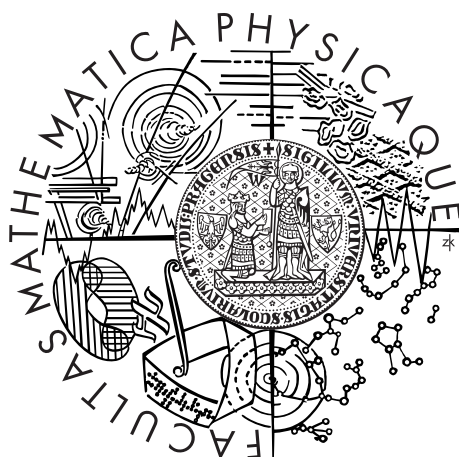


Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



David Marek

Implementace aproximativních Bayesovských metod pro odhad stavu v dialogových systémech

Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: Ing. Mgr. Filip Jurčíček, Ph.D.

Studijní program: Informatika

Studijní obor: Teoretická Informatika

2013

Chtěl bych poděkovat svému vedoucímu Ing. Mgr. Filipovi Jurčíčkovi, Ph.D. za cenné rady a čas, který mi věnoval. Dále bych chtěl poděkovat svým kolegům Lukáši Žilkovi, Matěji Korvasovi, Ondřeji Plátkovi a Ondřeji Duškovi, s kterými jsem mohl spolupracovat na projektu Vystadial. Nakonec bych chtěl poděkovat svým nejbližším za podporu, kterou mi poskytli po celou dobu studia.

Tato práce byla částečně financována z grantu Ministerstva školství, mládeže a tělovýchovy pod identifikačním kódem LK11221, Vystadial – Vývoj metod pro návrh statistických mluvených dialogových systémů.

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V dne

Podpis autora

Název práce: Implementace aproximativních Bayesovských metod pro odhad stavu v dialogových systémech

Autor: David Marek

Ústav: Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: Ing. Mgr. Filip Jurčíček, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: Každý dialogový systém obsahuje komponentu řízení dialogu, která vykonává akce dialogového systému na základě dialogového stavu. Dialogový stav reprezentuje všechny dostupné informace. Systém pro odhad stavu slouží k udržování stavu dialogu a integraci nových informací ze systému porozumění. Pro efektivní práci komponenty řízení dialogu je důležité co nejpřesněji modelovat pravděpodobnostní rozdělení definované nad všemi stavy dialogu. Těchto stavů může být velké množství, proto se většinou používají aproximativní techniky. V této práci se budeme zabývat implementací aproximativních metod pro odhad stavu dialogu pomocí inference v Bayesovských sítích. Výsledkem je knihovna použitelná v systému pro odhad stavu reálného dialogového systému.

Klíčová slova: dialogový systém, odhad dialogového stavu, aproximativní Bayesovské metody

Title: Approximative Bayes methods for belief monitoring in spoken dialogue systems

Author: David Marek

Department: Institute of Formal and Applied Linguistics

Supervisor: Ing. Mgr. Filip Jurčíček, Ph.D., Institute of Formal and Applied Linguistics

Abstract: The most important component of virtually any dialog system is a dialogue manager. The aim of the dialog manager is to propose an action (a continuation of the dialogue) given the last dialog state. The dialog state summarises all the past user input and the system input and ideally it includes all information necessary for natural progress in the dialog. For the dialog manager to work efficiently, it is important to model the probability distribution over all dialog states as precisely as possible. It is possible that the set of dialog states will be very large, so approximative methods usually must be used. In this thesis we will discuss an implementation of approximate Bayes methods for belief state monitoring. The result is a library for dialog state monitoring in real dialog systems.

Keywords: dialog system, belief state monitoring, approximative Bayes methods

Obsah

Úvod	3
1 Teorie dialogových systémů	6
1.1 Dialogový systém	6
1.2 Součásti dialogového systému	6
1.2.1 Systém rozpoznávání řeči (ASR)	7
1.2.2 Porozumění mluvené řeči (SLU)	8
1.2.3 Komponenta řízení dialogu	10
1.2.4 Generování přirozené řeči (NLG a TTS)	10
1.3 Dialogový stav	11
1.3.1 Aktualizace stavu	12
2 Bayesovské sítě a inference	13
2.1 Bayesovské sítě	13
2.2 Inference v Bayesovských sítích	14
2.2.1 Exaktní inference	15
2.2.2 Posílání zpráv ve faktor grafu	19
2.2.3 Belief Propagation	21
2.2.4 Loopy Belief Propagation	22
2.3 Propagace s aproximovanými zprávami	25
2.3.1 Rodina exponenciálních rozdělání	26
2.3.2 Expectation Propagation	27
3 Učení parametrů	31
3.1 Grafický model	31
3.2 Výpočet marginálních pravděpodobností	31
3.2.1 Marginální pravděpodobnost proměnných	32
3.2.2 Marginální pravděpodobnost parametrů	33
3.3 Aproximace marginálních pravděpodobností	34
3.4 Algoritmus	38
4 Implementace	40
4.1 Diskrétní faktor	40
4.1.1 Reprezentace faktorů	40
4.1.2 Operace s faktory	41
4.1.3 Algoritmus pro operace s různými doménami	42
4.1.4 Marginalizace proměnných	44
4.2 Vrcholy faktor grafu	44
4.2.1 Rozhraní vrcholů	45
4.2.2 Rozhraní vrcholů pro proměnné	46
4.3 Vrcholy pro Dirichletovské parametry	46
4.4 Inferenční algoritmus	46
4.4.1 Strategie výběru vrcholu v LBP algoritmu	47
4.5 Příklady	48
4.5.1 Použití jednotlivých komponent	48

4.5.2	Učení dirichletovských parametrů	55
4.6	Dialog State Tracking Challenge	56
4.6.1	Let's Go!	56
4.6.2	Popis systému	61
4.6.3	Evaluace	62
4.7	Existující řešení	62
	Závěr	65
	A Zpracování dialogů z DSTC	66
A.1	Soubory	66
A.2	Konfigurace	67
A.3	Výpočet	68
	B Dokumentace tříd	72
B.1	Balíček <code>alex.infer</code>	72
B.1.1	Modul <code>factor</code>	72
B.1.2	Modul <code>node</code>	78
B.1.3	Modul <code>lbp</code>	85
	Seznam použité literatury	87
	Přílohy	90

Úvod

Dialog je přirozený způsob dorozumívání a sdělování informací mezi lidmi. Počítač, který by dokázal vést dialog s uživatelem, byl vždy snem nejen příznivců vědecko-fantastické literatury. Už pro první počítače vznikaly programy snažící se využít přirozenou řeč pro interakci s uživatelem. Jedním z takových programů byla například Eliza [30], program jednající jako psychiatr. Fungoval na principu rozpoznání textu pomocí gramatiky a následné transformace textu do promluv dle pravidel. Avšak gramatiky a pravidlové systémy se ukázaly nedostačné pro praktické aplikace a tak se vývoj přesunul do statistických metod.

Z počátku byla výkonnost a úspěšnost systémů pracujících s přirozeným jazykem nízká, což vedlo ke skeptickým názorům na jejich budoucnost. Asi téměř všichni si pamatují první pokusy o diktování textu v kancelářských aplikacích. Systémy pro rozpoznání přirozeného jazyka se však stále zlepšovaly a dostaly se do fáze, kdy už je možné využít hlas jako vstup pro komplexní systémy.

Tato práce se zabývá doménově omezenými statistickými dialogovými systémy. Dialogové systémy mohou vyhledávat spojení v městské dopravě, hledat turistické informace, ovládat navigaci v automobilu atd. Nabízejí spoustu výhod, jsou dostupné pro nevidomé, mohou pracovat 24 hodin denně a mohou nahradit lidské operátory, což může potenciálně vést k úsporám.

Dialogové systémy mají stále daleko k dokonalosti. Neumí se dobře vyrovnat se špatně rozpoznanými vstupy, šumem a nejasnostmi. Na rozdíl od živých operátorů nedokáží improvizovat. V některých případech není uživatel spokojený s dialogovým systémem jen proto, že nepochopil otázku a dialogový systém není schopen se zeptat jinak.

První problém se snaží řešit statistické dialogové systémy. Pokud si nejsme jisti cílem uživatele, nabízí se řešení v modelování uživatele pomocí pravděpodobnostního modelu. Díky němu se pak můžeme vyrovnat s chybami zapříčiněnými špatně rozpoznanými vstupy. Tato práce se bude zabývat metodami pro efektivní výpočty ve statistických modelech aplikovatelných v dialogových systémech.

Rozdělení práce

Nejprve v kapitole 1 popíšeme dialogové systémy a jejich jednotlivé součásti. Blíže se budeme zabývat dialogovým stavem a jeho reprezentací pomocí Bayesovského přístupu.

Jako vhodnou strukturu pro reprezentaci dialogového stavu si představíme Bayesovské sítě v kapitole 2. Ukážeme několik metod pro inferenci v Bayesovských

sítích. Začneme jednoduchou, ale pomalou exaktní inferencí a přesuneme se až k efektivní aproximativní inferenci pomocí algoritmu Loopy Belief Propagation. Nakonec ukážeme metodu Expectation Propagation pro inferenci v grafickém modelu s aproximovanými faktory, která je zobecněním předchozích metod.

Pro využití v dialogových systémech je možné buď nastavit pevné parametry (ať už ručně nastavené nebo naučené z dat), anebo zvolit Bayesovský přístup i pro parametry. Parametry pocházející z Dirichletovského rozdělení jsou prezentovány v kapitole 3.

V kapitole 4 jsou popsány všechny implementované části a také příklady jejich použití. V této kapitole je také popsána soutěž Dialog State Tracking Challenge, do které byl přihlášen generativní systém pro odhad dialogového stavu používající popsanou knihovnu. Tato soutěž testovala systémy pro odhad dialogového stavu na datech z reálného dialogového systému.

Cíle

1. Dialogové systémy často využívají pouze nejlepší hypotézu ze systému porozumění přirozené řeči. Většina ovšem umí vytvořit seznam n nejlepších hypotéz. Tato práce si klade za cíl představit metody pro inferenci dialogového stavu v dialogovém systému s využitím více hypotéz. Představené metody budou založeny na reprezentaci dialogového stavu pomocí dynamických bayesovských sítí.
2. Bude představen algoritmus Loopy Belief Propagation a implementován pro využití v reálných systémech pro odhad dialogového stavu.
3. Algoritmus bude otestován na datech z reálného dialogového systému Let's Go! a porovnán s dalšími účastníky soutěže Dialog State Tracking Challenge.
4. Důležitou částí algoritmů pro inferenci je určení pořadí, v jakém má inference probíhat. Práce bude obsahovat implementaci několika strategií pro inferenci. Díky nim bude možné zvolit strategii pro efektivní inferenci v různých druzích Bayesovských sítí (stromy, dynamické sítě, obecné grafy).
5. Nakonec se práce bude zabývat učením parametrů sítě a představí algoritmus Expectation Propagation, algoritmus pro nalezení aproximace pravděpodobnostního rozdělení.
6. Při většině reálných použití dochází k aproximacím a úpravám modelu v závislosti na problému tak, aby bylo možné inferenci provádět v reálném

čase. Práce bude obsahovat implementaci frameworku, do kterého je možné jednoduše zasadit vlastní moduly pro aproximaci pravděpodobnostních rozdělání. Ty pak budou používány algoritmem Expectation Propagation. Jako příklad bude ukázán systém pro učení parametrů pravděpodobnostní distribuce v generickém modelu pro reprezentaci dialogového stavu.

