/webKDD00.pdf

[21] Firooz, H.: Simple Matrix Factorization with TensorFlow. [online], 2016, [cit. 2020-02-29]. Dostupné z: http://hameddaily.blogspot.com/2016/12/simple-matrix-factorization-with.html

[22] Takács, G.; Pilászy, I.; Tikk, D.: Applications of the conjugate gradient method for implicit feedback collaborative filtering. In *Proceedings of the fifth ACM conference on Recommender systems*, 2011, s. 297–300, doi: https://doi.org/10.1145/2043932.2043987, [cit. 2020-02-29].

[23] Eckart, C.; Young, G.: The approximation of one matrix by another of lower rank. *Psychometrika*, ročník 1, č. 3, 1936: s. 211–218, doi:https://doi.org/10.1007/BF02288367, [cit. 2020-03-01].

[24] Frederickson, B.: Implicit. [online], 2019, [cit. 2020-03-09]. Dostupné z: https://github.com/benfred/implicit/

[25] Haykin, S. S.; aj.: *Neural networks and learning machines*. New York: Prentice Hall„ 2009, ISBN 978-0-13-147139-9.

[26] Rojas, R.: Perceptron learning. In *Neural Networks*, Springer, 1996, s. 77–98, doi:https://doi.org/10.1007/978-3-642-61068-4_4, [cit. 2020-03-15].

[27] Da Silva, I. N.; Spatti, D. H.; Flauzino, R. A.; aj.: *Artificial neural networks*. Springer, 2017, ISBN 978-3-319-43162-8, 39 s.

[28] Khuong, B.: The Basics of Recurrent Neural Networks (RNNs). [online], 2019, [cit. 2020-03-17]. Dostupné z: https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

[29] Pascanu, R.; Mikolov, T.; Bengio, Y.: On the difficulty of training recurrent neural networks. In *International conference on machine learning*, 2013, s. 1310–1318, [cit. 2020-03-17].

[30] Hochreiter, S.; Schmidhuber, J.: Long short-term memory. *Neural computation*, ročník 9, č. 8, 1997: s. 1735–1780, doi:https://doi.org/10.1162/neco.1997.9.8.1735, [cit. 2020-03-17].

[31] Gers, F. A.; Schmidhuber, J.; Cummins, F.: Learning to forget: Continual prediction with LSTM. 1999, doi:10.1049/cp:19991218, [cit. 2020-03-18].

[32] Cho, K.; Van Merriënboer, B.; Gulcehre, C.; aj.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. [online], 2014, [cit. 2020-03-17]. Dostupné z: https://arxiv.org/pdf/1406.1078.pdf

[33] Werbos, P. J.: Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, ročník 78, č. 10, 1990: s. 1550–1560, doi: 10.1109/5.58337, [cit. 2020-03-18].

[34] Hanselmann, T.; Zaknich, A.; Attikiouzel, Y.: Connection between BPTT and RTRL. [online], 01 1999, [cit. 2020-03-18]. Dostupné z: http://www.wseas.us/e-library/conferences/athens1999/Papers/296.pdf

[35] Goodfellow, I.; Bengio, Y.; Courville, A.: Deep Learning. [online], 2016, [cit. 2020-03-17]. Dostupné z: http://www.deeplearningbook.org

[36] Ioffe, S.; Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. [online], 2015, [cit. 2020-03-19]. Dostupné z: https://arxiv.org/pdf/1502.03167.pdf

[37] Santurkar, S.; Tsipras, D.; Ilyas, A.; aj.: How Does Batch Normalization Help Optimization? [online], 2018, [cit. 2020-03-19]. Dostupné z: http://papers.nips.cc/paper/7515-how-does-batch-normalization-help-optimization.pdf

[38] Kingma, D. P.; Ba, J.: Adam: A method for stochastic optimization. [online], 2014, [cit. 2020-03-19]. Dostupné z: https://arxiv.org/pdf/1412.6980.pdf

[39] Quadrana, M.; Cremonesi, P.; Jannach, D.: Sequence-Aware Recommender Systems. [online], 2018, doi:https://doi.org/10.1145/3190616, [cit. 2020-03-17], 1802.08452. Dostupné z: http://arxiv.org/abs/1802.08452