1. Teorie dialogových systémů

1.1 Dialogový systém

Dialogový systém je počítačový systém umožňující uživatelům komunikovat s počítačem ve formě, která je přirozená a efektivní pro použití. Vývoj dialogových systémů má před sebou ještě spoustu problémů k překonání a pro praktické použití je třeba se uchýlit k mnoha předpokladům a zjednodušením. Prvním zjednodušením je doménová specializace. V současné době není možné vytvořit dialogový systém, který by se dokázal s uživatelem bavit o libovolném tématu. Z toho plyne i nutnost ontologie, systém potřebuje databázi popisující všechny poskytnutelné informace a jejich strukturu. Systém má definované použitelné fráze a věty, ale na rozdíl od živého operátora nedokáže improvizovat.

Další zjednodušení se týkají přímo dialogu. Většinou se předpokládá, že dialog probíhá mezi systémem a jedním uživatelem. Navíc se pravidelně střídají v obrátkách. Jedna obrátka dialogu je složená z jedné promluvy systému a jedné promluvy uživatele.

Příkladem dialogového systému může být systém pro nalezení spojení pomocí městské dopravy. Příkaz od uživatele může vypadat např. takto: „Chci jet z Malostranského náměstí na Anděl“. Dialogový systém z této věty odvodí, že uživatel hledá spojení. Výchozí stanice je Malostranské náměstí a cílová stanice je Anděl. Nyní záleží na chování dialogového systému, může například předpokládat, že uživatel chce vždy najít nejdřívější spojení, pokud neřekne jinak. V takovém případě už systém najde v databázi nejbližší spojení a uživateli jej sdělí.

Důležitou vlastností dialogového systému je robustnost. Pokud budeme používat dialogový systém v přirozeném prostředí, musíme se vyrovnat s tím, že často nebude uživateli rozumět. Systém může informaci přeslechnout, anebo si nemusí být jistý tím, co slyšel. Dialogový systém se proto musí umět uživatele doptat na chybějící informace a musí umět pracovat s nejistotou.

1.2 Součásti dialogového systému

Dialogový systém se skládá z několika spolupracujících částí. Na vstupu je zvukový záznam uživatele. O jeho převedení do textu se stará systém rozpoznávání řeči (ASR). Z textu je potřeba získat sémantické informace pomocí systému porozumění mluvené řeči (SLU). Nad sémanticky anotovanými informacemi už může pracovat komponenta řízení dialogu, která zvolí patřičnou odpověď. Výstup komponenty řízení dialogu je pouze ve formě dialogových aktů, které zachycují

podstatu odpovědi, ale ne její formu. O jejich převedení do textu se stará systém generování přirozené řeči (NLG). Do zvukového záznamu převede text syntetizér řeči (TTS).

1.2.1 Systém rozpoznávání řeči (ASR)

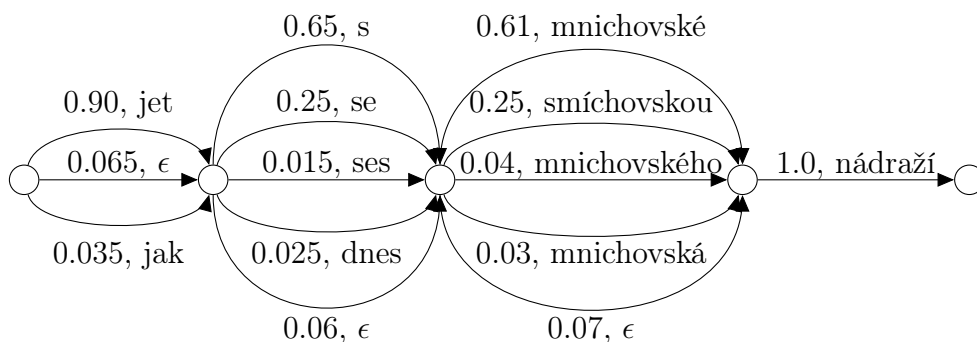
Systém rozpoznávání řeči slouží k převedení mluveného projevu do textové podoby. Až po získání textové podoby je možné se zabývat významem textu. Aktuálně nejlepší systémy jsou založené na pravděpodobnostním modelu a využívají Skryté Markovské modely (HMM) k určení nejpravděpodobnější sekvence slov pro daný zvukový záznam [11]. Pro tuto část dialogového systému existuje řada dostupných otevřených toolkitů, např. systém HTK [35], Kaldi [21] nebo SPHINX [29]. Existuje i celá řada komerčního softwaru od firem jako IBM nebo Nuance.

Úspěšnost systému rozpoznávání řeči je závislá na obtížnosti úlohy a na počtu trénovacích dat pocházejících ze stejné domény. Pro obecnou doménu se problém stává mnohem těžším a je třeba velké množství dat. K měření výkonu ASR systému se používá metrika Word Error Rate (WER). Při počítání WER je třeba nejprve provést zarovnání rozpoznávaného a originálního textu. WER je pak počet slov, která jsou změněná, smazaná, anebo přidaná, vydělený počtem slov v originálním textu. Dialogový systém Let's Go! [22] například dosahuje průměrné WER 64.3%.

Systém rozpoznávání řeči může produkovat více hypotéz pro jeden vstup. Často existuje pro jeden zvukový záznam více možných slovních sekvencí, ze kterých by mohl pocházet. Reprezentací možných hypotéz může být seznam slovních sekvencí s jejich věrohodností. Věrohodnosti jsou skóre přiřazená hypotézám určující jakou důvěru má systém rozpoznávání řeči ve správnost dané slovní sekvence. Příklad možného seznamu hypotéz je v tabulce 1.1.

Věrohodnost	Hypotéza
0.676	poliklinika modřany
0.072	poliklinika
0.063	cheb poliklinika modřany
0.054	poliklinika bory
0.045	poliklinika modřany i
0.036	polikliniky
0.027	cheb poliklinika
0.018	cheb poliklinika bory
0.009	poliklinika i

Tabulka 1.1: Příklad seznamu hypotéz ze systému pro rozpoznávání přirozené řeči. Uživatel by rád jel do zastávky Poliklinika Modřany.



Obrázek 1.1: Příklad konfúzní sítě.

Pro práci s hypotézami je vhodné, aby věrohodnost odpovídala aposteriorní pravděpodobnosti sekvence slov, dáno vstupním zvukem.

Další možností, jak reprezentovat výstup, je použití konfúzní sítě [4]. Konfúzní síť je vážený orientovaný graf, obsahující startovní a konečný vrchol a hrany označené slovy. Každá cesta ze startovního do konečného vrcholu vede přes všechny ostatní vrcholy. Váhy hran jsou pravděpodobnosti slova přiřazeného dané hraně. Hrany mohou obsahovat i prázdné slovo ϵ . Pravděpodobnost sekvence slov je součinem vah po cestě ze startovního do konečného uzlu. Výhodou konfúzní sítě je, že umožňuje v komprimované podobě uložit mnohem více hypotéz. Příklad konfúzní sítě je na obrázku 1.1.

1.2.2 Porozumění mluvené řeči (SLU)

Po získání hypotéz o promluvě uživatele se musí dialogový systém pokusit porozumět, co se uživatel snažil sdělit. Dialogový systém nepotřebuje vědět, co přesně uživatel řekl, důležité je pouze zjistit význam sdělení. Pokud například uživatel řekne „Chtěl bych nalézt spojení z Malostranského náměstí na Anděl“ anebo „Jak se dostanu na Anděl ze zastávky Malostranské náměstí?“, tak pro dialogový systém jsou obě tvrzení ekvivalentní. Uživatel požaduje informace o spojení mezi dvěma zastávkami, i když v jednom případě jde o větu oznamovací a v druhém případě o otázku.

První rozdělení mezi přesným významem sdělení a jeho účelem se objevilo ve formě tzv. speech aktů [2]. Následně byla idea speech aktů rozšířena pro potřeby dialogových systémů [28] a výsledek byl nazván dialogovým aktem.

V této práci používáme definici dialogového aktu ze soutěže Dialog State Tracking Challenge [32]. Sémantická reprezentace sdělení uživatele se tedy nazývá dialogový akt (DA), skládá se z jedné nebo více položek dialogového aktu (DAI) spojených v konjunkci. Každá DAI se skládá z typu, názvu slotu a jeho hodnoty. Typy jsou doménově nezávislé, sloty a jejich hodnoty reprezentují koncepty

ontologie. Příklad dialogového aktu ze systému pro hledání spojení v městské dopravě:

`hello(), inform(route="61a").`

Zde se dialogový akt skládá ze dvou položek, první položka má pouze typ *hello*, značící pozdrav. Druhá položka má typ *inform*, tzn. uživatel nás informuje o svém požadavku. Název slotu je *route* a hodnota je „61a“, tedy uživatel nám říká, že hledá spojení linkou 61a.

Typů může být libovolné množství, ale existuje několik základních, jejichž použití je ustálené.

- *inform* — sdělujeme informaci, doplňujeme hodnotu do slotu,
- *request* — požadujeme od protějšku doplnění hodnoty pro dotazovaný slot,
- *confirm* — chceme potvrdit hodnotu slotu, potvrzení může být implicitní, anebo explicitní. Při explicitním potvrzení očekáváme odpověď „Ano“ nebo „Ne“. U implicitního, pokud se nám nedostane odpovědi, předpokládáme, že protějšek souhlasí.

V tabulce 1.2 jsou ukázány všechny typy dialogových aktů, které může uživatel říct v dialogovém systému Let's Go! [32].

Existuje široké množství technik použitelných pro porozumění mluvené řeči. Unifikace pomocí šablon nebo gramatiky jsou příklady ručně psaných metod. Metody založené na datech jsou například Hidden Vector State model [10], techniky

Typ dialogového aktu
hello
bye
goback
restart
null
repeat
nextbus
prevbus
tellchoices
affirm
negate
deny
inform

Tabulka 1.2: Typy dialogových aktů, které může říct použít uživatel v systému Let's Go!

strojového překladu [34], Combinatory Categorical Grammars [38] nebo Support Vector Machines [14].

1.2.3 Komponenta řízení dialogu

Pokud už jsou pravděpodobné dialogové akty dekodovány, je třeba rozhodnout, jak bude systém reagovat. Komponenta řízení dialogu rozhoduje o dalším kroku systému. Odpověď systému je zakódována do formy dialogových aktů a nazývá se systémová akce.

Zvolená systémová akce a je vybrána z množiny možných akcí \mathcal{A} a závisí na vstupu, který systém obdržel z SLU. Tento vstup se nazývá pozorování $o \in \mathcal{O}$, obsahuje všechny informace získané od uživatele.

Zvolení správné akce potřebuje více znalostí než jen poslední pozorování. Historie dialogu a kontext hrají důležitou roli. Komponenta řízení dialogu proto udržuje interní reprezentaci celé historie dialogu. Všechny informace o historii dialogu a cílech uživatele nazýváme stavem dialogu.

Cíle uživatele ovšem neznáme a tedy musíme nějak uvažovat nad všemi možnými hodnotami. Pro vyrovnání se s tímto problémem se dialog považuje za Částečně pozorovatelný Markovský rozhodovací proces (POMDP) a cíle uživatele za skryté proměnné [33]. Pravděpodobnostní rozdělení nad možnými dialogovými stavy nazýváme stav důvěry $b \in \mathcal{B}$. Aktuální stav důvěry závisí na přechodové funkci, která jej aktualizuje pro každé nové pozorování a systémovou akci. Přechodová funkce je tedy mapování $\mathcal{T} : \mathcal{B} \times \mathcal{A} \times \mathcal{O} \rightarrow \mathcal{B}$. V této práci se budeme věnovat právě metodám pro odhad stavu.

Chování systému definuje dialogová strategie π . Strategie určuje další akci systému v závislosti na aktuálním dialogovém stavu. Stochastické strategie přiřadí možným akcím pravděpodobnost s jakou by je měl systém použít a pak z nich jednu náhodně vybere.

Pozorování, stav důvěry a akce jsou číslovány podle obrátky. Pokud je časový okamžik důležitý, pozorování, stav důvěry a akce z obrátky číslo t se značí o_t , b_t a a_t .

1.2.4 Generování přirozené řeči (NLG a TTS)

Posledním krokem dialogového systému je vytvoření odpovědi pro uživatele. Nejprve systém generování přirozené řeči (NLG) převede dialogové akty na text. Následně je text převeden na zvuk pomocí textového syntetizéru řeči (TTS).

Nejjednodušším přístupem ke generování přirozeného jazyka z dialogových aktů je použití šablon. Například pro dialogový akt `inform(from.stop="x")` bude

vytvořena šablona „Pojedete ze zastávky x“, kde „x“ bude nahrazeno názvem zastávky („Malostranská“, „Letňany“ atd.). Šablony jsou jednoduché a často efektivní řešení, protože počet možných frází je většinou malý.

Při syntéze řeči existuje mnoho alternativ. Je možné použít segmenty řeči z databáze pro vygenerování zvuků tvořících dohromady celou sekvenci slov. Příkladem těchto systémů je Festival [6] nebo jeho odnož FLite [7].

Alternativní metodou syntézy je použití Skrytých Markovských modelů pro generování zvuku, příkladem je HTS systém [37].

Pro syntézu řeči existují také komerční systémy, například systém SpeechTech TTS [24].

1.3 Dialogový stav

Dialogový stav je reprezentace všech získaných informací z aktuálního dialogu. To znamená vše co uživatel a systém řekli. Systém ovšem nezná cíle uživatele a navíc systémy pro rozpoznávání i porozumění řeči často chybují. Možnost chyb ve vstupních datech proto musí brát komponenta řízení dialogu v potaz. Vypořádat se s nejistotou lze pomocí jejího zakomponování do modelu pro odhad dialogového stavu.

Cíle uživatele a další vlastnosti prostředí lze považovat za náhodné částečně pozorovatelné proměnné. Pravděpodobnostní rozložení těchto náhodných proměnných dává dobře definovanou reprezentaci nejistoty, navíc je možné je úsporně reprezentovat pomocí Bayesovské sítě.

Jedním z možných modelů dialogového stavu je generativní model. Definujeme množinu dialogových stavů \mathcal{S} . Předpokládáme, že pozorování závisí podmíněně pouze na stavu prostředí $s \in \mathcal{S}$ a definujeme pravděpodobnostní rozdělení pro pozorování $o \in \mathcal{O}$, $p(o \mid s)$. Dále předpokládáme, že stav prostředí je závislý pouze na stavu v předchozí obrátce a na poslední akci systému. Tato závislost je zachycena v přechodové pravděpodobnosti $p(s_{t+1} \mid s_t, a_t)$. Předpoklad, že dialogový stav závisí pouze na minulé hodnotě se nazývá Markovská vlastnost.

Pokud vezmeme předchozí předpoklady, tak lze využít Bayesovský přístup pro počítání s nejistotou. Celou historii do obrátky t označíme jako h_t . Můžeme spočítat pravděpodobnost stavu v čase $t+1$ po přijetí nového pozorování $o_{t+1} = o'$.

$$p(s_{t+1} = s' \mid h_{t+1}) \propto p(o_{t+1} = o' \mid s_{t+1} = s') \quad (1.1)$$

$$\sum_{s \in \mathcal{S}} p(s_t = s \mid h_t) p(s_{t+1} = s' \mid s_t = s, a_t = a)$$

Nyní můžeme definovat stav důvěry v čase t , b_t , jako pravděpodobnost stavů dáno všemi pozorováními až do času t . Množina možných stavů důvěry jsou pravděpodobnostní distribuce přes všechny možné dialogové stavy $\mathcal{B} = \prod(\mathcal{S})$.

Můžeme přepsat rovnici (??) s pomocí stavů důvěry.

$$b(s_{t+1}) \propto p(o_{t+1} \mid s_{t+1}) \sum_{s \in \mathcal{S}} b(s_t) p(s_{t+1} \mid s_t, a_t) \quad (1.2)$$

Rovnice (1.2) nám dává předpis pro přechodovou funkci. V praxi ovšem bude množina možných hodnot pro stav s_t příliš velká, protože stav musí obsahovat všechny informace potřebné pro rozhodování. To znamená celou historii dialogu a cíle uživatele. Pokud systém obsahuje sloty, tak každá kombinace hodnot slotu je jedním možným cílem uživatele. Tedy velikost stavového prostoru roste exponenciálně.

1.3.1 Aktualizace stavu

Efektivní metodou pro aktualizaci dialogového stavu je použití Dynamických Bayesovských sítí [26]. Bayesovské sítě umožňují efektivní výpočet (inferenci) využitím podmíněných nezávislostí mezi sloty. Stále ovšem zůstává problém s výpočtem, pokud i jednotlivé sloty obsahují příliš mnoho hodnot. Lze použít aproximace a počítat jen s k nejpravděpodobnějšími hodnotami [27].

Alternativním zjednodušením je rozdělit stav prostředí do skupin. Tento přístup se nazývá Hidden Information State (HIS) [36]. Základním předpokladem zde musí být, že uživatel nezmění svůj cíl v průběhu dialogu. Pak lze efektivně provádět aktualizaci, protože rovnice pro aktualizace pravděpodobnosti se nemění mezi jednotlivými skupinami.

V této práci se budeme zabývat prvním přístupem, tedy použitím Bayesovských sítí. Pro inferenci použijeme Loopy Belief Propagation (LBP) algoritmus, který je aproximativní metodou pro sítě s diskrétními náhodnými proměnnými. Pro učení parametrů představíme Expectation Propagation (EP) algoritmus. EP je zobecněním LBP na libovolné pravděpodobnostní rozdělení.

2. Bayesovské sítě a inference

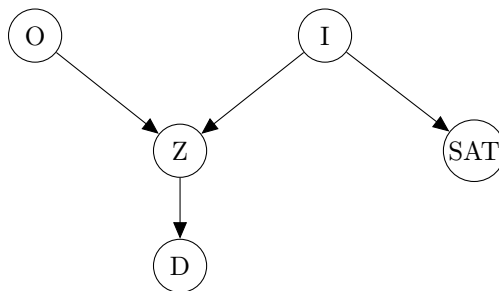
V této kapitole představíme Bayesovské sítě, grafický model pro efektivní reprezentaci pravděpodobnostních rozdělání a nezávislostí mezi náhodnými proměnnými. Bayesovská síť zároveň vytváří koncept pro inferenci, tedy zodpovídání dotazů nad proměnnými v síti. Ukážeme si několik přístupů k inferenci, začneme naivním výpočtem vycházejícím přímo z definice. Následně využijeme vlastností sítě a konceptů dynamického programování pro jeho zlepšení. Analýzou složitosti exaktní inference dojdeme k závěru, že pro větší a komplexnější modely bude třeba se uchýlit k aproximacím.

Nejprve aproximujeme sdruženou pravděpodobnostní distribuci součinem marginálních distribucí a představíme Loopy Belief Propagation algoritmus. Stále můžeme použít pouze diskrétní pravděpodobnostní distribuce se známými parametry. Pro učení parametrů lze použít Expectation Maximization metodu, pro dialogové systémy je ovšem těžké získat dostatek učících dat. Dostaneme se k metodě Expectation Propagation, která je zobecněním LBP a je možné ji použít pro libovolné pravděpodobnostní rozdělání. Díky ní budeme schopni vytvořit generativní model pro aktualizaci dialogového stavu, který bude pracovat stejně jako LBP, ale bude schopen adaptace za běhu dialogového systému.

2.1 Bayesovské sítě

Bayesovské sítě jsou pravděpodobnostní grafický model, který využívá podmíněných nezávislostí pro úspornou reprezentaci sdružené pravděpodobnosti. Bayesovská síť je orientovaný acyklický graf, jehož vrcholy jsou náhodné proměnné a hrany odpovídají přímé závislosti jednoho uzlu na druhý. Pro každou náhodnou proměnnou v síti platí, že její pravděpodobnost je jednoznačně určena jejími rodiči v grafu. Podmíněná pravděpodobnostní distribuce (CPD) proměnné X popisuje pravděpodobnost proměnné X dáno jejími rodiči, $P(X \mid \text{parents}(X))$. Pokud proměnná nemá žádné rodiče, pak její podmíněná pravděpodobnostní distribuce je ekvivalentní marginální pravděpodobnostní distribuci.

Příklad Student [12]: firma zvažuje, zda-li přijme studenta. Firma chce přijímat chytré studenty, ale nesmí je testovat na inteligenci (I) přímo. Má však výsledek studentových SAT testů, které ovšem nemusí stačit pro správné zhodnocení inteligence. Požadují tedy i doporučení (D) od jednoho z učitelů. Učitel studentovi napíše doporučující dopis na základě známky (Z), kterou student získal z jeho předmětu. Předměty se ovšem liší v obtížnosti (O) a tak je studentova známka z předmětu závislá nejen na jeho inteligenci, ale také na obtížnosti před-



Obrázek 2.1: Bayesovská síť pro příklad se studentem.

mětu. Grafický model reprezentující tento problém je na obrázku 2.1.

V tomto modelu je několik nezávislostí. Obtížnost předmětu a inteligence studenta jsou zjevně nezávislé. Studentova známka z předmětu je závislá na obtížnosti předmětu a inteligenci studenta, ale je podmíněně nezávislá na jeho výsledku ze SAT, dáno studentovou inteligencí. Konečně doporučení, které student obdrží, je podmíněně nezávislé na všech ostatních proměnných, dáno studentovou známkou.