[40] Hidasi, B.; Karatzoglou, A.; Baltrunas, L.; aj.: Session-based recommendations with recurrent neural networks. [online], 2015, [cit. 2020-03-21]. Dostupné z: https://arxiv.org/pdf/1511.06939.pdf

[41] Tan, Y. K.; Xu, X.; Liu, Y.: Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, 2016, s. 17–22, [cit. 2020-03-21]. Dostupné z: https://arxiv.org/pdf/1606.08117.pdf

[42] Hidasi, B.; Quadrana, M.; Karatzoglou, A.; aj.: Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, 2016, s. 241–248, doi:https://doi.org/10.1145/2959100.2959167, [cit. 2020-03-21].

[43] Soh, H.; Sanner, S.; White, M.; aj.: Deep sequential recommendation for personalized adaptive user interfaces. In *Proceedings of the 22nd international conference on intelligent user interfaces*, 2017, s. 589–593, doi:https://doi.org/10.1145/3025171.3025207, [cit. 2020-03-21].

[44] http://www.numpy.org/: Numpy. 2020. Dostupné z: https://github.com/numpy/numpy/

[45] Virtanen, P.; Gommers, R.; Oliphant, T. E.; aj.: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, ročník 17, 2020: s. 261–272, doi:https://doi.org/10.1038/s41592-019-0686-2.

[46] Abadi, M.; Agarwal, A.; Barham, P.; aj.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, software available from tensorflow.org. Dostupné z: https://www.tensorflow.org/

[47] Chollet, F.; aj.: Keras. https://keras.io, 2015.

[48] Ben-Shimon, D.; Tsikinovsky, A.; Friedmann, M.; aj.: Recsys challenge 2015 and the yoochoose dataset. In *Proceedings of the 9th ACM Conference on Recommender Systems*, 2015, s. 357–358, doi:https://doi.org/10.1145/2792838.2798723, [cit. 2020-03-01].

Příloha **A**

Seznam použitých zkratek

RSs doporučovací systémy (angl. recommender systems)

RP úloha predikce hodnocení (angl. rating prediction)

CF kolaborativní filtrování (angl. collaborative filtering)

k-NN algoritmus hledání k nejbližších sousedů (angl. k-nearest neightbors)

MF maticová faktorizace (angl. matrix factorization)

SVD singulární rozklad matice (angl. singular value decomposition)

SGD stochastický gradientní sestup (angl. stochastic gradient descent)

ALS metoda alternujících nejmenších čtverců (angl. alternating least squares)

NN neuronové sítě (angl. neural networs)

ANN umělé neuronové sítě (angl. artificial neural networs)

RNN rekurentní neuronové sítě (angl. recurent neural networks)

BP algoritmus zpětného šíření chyby (angl. backpropagation)

BPTT algoritmus zpětného šíření chyby v čase (angl. backpropagation through time)

RTRL algoritmus trénování rekurentní neuronové sítě v reálném časee (angl. real time recurrent learning, RTRL)

LSTM rekurentní buňka neuronové sítě řízená bránami s kombinací dlouhodobé a krátkodobé paměti (angl. long short-term memory)

GRU rekurentní buňka neuronové sítě řízená bránami (angl. Gated recurrent units)

TP položky určené správně jako relevantní (angl. true positive)

FP položky určené jako relevantní i když neměly být (angl. false positive)

FN položky určené špatně jako nerelevantní (angl. false negative)

TN položky určené správně jako relevantní (angl. true negative)

MSE střední kvadratická chyba (angl. mean squared error)

RMSE odmocnina ze střední kvadratický chyby (angl. root mean square error)

yaml rekurzivní formát pro datovou serializaci, které je čitelný jak strojově, tak lidmi, je použitelný pro konfigurační soubory, které jsou lehce čitelné.

csv datový formát pro uložení taulkových dat, který má hodnoty oddělené čárkami

tf-idf schéma pro hodnocení relelevance, které je nejčastěji používané při zpracování textu, ale je možné jej využít i na interakční matici s implicitními interakcemi

bm25 schéma hodnotící relevanci podobně jako tf-idf, ale je novější a ne tak citlivé na vysoké hodnoty a lze jej využít pro vážení interakční matice.

hdf5 standard v knihovně keras pro uložení neuronové sítě obsahující důležité údaje pro načtení modelu a připadné vybavování nebo pokračování trénování

CPU centrální procesorová jednotka

GPU specializovaný grafický akcelerátor, který v současné době kromě práci s obrazovými daty slouží k provádění výpočtu v rámci strojového učení

CTR míra prokliku (angl. click-through rate) – procento doporučení, na které uživatel nějakým způsobem reagoval

CR konverzní poměr (angl. conversion rate – procento lidí kteří si zakoupili položku po jejím doporučení

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `release.sh` ..... skript pro sestavení nové verze textu
- `requirements.txt` ..... požadované knihovny Pythonu
- `data`
  - `D1` ..... soubory s výsledky pro D1
  - `D2` ..... soubory s výsledky pro D2
  - `yoochoose-example` ..... vzorový formát všech používaných souborů
  - `youchoose-experiments` ..... soubory s výsledky pro yoochoose
- `src`
  - `src` ..... zdrojové kódy implementace
  - `config` ..... configurační soubory
  - `cover` ..... zdrojové kódy pro desky práce
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `plots` ..... python notebooky se zdrojovými kódy pro grafy
  - `mf.ipynb` ..... spuštění modulu MF
  - `evaluation.ipynb` ..... spuštění vyhodnocování
  - `splitAndSaveData.ipynb` ..... spuštění modulu pro rozdělení dat
  - `trainLSTM.ipynb` ..... spuštění trénování NN
- `text`
  - `DP_Martinek_Ladislav_2020.pdf` ..... text práce ve formátu PDF