Sdruženou nezávislost tohoto modelu lze zapsat ve formě podmíněných pravděpodobnostních distribucí s pomocí řetízkového pravidla.

$$P(O, I, Z, S, D) = P(D | Z)P(Z | O, I)P(SAT | I)P(O)P(I) \quad (2.1)$$

Předpokládejme, že obtížnost předmětu, inteligence studenta, doporučující dopis a výsledek SAT jsou binární proměnné. Zámka z předmětu je ternární proměnná. Pokud bychom zapsali sdruženou pravděpodobnost ve formě tabulky, tak se dostaneme k 48 položkám. Díky rozdělení do podmíněných pravděpodobnostních distribucí, které nám Bayesovská síť poskytuje, se dostáváme k $2 + 2 + 12 + 4 + 6 = 26$ položkám. Tedy i v tomto jednoduchém modelu dochází ke značné úspoře.

2.2 Inference v Bayesovských sítích

Bayesovské sítě reprezentují pravděpodobnostní model a umožňují nám nad ním provádět dotazy. Můžeme se například ptát na marginální pravděpodobnost jednotlivých proměnných. Tu získáme marginalizací sdružené pravděpodobnosti. V příkladu se studentem můžeme chtít znát marginální pravděpodobnost známek. Pro její nalezení musíme ze sdružené pravděpodobnosti vyysčítat všechny ostatní proměnné.

$$P(Z) = \sum_{O, I, S, D} P(D | Z)P(Z | O, I)P(SAT | I)P(O)P(I) \quad (2.2)$$

Další a asi nejčastější dotaz nastává, pokud některé náhodné proměnné pozorujeme. Pak chceme znát pravděpodobnost jiných proměnných dáno naším pozorováním, $P(\mathbf{X} \mid \mathbf{E} = \mathbf{e})$, kde \mathbf{X} jsou dotazované proměnné, \mathbf{E} jsou pozorované proměnné a \mathbf{e} jsou pozorované hodnoty. Z definice podmíněné pravděpodobnosti dostáváme

$$P(\mathbf{X} \mid \mathbf{E} = \mathbf{e}) = \frac{P(\mathbf{X}, \mathbf{E} = \mathbf{e})}{P(\mathbf{E} = \mathbf{e})} \quad (2.3)$$

Každou instanci jmenovatele $P(\mathbf{X}, \mathbf{E} = \mathbf{e})$ jde spočítat sumou sdružených pravděpodobností s ohodnoceními proměnných, které jsou kompatibilní s pozorováním a aktuální instancí. Pokud počítáme instanci $P(\mathbf{X} = \mathbf{x}, \mathbf{E} = \mathbf{e})$, pak získáme výsledek marginalizací všech proměnných, kromě \mathbf{X} a \mathbf{E} , které jsou fixovány. Pokud tedy množinu všech proměnných bez \mathbf{X} a \mathbf{E} označíme $\mathcal{W} = \mathcal{X} - \mathbf{X} - \mathbf{E}$, pak pravděpodobnost dané instance je

$$P(\mathbf{X} = \mathbf{x}, \mathbf{E} = \mathbf{e}) = \sum_{\mathbf{w}} P(\mathbf{x}, \mathbf{e}, \mathbf{w}) \quad (2.4)$$

Pro výpočet normalizační konstanty $P(\mathbf{E})$ musíme opět marginalizovat sdruženou pravděpodobnost anebo si můžeme povšimnout, že platí

$$P(\mathbf{E} = \mathbf{e}) = \sum_{\mathbf{x}} P(\mathbf{x}, \mathbf{e}) \quad (2.5)$$

a tedy můžeme použít už vypočítané hodnoty.

2.2.1 Exaktní inference

V předchozí části jsme viděli jak lze pomocí definice podmíněné pravděpodobnosti a marginalizace sdružené pravděpodobnosti najít odpověď na libovolný dotaz. Nyní si ukážeme algoritmus, který využívá struktury Bayesovské sítě pro inferenci a navíc díky metodám dynamického programování umožňuje samotný výpočet urychlit. Nakonec ovšem zjistíme, že pro velké sítě, které nás většinou zajímají nejvíce, nám přesná inference nebude stačit a musíme se uchýlit k aproximacím.

Začneme s inferencí v jednoduchém modelu $A \rightarrow B \rightarrow C \rightarrow D$. Sdružená pravděpodobnost $P(A, B, C, D)$ je součinem jednotlivých podmíněných pravděpodobnostních distribucí

$$P(A, B, C, D) = P(D \mid C) P(C \mid B) P(B \mid A) P(A) \quad (2.6)$$

Pokud nyní budeme chtít spočítat marginální distribuci D , tak musíme mar-

ginalizovat všechny ostatní proměnné

$$P(D) = \sum_{A,B,C} P(D | C) P(C | B) P(B | A) P(A) \quad (2.7)$$

Můžeme si povšimnout, že spousta členů se bude počítat vícekrát. Využitím metod dynamického programování a přeuspořádáním sum si můžeme mezivýsledky uložit a použít vícekrát.

$$P(D) = \sum_C P(D | C) \sum_B P(C | B) \sum_A P(B | A) P(A) \quad (2.8)$$

Při výpočtu pak nejprve spočítáme $\psi_1(A, B) = P(B | A) P(A)$, pak vysčítáme proměnnou A a získáme $\tau_1(B) = \sum_A \psi_1(A, B)$. Pokračujeme obdobně

$$\psi_2(B, C) = P(C | B) \tau_1(B) \quad (2.9)$$

$$\tau_2(C) = \sum_B \psi_2(B, C) \quad (2.10)$$

A nakonec spočítáme finální marginální pravděpodobnost

$$\psi_3(D, C) = P(D | C) \tau_2(C) \quad (2.11)$$

$$P(D) = \sum_C \psi_3(D, C) \quad (2.12)$$

Nechť \mathcal{X} je množina náhodných proměnných. $Val(\mathcal{X})$ definujeme jako množinu možných ohodnocení proměnných \mathcal{X} . Potom definujeme faktor ϕ jako zobrazení z $Val(\mathcal{X})$ do \mathbb{R} . Faktor je nezáporný, pokud všechny jeho obrazy jsou nezáporné. Doménu faktoru ϕ a značíme jako $Dom(\phi)$.

Faktor, jehož doménu tvoří diskrétní proměnné, si můžeme představit jako tabulku, která obsahuje jednu hodnotu pro každé možné ohodnocení proměnných z domény.

Nechť $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ jsou tři disjunktní množiny náhodných proměnných. Nechť $\phi_1(\mathbf{X}, \mathbf{Y})$ a $\phi_2(\mathbf{Y}, \mathbf{Z})$ jsou faktory. Definujeme součin faktorů $\phi_1 \times \phi_2$ jako faktor $\psi : Val(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) \rightarrow \mathbb{R}$ následovně:

$$\psi(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) = \phi_1(\mathbf{X}, \mathbf{Y}) \cdot \phi_2(\mathbf{Y}, \mathbf{Z})$$

Násobíme prvky se stejným ohodnocením společných proměnných \mathbf{Y} . Stejný princip použijeme pro všechny matematické operace.

Nechť \mathbf{X} je množina náhodných proměnných a $Y \notin \mathbf{X}$ náhodná proměnná. Nechť $\phi(\mathbf{X}, Y)$ je faktor. Definujeme marginalizaci Y ve ϕ , značenou $\sum_Y \phi$, jako

faktor ψ s doménou \mathbf{X} takový, že

$$\psi(\mathbf{X}) = \sum_Y \phi(\mathbf{X}, Y)$$

Této operaci také říkáme vysčítání Y ve ϕ .

Všechny operace s faktory probíhají po prvcích, pro které je ohodnocení náhodných proměnných z průniku domén faktorů stejné. Proto platí komutativita $\phi_1 \cdot \phi_2 = \phi_2 \cdot \phi_1$ a $\sum_X \sum_Y \phi = \sum_Y \sum_X \phi$. Dále platí asociativita součinu $(\phi_1 \cdot \phi_2) \cdot \phi_3 = \phi_1 \cdot (\phi_2 \cdot \phi_3)$. Nakonec můžeme vyměnit sumu a součin, pokud $X \notin \text{Dom}(\phi_1)$, potom $\sum_X (\phi_1 \cdot \phi_2) = \phi_1 \sum_X \phi_2$.

Sdruženou pravděpodobnost z minulého příkladu tedy můžeme přepsat do formy faktorů

$$P(A, B, C, D) = \phi_A \cdot \phi_B \cdot \phi_C \cdot \phi_D \quad (2.13)$$

Opět se pokusíme spočítat marginální pravděpodobnost proměnné D .

$$P(D) = \sum_C \sum_B \sum_A \phi_A \phi_B \phi_C \quad (2.14)$$

$$= \sum_C \phi_D \cdot \left(\sum_B \phi_C \cdot \left(\sum_A \phi_A \cdot \phi_B \right) \right) \quad (2.15)$$

Přesuny sum můžeme provést díky doméně jednotlivých faktorů. Faktory ϕ_C a ϕ_D neobsahují proměnnou A a tedy je můžeme vytknout před sumu přes A . Stejně tak faktor ϕ_D neobsahuje proměnnou B a opět jej můžeme vytknout před sumu přes B . Tyto úpravy můžeme provádět v libovolném pořadí, pokud vždy platí, že vysčítáme proměnnou X až poté, co spolu vynásobíme všechny faktory, které ji obsahují.

V obecnosti vždy počítáme výraz ve tvaru

$$\sum_X \prod_{\phi \in \Phi} \phi.$$

Z tohoto také vychází název pro tuto metodu: sum-product. Jednoduchý algoritmus pro exaktní inferenci využívající tuto metodu se nazývá eliminace proměnných. Základní myšlenka je, že máme dán seznam náhodných proměnných v pořadí, v jakém se mají eliminovat. Pro eliminaci proměnné X je třeba nejprve vynásobit všechny faktory obsahující X a následně X z tohoto součinu vysčítat. Tak získáme faktor, který už tuto proměnnou neobsahuje. Tedy jsme ji eliminovali. Eliminace proměnných je popsána v algoritmu 1.

Algoritmus 1 Eliminace proměnných

function SUM-PRODUCT-VE(Φ, \mathbf{X}, \prec)

Φ množina všech faktorů.

\mathbf{X} množina náhodných proměnných, které mají být eliminovány.

\prec pořadí proměnných, v jakém mají být eliminovány.

Nechť X_1, \dots, X_k je seřazení proměnných z \mathbf{X} , t.ž. $X_i \prec X_j \Leftrightarrow i < j$.

for $i = 1 \dots k$ **do**

$\Phi \leftarrow \text{Sum-Product-Eliminate-Var}(\Phi, X_i)$

end for

$\phi^* \leftarrow \prod_{\phi \in \Phi} \phi$

return ϕ^*

end function

function SUM-PRODUCT-ELIMINATE-VAR(Φ, X)

Φ množina všech faktorů.

X , proměnná, která má být eliminována.

$\Phi' \leftarrow \{\phi \in \Phi : X \in \text{Dom}(\phi)\}$

$\Phi'' \leftarrow \Phi - \Phi'$

$\psi \leftarrow \prod_{\phi \in \Phi'} \phi$

$\tau \leftarrow \sum_X \psi$

return $\Phi'' \cup \{\tau\}$

end function

Nechť \mathbf{X} je množina náhodných proměnných, nechť Φ je množina faktorů, t.ž. pro každé $\phi \in \Phi$, $\text{Dom}(\phi) \subseteq \mathbf{X}$. Nechť $\mathbf{Y} \subset \mathbf{X}$ je množina dotazovaných náhodných proměnných a nechť $\mathbf{Z} = \mathbf{X} - \mathbf{Y}$. Pak pro každé seřazení \prec nad \mathbf{Z} , Sum-Product-VE(Φ, \mathbf{Z}, \prec) vrátí faktor $\phi^*(\mathbf{Y})$ takový, že

$$\phi^*(\mathbf{Y}) = \sum_{\mathbf{Z}} \prod_{\phi \in \Phi} \phi$$

Nyní provedeme analýzu algoritmu eliminace proměnných. Předpokládejme, že na vstupu je n proměnných. Bayesovská síť obsahuje pro každou proměnnou jeden faktor. Pro jednoduchost budeme předpokládat, že algoritmus bude eliminovat všechny proměnné. Běh algoritmu se skládá z jednotlivých eliminačních kroků, v každém je eliminována jedna proměnná.

V eliminačním kroku je vybrána proměnná X_i . Všechny faktory, které ji obsahují jsou vynásobeny a vytvoří jeden velký faktor ψ_i . Proměnná X_i je pak vysčítána z tohoto faktoru. Počet operací pro jeden eliminační krok tedy závisí na velikosti faktoru ψ_i , označme ji N_i . Maximum z velikostí faktorů označme $N_{\max} = \max_i N_i$.

Nyní se zaměříme na počet násobení. Celkem vznikne $n + m$ faktorů, z toho

m při vysčítání proměnné. Každý z těchto faktorů je zahrnut do součinu pouze jednou, když je eliminována nějaká proměnná, kterou obsahuje. Cena násobení faktorů pro vznik ψ_i je nejvýše N_i . Celkový počet násobení tedy bude nejvýše $(n + m)N_{max}$ což je $\mathcal{O}(nN_{max})$.

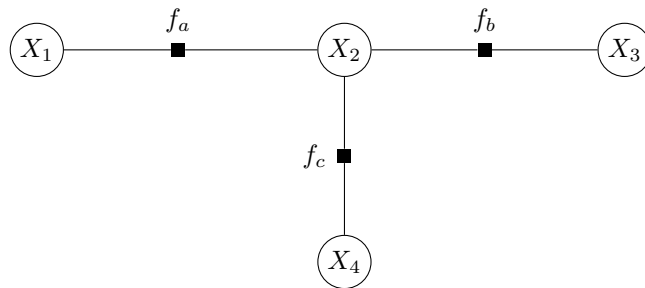
Pokud k je maximum z velikostí domén proměnných, pak velikost faktoru obsahujícího n proměnných může být až k^n . Složitost eliminace proměnných je tedy dominována velikostí faktorů vznikajících při výpočtu a je exponenciální. Navíc bylo dokázáno, že výběr nejlepšího pořadí proměnných pro eliminaci je NP-těžký [1].

2.2.2 Posílání zpráv ve faktor grafu

Algoritmy exaktní inference naráží při reálném použití na příliš velkou složitost násobení faktorů. Hlavním problémem je velikost sdružené pravděpodobnosti, která roste exponenciálně s počtem náhodných proměnných. Většinou nás ovšem zajímá marginální pravděpodobnost jedné nebo jen mála proměnných. Je tedy zbytečné počítat celou sdruženou pravděpodobnost, abychom z ní pak vysčítali většinu proměnných. Řešení se nabízí ve formě aproximace faktorizací sdružené pravděpodobnosti.

$$P(X_1, \dots, X_n) = \prod_i P(X_i)$$

Pro výpočet s faktorizovanou distribucí si zavedeme novou datovou strukturu, tzv. faktor graf. Faktor graf je bipartitní graf, kde jednu partitu tvoří faktory a druhou partitu tvoří náhodné proměnné. Hrany ve faktor grafu vedou vždy mezi proměnnou a faktorem, v jehož doméně leží.



Obrázek 2.2: Příklad faktor grafu se čtyřmi náhodnými proměnnými X_1, X_2, X_3, X_4 a třemi faktory f_a, f_b, f_c .

Nejprve začneme s inferencí na stromech. Pokud zafixujeme jednu náhodnou proměnnou X ve faktor grafu, pak sdruženou pravděpodobnost můžeme spočítat

jako

$$P(\mathbf{X}) = \prod_{s \in ne(X)} F_s(X, \mathbf{X}_s), \quad (2.16)$$

kde $ne(X)$ jsou faktory obsahující proměnnou X (tedy sousedi ve faktor grafu), F_s je součin všech faktorů v podstromu určeném faktorem f_s a \mathbf{X}_s je množina všech proměnných v daném podstromu.

Pro výpočet marginální pravděpodobnosti X substitujeme (2.16) do výpočtu marginální pravděpodobnosti ze sdružené a po výměně sumy a produktu dostaneme

$$P(X) = \prod_{s \in ne(X)} \sum_{\mathbf{X}_s} F_s(X, \mathbf{X}_s) \quad (2.17)$$

$$= \prod_{s \in ne(X)} \mu_{f_s \rightarrow X}(X) \quad (2.18)$$

Zavedli jsme funkce

$$\mu_{f_s \rightarrow X} \equiv \sum_{\mathbf{X}_s} F_s(X, \mathbf{X}_s), \quad (2.19)$$

budeme je nazývat zprávami z faktoru f_s do proměnné X .

Každé $F_s(X, \mathbf{X}_s)$ je popsáno podgrafem faktor grafu a tedy může být znova faktorizováno. Namísto proměnné X nyní vezmeme faktor f_s . Náhodné proměnné sousedící s faktorem f_s bez X si označíme X_1, \dots, X_M . Součin faktorů v podstro-mech určených těmito proměnnými označíme $G_i(X_i, \mathbf{X}_{si})$. Faktor $F_s(X, \mathbf{X}_s)$ tedy můžeme přepsat jako

$$F_s(X, \mathbf{X}_s) = f_s(X, X_1, \dots, X_M) G_1(X_1, \mathbf{X}_{s1}), \dots, G_M(X_M, \mathbf{X}_{sM}) \quad (2.20)$$

Pokud substitujeme přepsaný faktor $F_s(X, \mathbf{X}_s)$ do definice zprávy z faktoru, dostaneme

$$\mu_{f_s \rightarrow X}(X) = \sum_{X_1} \cdots \sum_{X_M} f_s(X, X_1, \dots, X_M) \prod_{m \in ne(f_s) \setminus X} \sum_{\mathbf{X}_{sm}} G_m(X_m, \mathbf{X}_{sm}) \quad (2.21)$$

$$= \sum_{X_1} \cdots \sum_{X_M} f_s(X, X_1, \dots, X_M) \prod_{m \in ne(f_s) \setminus X} \mu_{X_m \rightarrow f_s}(X_m) \quad (2.22)$$

Zavedli jsme další funkce

$$\mu_{X_m \rightarrow f_s}(X_m) \equiv \sum_{\mathbf{X}_{sm}} G_m(X_m, \mathbf{X}_{sm}), \quad (2.23)$$

budeme je nazývat zprávami z proměnné X_m do faktoru f_s .

Z rovnice 2.18 vidíme, že marginální pravděpodobnost proměnné vypočítáme

jako součin zpráv ze všech okolních faktorů. Každou z těchto zpráv můžeme spočítat jako součin faktoru a zpráv přicházejících ze sousedních proměnných, kromě proměnné, které chceme zprávu posílat. Zbývá nám tedy zjistit v jakém tvaru jsou zprávy z proměnných do faktoru.

Stejně jako u $F_s(X, \mathbf{X}_s)$ nám tady $G_m(X_m, \mathbf{X}_{sm})$ definuje podgraf faktor grafu. V kořeni tohoto podgrafu leží proměnná X_m a tím se dostáváme na už známý případ. Budeme ignorovat faktor f_s , protože neleží v podgrafu určeném $G_m(X_m, \mathbf{X}_{sm})$, a pak

$$G_m(X_m, \mathbf{X}_{sm}) = \prod_{l \in ne(X_m) \setminus f_s} F_l(X_m, \mathbf{X}_{ml}) \quad (2.24)$$

Přepsáno ve formě zpráv

$$\mu_{X_m \rightarrow f_s}(X_m) = \prod_{l \in ne(X_m) \setminus f_s} \sum_{\mathbf{X}_{ml}} F_l(X_m, \mathbf{X}_{ml}) \quad (2.25)$$

$$= \prod_{l \in ne(X_m) \setminus f_s} \mu_{f_l \rightarrow X_m}(X_m) \quad (2.26)$$

Zpráva z náhodné proměnné je tedy součinem zpráv ze všech ostatních faktorů.

Zprávy tedy dokážeme počítat rekurzivně, chybí nám ovšem pravidla pro zprávy z listů. V případě faktoru je odchozí zpráva ekvivalentní faktoru.

$$\mu_{f \rightarrow X}(X) = f(X) \quad (2.27)$$

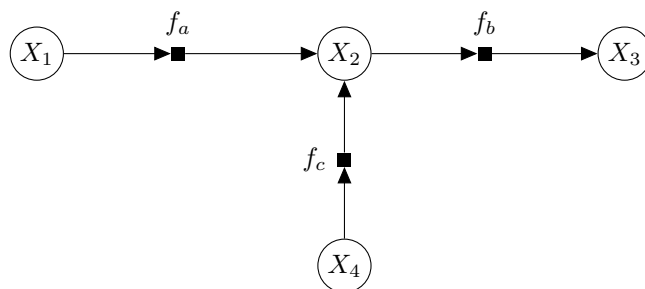
V případě náhodné proměnné je odchozí zpráva uniformní

$$\mu_{X \rightarrow f}(X) = 1 \quad (2.28)$$

Pro pozorovanou proměnnou je zpráva vždy rovna pozorované hodnotě bez ohledu na sousední faktory.

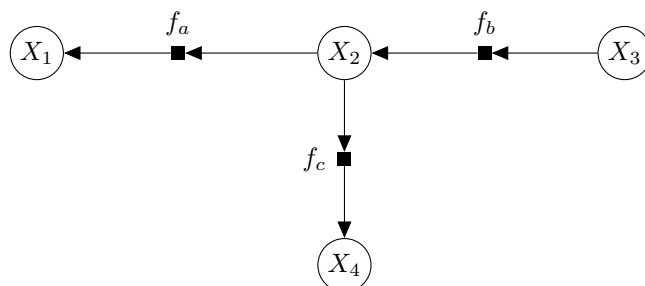
2.2.3 Belief Propagation

Algoritmus exaktní inference na stromech s pomocí posílání zpráv se nazývá Belief Propagation [20]. Zpráva z vrcholu faktor grafu může být poslána do jiného vrcholu, až když byly obdrženy zprávy ze všech ostatních vrcholů. Jakmile vrchol obdržel zprávy ze všech vrcholů, lze spočítat jeho marginální pravděpodobnost. Strom můžeme zakořenit v libovolné náhodné proměnné a posílat zprávy z listů do kořene. Kořen pak obdrží všechny zprávy ze všech sousedních faktorů a tedy je možné spočítat jeho marginální pravděpodobnost. Tomuto se říká dopředný krok propagace.