Příloha C

Konfigurační soubor algoritmu

V této příloze je uveden konfigurační soubor, který je kvůli přehlednosti rozdělen po stránkách, ale lze podle struktury spojit do jednoho souboru.

```
experiment:
  general:
    experiment-name: dp
    dataset: yoochoose
    interaction-data-file: yoochoose-interactions.csv
    max-memory-mb: 20000
  evaluation:
    # Models to evaluate (item-knn, pre-item-knn)
    algorithms: user-knn, reminder, lstm, popularity
    n-test-users: 50
    top-n: 5
    # last-leave-one-out or leave-one-out
    evaluation-type: last-leave-one-out
    max-in-leave-one-out: 5 # max out items for one test user
    # raw - no pre-processing
    # merge - joins the same interactions side by side
    # merge;10 - joins the same interactions side by side
    #          - if the time difference < 10 minutes
    # end_purchase - sequence end with purchase
    # end_purchase_merge - a combination of the above
    # end_purchase_merge;10 - a combination of the above
    # end_purchase_sub - generates shortening sequences
    # end_purchase_sub_merge - a combination of the above
    # end_cart_add - sequence end with card addition
    #              - and combination above
    # cross_merge - merge across whole sequence
    data-prec: raw # pre-processing for test users
```

```
28  data-preparation:
29    days: -1 # vertical scaling, -1 for disable
30
31    number-of-train-users: 1000000 # horizontal scaling
32    number-of-valid-users: 100000 # -1 for disable
33
34    min-weight: -1
35    max-weight: 1
36
37    # same as data-prec in evaluation
38    data-prec: raw # train and validation users
39
40    min-ratings-per-user: 3
41    max-ratings-per-user: 5000
42
43    train-p: 0.9
44    valid-p: 0.05
45    test-p: 0.05
46    num-users-take: 1. # percentage of users to take
47
48    save-to-csv: True
49
50  factorization:
51    parametrizations:
52      - factors: 128 # embedding size
53        regularization: 1
54        iterations: 15
55        num-threads: 5
56        # remove items and users with less interaction
57        pruning: 10
58        use-bm25: True
59        bm25B: 0.8
60        bm25M: 5.
61        use-gpu: False
62
63      - factors: 64
64        regularization: 1
65        iterations: 15
66        num-threads: 5
67        pruning: 10
68        use-bm25: True
69        bm25B: 0.8
70        bm25M: 5.
71        use-gpu: False
```

```
lstm:
  # use for generate sequences
  # for train and evaluate the RNN network
  min-sequence-length: 3
  max-sequence-length: 50
  max-train-seq-one-user: 3

  # switches to use context data
  add-weight: False
  add-timestamps: False
  use-bins: False # diskretization timestamps
  per-seq-bins: False
  # 2^4 - number of bins for discretization
  bins-exponent: 2

  # loss function weights
  cosine-weight: 1.
  mse-weight: 0.01
  timestamp-weight: 0.01
  weight-weight: 0.01

  cache-size: 0 # cache size in MB
  workers: 8 # the work processes used for training RNN
  LSTM: True # use LSTM cell otherwise GRU

  model-type: many-to-one # or many-to-many
  remove-items-without-embedding: True

  model-postfix: all_data_1 # model naming
  test-batch-evaluate: True # fast batch evaluating on GPU
  # models for evaluation split by ;
  models-to-test: e32b32nn1n32m40_all_data_1

  # work as python range split value by ;
  model-load-epoch-range: 1;7
  n-rec-cycle: 1 # number of evaluation cycles
  nearest-item-cycle: True
  k: 5 # number of items for recommendation
  beta: 0 # popularity weighting

  # possibility train existing model with other data
  # path to model to load
  # None to create a new empty model
  model-to-train: None
```

```
116     # file with embeddings
117     embeddings: embeddings_als_dp_f_32_l_1_p_5_B_0_8_M_5_0.emb
118     # embeddings size
119     input-embeddings: 32
120
121     # NN architecture
122     layer-size: 32
123     n-recurent-layers: 1
124     epochs: 100
125     batch-size: 256
126
127     # dropout layer for input
128     # each step use another dropout mask
129     dropout-rate: 0.0
130     dropout-all: False
131
132     # dropout layer for input
133     # use same dropout mask for all step
134     first-dropout: 0.4
135     first-recurrent-dropout: 0.3
136
137     # dropout in next recurrent layers
138     dropout: 0.2
139     recurrent-dropout: 0.1
140
141     # last dense layer dropout
142     dense-dropout: 0.01
143     dense-dropout-all: True
144
145     # possibilities: adam, adagrad, momentum, SGD
146     optimizer: adam
147     activation: tanh # tanh, relu
148     recurrent-activation: sigmoid
149
150     # for adam : 0.001
151     # for SGD, momentum, adamgrad : 0.01
152     learning-rate: 0.0015
153     # step(factor, drop_every),
154     # poly(power(0-constant)), None - constant
155     ls-scheduler: step
156     factor: 0.6
157     drop-every: 10
158     power: 2
159
```

```
160     # weights constrains
161     # maxnorm (max value need), nonnegnorm (no value need)
162     # minmaxnorm (both mina and max need) or none
163     dense-l-constraint: none
164     dl-min: -1
165     dl-max: 3
166     dense-l-bias-constraint: none
167     dlb-min: -1
168     dlb-max: 3
169     lstm-l-constraint: none
170     lstm-min: -1
171     lstm-max: 3
172     lstm-l-bias-constraint: none
173     lstmb-min: -1
174     lstmb-max: 3
175     lstm-l-recurrent-constraint: none
176     lstmr-min: -1
177     lstmr-max: 3
178
179   user-knn:
180     # work as python range
181     # if is last value negative -> for multiplicative constant
182     k: 1 10000 -1.5
183     beta: 0.2 # popularity weighting
184
185   item-knn:
186     k: 1 50 -1.3
187     beta: 0 # popularity weighting
188
189   precomputed-similar-items:
190     similar-items-files-regex: sim.*
191     k: 1 50 -1.3
192     beta: 0 # popularity weighting
193
194   similarity-items-rating:
195     similarity-rating-file: similar_items_ratings_test
196     max-n: 10 # number of similars to save
```