Obrázek 2.3: Směr posílání zpráv pro strom zakořeněný ve vrcholu X_3 v dopředném kroce.

Belief Propagation algoritmus tedy umožňuje na stromech spočítat marginální pravděpodobnost jedné proměnné s lineárním počtem poslaných zpráv. Při rozšíření na spočítání marginální pravděpodobnosti všech proměnných není třeba algoritmus pouštět n -krát. Stačí si uvědomit, že všem vrcholům chybí pouze zpráva od faktoru na cestě ke kořenu. A my můžeme z kořene poslat zprávy zpět k listům hned, jak obdržíme všechny příchozí zprávy. Této části se říká zpětný krok propagace.



Obrázek 2.4: Směr posílání zpráv pro strom zakořeněný ve vrcholu X_3 ve zpětném kroce.

Po propagaci všech zpráv až k listům už každá proměnná získala zprávy od všech sousedních faktorů a tedy je možné spočítat marginální pravděpodobnost všech proměnných.

2.2.4 Loopy Belief Propagation

Algoritmus Belief Propagation funguje pro inferenci na stromech. Hlavním důvodem, proč nelze použít Belief Propagation na obecných grafech je, že můžeme narazit na cyklus v grafu. Žádný z vrcholů v tomto cyklu nebude nikdy mít dostatek příchozích zpráv, aby mohl nějakou zprávu odeslat.

Algoritmus Loopy Belief Propagation řeší tento problém relaxací podmínky na příchozí zprávy. Pro odeslání zprávy do uzlu není třeba znát zprávy ze všech ostatních sousedních vrcholů. Chybějící zprávy jsou nastaveny na jedničku. Loopy Belief Propagation je iterativní algoritmus. V grafu nemusí existovat vrchol, který by mohl poslat zprávu k nějakému ze svých sousedů. Je tedy třeba vybrat vrchol podle jiné strategie a z něj poslat zprávy do všech sousedních vrcholů. V další iteraci je pak vybrán zase jiný vrchol. Iterace končí ve chvíli, kdy se už nemění hodnoty zpráv.

Vzhledem k tomu, že posílané zprávy už neodpovídají skutečným faktorům, je důležitá otázka, zda Loopy Belief Propagation vůbec nalezne správné marginální pravděpodobnosti. Existují podmínky, za kterých algoritmus bude konvergovat [25]. V obecném případě může algoritmus konvergovat ke špatným pravděpodobnostem anebo nemusí konvergovat vůbec a pak dochází k oscilacím. Pokud algoritmus nekonverguje ke správným pravděpodobnostem, tak není ani možné se k nim přiblížit nebo je odvodit z oscilací [17]. V praxi ovšem k problémům většinou nedochází a inference konverguje v rozumném čase.

Popis LBP

Popis Loopy Belief Propagation je v algoritmu 2. Při inferenci ve faktor grafu je nejprve třeba inicializovat všechny zprávy. Posílání zpráv budeme iterovat, dokud zprávy nekonvergují. Druhou možností je nastavit pevný počet iterací. Pak vybíráme vrcholy a z každého pošleme zprávu do všech okolních. Způsobem výběru vrcholů se budeme zabývat později. I náhodný výběr vrcholů povede k výsledku. Dodání externí informace či chytřejší heuristiky ovšem může výpočet výrazně urychlit.

Poslání zprávy se liší podle toho, zda-li posíláme z proměnné do faktoru anebo naopak. V obou případech vynásobíme příchozí zprávy ze všech sousedních vrcholů kromě toho, do kterého zprávu posíláme. Při posílání zprávy z faktoru f do proměnné X musíme navíc vynásobit příchozí zprávy hodnotou f , následně vysčítat všechny proměnné až na X .

Ve faktor grafu se tedy zprávy šíří pouze přes pravděpodobnostní rozdělení jedné proměnné. Právě to může vést k oscilacím nebo konvergenci ke špatným hodnotám. Příkladem je Skrytý Markovský model se dvěma skrytými binárními proměnnými a pozorováním pro každou skrytou proměnnou. Pokud je mezi proměnnými závislost XOR, pak nikdy nebudeme pozorovat $(1, 1)$, ale přesto bude každá proměnná konvergovat k rovnoměrnému rozdělení. A tedy sdružená pravděpodobnost pro pozorování $(1, 1)$ dostane nenulovou pravděpodobnost.

Při počítání zpráv často musíme pronásobit spoustu zpráv z okolních vrcholů. Přitom se zprávy zas tak často nemění. Častou optimalizací je pro každý vrchol

Algoritmus 2 Loopy Belief Propagation

```
function LBP( $F, S$ )  
   $F$  – faktor graf  
   $S$  – strategie  
  
  INIT-FACTOR-GRAPH( $F$ )  
  repeat  
    for vrchol  $v \in F$  vybraný podle strategie  $S$  do  
      for soused  $n$  vrcholu  $v$  do  
        if  $v$  je faktor then  
           $\mu_{v \rightarrow n}(n) \leftarrow \text{MESSAGE-TO-VAR}(v, n)$   
        else  
           $\mu_{v \rightarrow n}(v) \leftarrow \text{MESSAGE-TO-FACTOR}(v, n)$   
        end if  
      end for  
    end for  
  until konvergence  
  for každá proměnná  $X$  do  
     $P(X) = \prod_{f \in ne(X)} \mu_{f \rightarrow X}(X)$   
  end for  
end function  
  
function INIT-FACTOR-GRAPH( $F$ )  
   $F$  – faktor graf  
  
  for každou proměnnou  $X \in F$  do  
    for sousední faktor  $f$  do  
       $\mu_{X \rightarrow f}(X) \leftarrow 1$   
       $\mu_{f \rightarrow X}(X) \leftarrow 1$   
    end for  
  end for  
end function  
  
function MESSAGE-TO-VAR( $f, v$ )  
   $f$  – zdrojový faktor  
   $v$  – cílová proměnná  
  
   $\mathbf{X} \leftarrow \text{Dom}(f)$   
   $\mu_{f \rightarrow v}(v) \leftarrow \sum_{\mathbf{X} \setminus v} f(\mathbf{X}) \prod_{u \in ne(f) \setminus v} \mu_{u \rightarrow f}(u)$   
end function  
  
function MESSAGE-TO-FACTOR( $v, f$ )  
   $v$  – zdrojová proměnná  
   $f$  – cílový faktor  
  
   $\mu_{v \rightarrow f}(v) \leftarrow \prod_{g \in ne(v) \setminus f} \mu_{g \rightarrow v}(v)$   
end function
```

si pamatovat součin všech jeho zpráv

$$b(v) = \begin{cases} \prod_{u \in ne(v)} \mu_{u \rightarrow v}(v) & v \text{ je proměnná} \\ f_v(\mathbf{X}) \prod_{u \in ne(v)} \mu_{u \rightarrow v}(u) & v \text{ je faktor} \end{cases} \quad (2.29)$$

Při odesílání zprávy z vrcholu v do vrcholu u pouze $b(v)$ vydělíme zprávou $\mu_{u \rightarrow v}$. Pokud je v faktor, pak ještě vysčítáme všechny proměnné kromě u a zpráva je připravena k odeslání. Hodnotu $b(v)$ můžeme aktualizovat po výběru vrcholu v pro posílání zpráv nebo po každém přijetí zprávy.

Pro pozorované proměnné je třeba metodu MESSAGE-TO-FACTOR upravit tak, aby pouze posílala pozorovanou hodnotu.

Implementace Loopy Belief Algoritmu i se strategiemi pro výběr vrcholů bude popsána v kapitole 4.

2.3 Propagace s aproximovanými zprávami

V předchozí sekci byla faktorizována sdružená pravděpodobnost a tak došlo ke zjednodušení inference. Mezi jednotlivými vrcholy byly stále posílány exaktní zprávy. V této sekci se zaměříme na inferenci v modelech, kde není možné spočítat zprávy exaktně a je třeba je aproximovat. Tento problém nastává u některých modelů se spojitými náhodnými proměnnými.

Předpokládáme, že máme pravděpodobnostní grafický model, reprezentující sdruženou pravděpodobnost dat a pozorování pomocí součinu faktorů.

$$P(\mathbf{X}, \mathbf{E}) = \prod_i f_i(\mathbf{X}) \quad (2.30)$$

Nás zajímá aposteriorní distribuce $P(\mathbf{X} \mid \mathbf{E})$ pro zjištění stavu, stejně tak jako pravděpodobnost pozorování $P(\mathbf{E})$ pro normalizaci. Aposteriorní distribuci vyjádříme ze sdružené.

$$P(\mathbf{X} \mid \mathbf{E}) = \frac{1}{P(\mathbf{E})} \prod_i f_i(\mathbf{X}) \quad (2.31)$$

a pravděpodobnost pozorování je dána

$$P(\mathbf{E}) = \int \prod_i f_i(\mathbf{X}) d\mathbf{X} \quad (2.32)$$

Pro aproximativní inferenci ukážeme algoritmus Expectation propagation (EP) [16].
Vytvoříme aproximaci aposteriorní distribuce, která je také dána součinem fak-

torů

$$q(\mathbf{X}) = \frac{1}{Z} \prod_i \tilde{f}_i(\mathbf{X}), \quad (2.33)$$

kde každý faktor \tilde{f}_i je aproximace odpovídající skutečnému faktoru f_i a faktor $\frac{1}{Z}$ je normalizační konstanta. Aproximované faktory musíme zkonstruovat tak, abychom byli schopni provádět inferenci.

Pro měření vzdálenosti aproximovaného rozdělení od skutečného používáme Kullback-Leiblerovu divergenci (KL) [13]. KL divergence, také známá jako relativní entropie, mezi dvěma pravděpodobnostními rozděleními $p(x)$ a $q(x)$ je

$$KL(p||q) = \int p(x) \log \left(\frac{p(x)}{q(x)} \right) dx. \quad (2.34)$$

Divergence splňuje tři vlastnosti:

1. $KL(p||p) = 0$,
2. $KL(p||q) = 0$ právě tehdy když $p = q$,
3. $KL(p||q) > 0$ pro všechna p, q .

2.3.1 Rodina exponenciálních rozdělení

Rodina exponenciálních rozdělení [3] přes \mathbf{X} , dáno parametry $\boldsymbol{\eta}$ je definována jako množina distribucí ve tvaru

$$P(\mathbf{X} | \boldsymbol{\eta}) = h(\mathbf{X})g(\boldsymbol{\eta}) \exp(\boldsymbol{\eta}^T u(\mathbf{X})), \quad (2.35)$$

kde \mathbf{X} jsou náhodné proměnné, které mohou být diskrétní nebo spojitě, $\boldsymbol{\eta}$ jsou nazývány přirozené parametry rozdělení, $u(\mathbf{X})$ je funkce \mathbf{X} . Funkce $g(\boldsymbol{\eta})$ může být interpretována jako koeficient, který zajišťuje, že je distribuce normalizována a tedy splňuje

$$g(\boldsymbol{\eta}) \int h(\mathbf{X}) \exp(\boldsymbol{\eta}^T u(\mathbf{X})) d\mathbf{X} = 1 \quad (2.36)$$

Výhody distribucí z exponenciální rodiny si ukážeme na KL divergenci. Nechť počítáme divergenci $KL(p||q)$, kde $p(\mathbf{X})$ je zafixovaná distribuce a $q(\mathbf{X})$ je distribuce z exponenciální rodiny. Pokud zapíšeme KL divergenci jako funkci $\boldsymbol{\eta}$, tak dostaneme

$$KL(p||q) = -\log g(\boldsymbol{\eta}) - \boldsymbol{\eta}^T \mathbb{E}_{p(\mathbf{X})}[u(\mathbf{X})] + \text{const}. \quad (2.37)$$

KL divergenci můžeme minimalizovat tak, že nastavíme první derivaci podle $\boldsymbol{\eta}$ rovnou nule, z čehož dostaneme

$$-\nabla \log g(\boldsymbol{\eta}) = \mathbb{E}_{p(\mathbf{X})}[u(\mathbf{X})]. \quad (2.38)$$

Nyní se musíme podívat na $-\nabla \log g(\boldsymbol{\eta})$. Vezmeme derivaci obou stran (2.36) podle $\boldsymbol{\eta}$. Získáme

$$\begin{aligned} \nabla g(\boldsymbol{\eta}) \int h(\mathbf{X}) \exp(\boldsymbol{\eta}^T u(\mathbf{X})) d\mathbf{X} \\ + g(\boldsymbol{\eta}) \int h(\mathbf{X}) \exp(\boldsymbol{\eta}^T u(\mathbf{X})) u(\mathbf{X}) d\mathbf{X} = 0 \end{aligned} \quad (2.39)$$

Přeuspořádáním a znova použitím (2.35) a (2.36) získáme

$$-\frac{1}{g(\boldsymbol{\eta})} \nabla g(\boldsymbol{\eta}) = \int h(\mathbf{X}) \exp(\boldsymbol{\eta}^T u(\mathbf{X})) u(\mathbf{X}) d\mathbf{X} = \mathbb{E}[u(\mathbf{X})]. \quad (2.40)$$

Ve výsledku získáváme

$$-\nabla \log g(\boldsymbol{\eta}) = \mathbb{E}[u(\mathbf{X})] \quad (2.41)$$

Nový poznatek o funkci $g(\boldsymbol{\eta})$ z (2.41) můžeme substituovat do (2.38) a získáme

$$\mathbb{E}_{q(\mathbf{X})}[u(\mathbf{X})] = \mathbb{E}_{p(\mathbf{X})}[u(\mathbf{X})], \quad (2.42)$$

z čehož vidíme, že pro minimalizaci KL divergence nám stačí najít parametry rozdělení $q(\mathbf{X})$ tak, aby mělo stejné očekávané statistiky jako $p(\mathbf{X})$.

2.3.2 Expectation Propagation

Expectation propagation je algoritmus založený na aproximaci aposteriorní distribuce součinem aproximovaných faktorů

$$q(\mathbf{X}) = \frac{1}{Z} \prod_i \tilde{f}_i(\mathbf{X}). \quad (2.43)$$

Z praktických důvodů uvedených v předchozí sekci jsou aproximované faktory z exponenciální rodiny. Díky tomu bude jejich součin také z exponenciální rodiny.

Aproximující distribuci bychom chtěli nalézt pomocí minimalizace KL divergence mezi skutečnou a aproximovanou distribucí.

$$KL(p||q) = KL\left(\frac{1}{p(\mathbf{E})} \prod_i f_i(\mathbf{X}) \left\| \frac{1}{Z} \prod_i \tilde{f}_i(\mathbf{X})\right.\right) \quad (2.44)$$

Pro počítání KL divergence ovšem potřebujeme umět počítat se skutečnou pravděpodobností a když už s ní umíme efektivně počítat, proč ztrácet čas s aproximací.

Další možností je aproximovat jednotlivé faktory, díky tomu bychom v jednom kroku našli aproximace pro všechny faktory a měli bychom hotovo. Naším cílem

je ale nalézt nejlepší aproximaci celé aposteriorní pravděpodobnosti a to není v tomto případě zaručeno.

Expectation propagation sice aproximuje jednotlivé faktory, ale vždy v kontextu všech ostatních [5]. Nejprve jsou všechny inicializovány a pak jsou procházeny jeden po druhém a každý je aktualizován. Tento přístup je podobný metodám, které byly ukázány v předešlých sekcích. Předpokládejme, že chceme aktualizovat faktor $\tilde{f}_j(\mathbf{X})$. Nejprve odstraníme faktor z produktu a získáme $\prod_{i \neq j} \tilde{f}_i(\mathbf{X})$. Následně nalezneme novou hodnotu pro faktor \tilde{f}_j tak, aby pravděpodobnostní rozložení

$$q^{new}(\mathbf{X}) \propto \tilde{f}_j(\mathbf{X}) \prod_{i \neq j} \tilde{f}_i(\mathbf{X}) \quad (2.45)$$

bylo co nejblíže

$$f_j(\mathbf{X}) \prod_{i \neq j} \tilde{f}_i(\mathbf{X}). \quad (2.46)$$

Všechny faktory $i \neq j$ necháváme zafixované. Díky tomu zajistíme, že nová aproximace faktoru je nejpřesnější v oblastech, které mají největší aposteriorní pravděpodobnost definovanou zbývajících faktory.

Odstranění faktoru $\tilde{f}_j(\mathbf{X})$ z aktuální aproximace aposteriorní distribuce provedeme vytvořením nenormalizované distribuce

$$q^{\setminus j}(\mathbf{X}) = \frac{q(\mathbf{X})}{\tilde{f}_j(\mathbf{X})} \quad (2.47)$$

Mohli bychom počítat $q^{\setminus j}(\mathbf{X})$ jako součin všech faktorů kromě j , ale v praxi je dělení rychlejší. Tuto nenormalizovanou distribuci budeme nazývat neúplná distribuce. Její kombinací se skutečným faktorem $f_j(\mathbf{X})$ dostaneme distribuci

$$\frac{1}{Z_j} f_j(\mathbf{X}) q^{\setminus j}(\mathbf{X}), \quad (2.48)$$

kde Z_j je normalizační konstanta, dána

$$Z_j = \int f_j(\mathbf{X}) q^{\setminus j}(\mathbf{X}) d\mathbf{X}. \quad (2.49)$$

Nyní můžeme nalézt novou hodnotu pro faktor \tilde{f}_j minimalizací KL divergence

$$KL\left(\frac{f_j(\mathbf{X}) q^{\setminus j}(\mathbf{X})}{Z_j} \parallel q^{new}(\mathbf{X})\right). \quad (2.50)$$

To uděláme jednoduše použitím poznatku z (2.42), který říká, že nám stačí najít parametry q^{new} tak, aby postačující statistiky odpovídali momentům aproximovaného rozdělení (2.48).

Z nalezené aproximace q^{new} pak získáme novou hodnotu faktoru \tilde{f}_j z (2.45) vydělením zbývajících faktorů

$$\tilde{f}_j(\mathbf{X}) = K \frac{q^{new}(\mathbf{X})}{q^{\setminus j}(\mathbf{X})} \quad (2.51)$$

Koeficient K získáme tak, že obě strany (2.51) vynásobíme $q^{\setminus j}(\mathbf{X})$ a zintegrováním, čímž získáme

$$K = \int \tilde{f}_j(\mathbf{X}) q^{\setminus j}(\mathbf{X}) d\mathbf{X}, \quad (2.52)$$

použili jsme navíc toho, že $q^{new}(\mathbf{X})$ je normalizovaná distribuce. Pro nalezení hodnoty K opět použijeme srovnání momentů, v tomto případě nultých momentů:

$$\int \tilde{f}_j(\mathbf{X}) q^{\setminus j}(\mathbf{X}) d\mathbf{X} = \int f_j(\mathbf{X}) q^{\setminus j}(\mathbf{X}) d\mathbf{X} \quad (2.53)$$

Vidíme, že pravá strana je rovna Z_j a tedy platí rovnost $K = Z_j$. Díky tomu můžeme K nalézt přímo výpočtem integrálu (2.49). Bez tohoto poznatku bychom se dostali do problému, protože pro výpočet K by bylo potřeba \tilde{f}_j , které ale bez K neznáme.

Expectation Propagation je popsán v algoritmu 4. Dokud aproximované rozdělení nekonverguje, vybíráme jednotlivé faktory a aktualizujeme je podle popsaného postupu.

V praxi je třeba provést několik iterací, a v každé aktualizovat všechny faktory. Stejně jako u LBP zde nemáme žádnou garanci, že bude algoritmus konvergovat. Pro aproximace $q(\mathbf{X})$ v exponenciální rodině, pokud iterace konverguje, pak nalezené řešení je stacionární bod specifické potenciální energie [16].

Algoritmus 3 Expectation propagation

Inicializujeme všechny aproximativní faktory $\tilde{f}_i(\mathbf{X})$ na neinformativní.

Inicializujeme aposteriorní aproximace nastavením:

$$q(\mathbf{X}) \propto \prod_i \tilde{f}_i(\mathbf{X})$$

repeat

Vyber faktor $\tilde{f}_j(\mathbf{X})$, který bude aktualizován.

Odeber $\tilde{f}_j(\mathbf{X})$ z aposteriorní distribuce vydělením:

$$q^{\setminus j}(\mathbf{X}) = \frac{q(\mathbf{X})}{\tilde{f}_j(\mathbf{X})}$$

Vypočítej novou aposteriorní distribuci nastavením momentů q^{new} na hodnotu momentů $q^{\setminus j}(\mathbf{X})\tilde{f}_j(\mathbf{X})$, včetně vyhodnocení normalizační konstanty

$$Z_j = \int q^{\setminus j}(\mathbf{X})\tilde{f}_j(\mathbf{X})d\mathbf{X}$$

Nastav novou hodnotu faktoru:

$$\tilde{f}_j(\mathbf{X}) = Z_j \frac{q^{new}(\mathbf{X})}{q^{\setminus j}(\mathbf{X})}$$

until konvergence

Spočítej aproximaci pozorování:

$$p(\mathbf{E}) \simeq \int \prod_i \tilde{f}_i(\mathbf{X})d\mathbf{X}$$

3. Učení parametrů

Představená metoda LBP v minulé kapitole funguje pro modely, kde máme nastavené parametry faktorů. V této kapitole si představíme grafický model pro diskrétní proměnné, u kterého jsou i parametry faktorů proměnné a je možné pro ně použít Bayesovský přístup. Díky tomu bude možné nalézt aposteriorní distribuci pro tyto parametry a tedy je naučit podle dat.

Inferenci v tomto upraveném modelu už není možné dělat pomocí LBP a budeme muset použít algoritmus Expectation Propagation.

3.1 Grafický model

Model se skládá z faktorů a proměnných. Nechť máme vybraný faktor f_β , tento faktor je spojený s několika proměnnými $\mathbf{x} = (x_0, x_1, \dots, x_{N_x})$ a množinami parametrů $\Theta = (\theta_1, \dots, \theta_{N_\theta})$. Tento faktor reprezentuje podmíněnou pravděpodobnost:

$$f_\beta(\mathbf{x}, \Theta) = p(x_0 | x_1, \dots, x_{N_x}; \Theta)$$

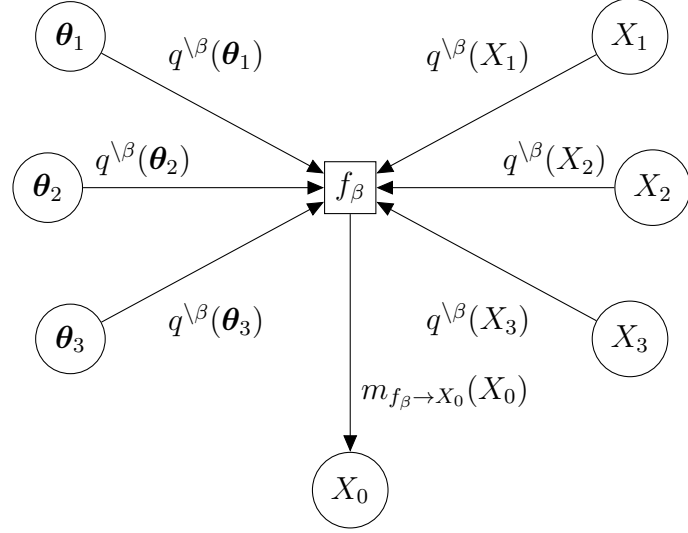
Rodičovské proměnné x_1, \dots, x_{N_x} označujeme jako \mathbf{x}' . Vektor \mathbf{x}' určuje, která množina parametrů bude použita. Protože množiny parametrů jsou číslovány $1, \dots, N_\theta$ a rodičovské proměnné $1, \dots, N_x$, musí být pro vybrání správné množiny parametrů použito mapování $\rho(\mathbf{x}')$. Faktor pak může být zapsán zkráceně:

$$f_\beta(\mathbf{x}, \Theta) = p(x_0 | x_1, \dots, x_{N_x}; \Theta) = \theta_{\rho(\mathbf{x}'), x_0}$$

Příklad faktoru f_β je na obrázku 3.1, pro ilustraci pouze s třemi parametry a čtyřmi proměnnými. Správně by měl faktor mít tolik parametrů, kolik je možných přiřazení pro rodičovské proměnné.

3.2 Výpočet marginálních pravděpodobností

Pro výpočet sdružené pravděpodobnosti používáme plně faktorizovanou distribuci. Pro každou proměnnou anebo množinu parametrů je její marginální pravděpodobnost rovna součinu zpráv přicházejících z faktorů, které jsou s danou proměnnou nebo množinou parametrů propojeny. Pro daný faktor f_β je neúplná distribuce $q^{\setminus\beta}(x_i)$, popř. $q^{\setminus\beta}(\theta_i)$ rovna součinu zpráv ze všech ostatních faktorů do x_i , popř. θ_i . Aproximovaná marginální pravděpodobnost proměnné je pak



Obrázek 3.1: Vybraný faktor f_β pro aktualizaci při učení parametrů.

součinem neúplné distribuce a zprávy z faktoru:

$$q(x_i) = q^{\backslash\beta}(x_i)m_{f_\beta \rightarrow x_i}(x_i), \quad q(\theta_i) = q^{\backslash\beta}(\theta_i)m_{f_\beta \rightarrow \theta}(\theta_i)$$

Tyto neúplné distribuce jsou právě zprávy z proměnné, popř. množiny parametrů do faktoru.

$$m_{x_i \rightarrow f_\beta} = q^{\backslash\beta}(x_i), \quad m_{\theta_i \rightarrow f_\beta} = q^{\backslash\beta}(\theta_i)$$

3.2.1 Marginální pravděpodobnost proměnných

Pokud chceme aktualizovat hodnotu naší aproximace marginální pravděpodobnosti, je třeba minimalizovat její vzdálenost od skutečné marginální pravděpodobnosti, ve tvaru

$$p^*(\tilde{x}_j) = \sum_{\mathbf{x} \setminus x_j} \int_{\Theta} \prod_i q^{\backslash\beta}(x_i) \prod_l q^{\backslash\beta}(\theta_l) f_\beta(\mathbf{x}; \Theta) \quad (3.1)$$

$$= \sum_{\mathbf{x} \setminus x_j} \prod_i q^{\backslash\beta}(x_i) \int_{\theta_{\rho(\mathbf{x}')}} q^{\backslash\beta}(\theta_{\rho(\mathbf{x}')}) \theta_{\rho(\mathbf{x}'), x_0} \quad (3.2)$$

$$= \sum_{\mathbf{x} \setminus x_j} \prod_i q^{\backslash\beta}(x_i) \mathbb{E}_{q^{\backslash\beta}}(\theta_{\rho(\mathbf{x}'), x_0}) \quad (3.3)$$

Rovnost (3.1) vychází z definice výpočtu marginální pravděpodobnosti ze sdružené pravděpodobnosti. V (3.2) byla použita definice faktoru, z integrálu byly vytaženy členy, které neobsahují Θ a nakonec bylo využito toho, že pro množiny parametrů, které nejsou spojeny s faktorem f_β , je jejich neúplná distribuce rovná

marginální distribuci a tedy $\int_{\theta_i} q(\theta_i) = 1$. V (3.3) byla použita definice očekávané hodnoty.

Marginální pravděpodobnost proměnné x_i tedy je

$$p^*(x_i) = \sum_{\mathbf{x} \setminus x_j} \prod_i q^{\setminus \beta}(x_i) \mathbb{E}_{q^{\setminus \beta}}(\theta_{\rho(\mathbf{x}'), x_0}) \quad (3.4)$$

Tady docházíme k výsledku, který je velmi podobný výpočtu marginální pravděpodobnosti v Loopy Belief Propagation algoritmu.

Zprávu z faktoru f_β do vrcholu x_j pak získáme vydělením zprávy z x_j z marginální pravděpodobnosti.

$$m_{f_\beta \rightarrow x_j}(x_j) = \sum_{\mathbf{x} \setminus x_j} \prod_{i \neq j} q^{\setminus \beta}(x_i) \mathbb{E}_{q^{\setminus \beta}}(\theta_{\rho(\mathbf{x}'), x_0}) \quad (3.5)$$

3.2.2 Marginální pravděpodobnost parametrů

Pro množiny parametrů se jejich marginální pravděpodobnost spočítá podobně jako pro proměnné.

$$p^*(\tilde{\theta}_j) = \sum_{\mathbf{x}} \int_{\Theta: \theta_j = \tilde{\theta}_j} \prod_i q^{\setminus \beta}(x_i) \prod_l q^{\setminus \beta}(\theta_l) f_\beta(\mathbf{x}; \Theta) \quad (3.6)$$

$$= \sum_{l \neq j} \sum_{\mathbf{x}: \rho(\mathbf{x}') = l} \prod_i q^{\setminus \beta}(x_i) \int_{\Theta: \theta_j = \tilde{\theta}_j} \prod_k q^{\setminus \beta}(\theta_k) \theta_{l, x_0} + \quad (3.7)$$

$$+ \sum_{\mathbf{x}: \rho(\mathbf{x}') = j} \prod_i q^{\setminus \beta}(x_i) \int_{\Theta: \theta_j = \tilde{\theta}_j} \prod_k q^{\setminus \beta}(\theta_k) \tilde{\theta}_{j, x_0}$$

$$= \left[\sum_{l \neq j} \sum_{\mathbf{x}: \rho(\mathbf{x}') = l} \prod_i q^{\setminus \beta}(x_i) \mathbb{E}_{q^{\setminus \beta}(\theta_l)}(\theta_{l, x_0}) \right] q^{\setminus \beta}(\tilde{\theta}_j) + \quad (3.8)$$

$$+ \sum_{\mathbf{x}: \rho(\mathbf{x}') = j} \prod_i q^{\setminus \beta}(x_i) \tilde{\theta}_{j, x_0} q^{\setminus \beta}(\tilde{\theta}_j)$$

$$= w_0 q^{\setminus \beta}(\tilde{\theta}_j) + \sum_k w_k \tilde{\theta}_{j, k} q^{\setminus \beta}(\tilde{\theta}_j), \quad (3.9)$$

kde

$$w_0 = \sum_{l \neq j} \sum_{\mathbf{x}: \rho(\mathbf{x}') = l} \prod_i q^{\setminus \beta}(x_i) \mathbb{E}_{q^{\setminus \beta}(\theta_l)}(\theta_{l, x_0}) \quad (3.10)$$

$$w_k = \sum_{\mathbf{x}: \rho(\mathbf{x}') = j, x_0 = k} \prod_i q^{\setminus \beta}(x_i) \quad (3.11)$$

Opět vycházíme z výpočtu marginální pravděpodobnosti podle definice. V

rovnici (3.7) jsme rozdělili sumu přes \mathbf{x} na ty, pro které se ve faktoru použije množina parametrů $\tilde{\boldsymbol{\theta}}_j$ a na ty ostatní. Také jsme z integrálu vytknuli součin neúplných distribucí pro proměnné. V dalším kroku (3.8) jsme opět použili toho, že integrál přes $\boldsymbol{\Theta}$ je ve skutečnosti několik integrálů přes jednotlivé množiny parametrů. A tedy je můžeme vložit mezi jednotlivé členy produktu neúplných distribucí pro množiny parametrů. Ve výsledku získáme $q^{\setminus\beta}(\tilde{\boldsymbol{\theta}}_j) \int_{\boldsymbol{\theta}_l} q^{\setminus\beta}(\boldsymbol{\theta}_l) \theta_{l,x_0}$ a pak zbylé členy, které zmizí.

Docházíme k vyjádření skutečné marginální pravděpodobnosti, ve které není třeba integrovat přes všechny množiny parametrů, ale stačí jen očekávaná hodnota těchto parametrů.

3.3 Aproximace marginálních pravděpodobností

Při výpočtu aproximující distribuce $q(\boldsymbol{\theta}_j)$ se může stát, že skutečná marginální distribuce je v komplikovaném tvaru a pak je aproximace složitá na výpočet a může se odchylovat od skutečné distribuce. Například se jedná o směs distribucí. Ideální případ je takový, kdy aproximace je stejného typu jako skutečná distribuce.

Pro zjednodušení výpočtů, jak se dále ukáže, zvolíme zprávy z faktoru do množiny parametrů, $m_{f \rightarrow \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i)$, ve tvaru Dirichletovské distribuce s parametry $\boldsymbol{\alpha}_{f \rightarrow \boldsymbol{\theta}_i}$:

$$m_{f \rightarrow \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i) = \text{Dir}(\boldsymbol{\theta}_i; \boldsymbol{\alpha}_{f \rightarrow \boldsymbol{\theta}_i}) = \frac{\Gamma(\sum_j \boldsymbol{\alpha}_{f \rightarrow \boldsymbol{\theta}_i, j})}{\prod_j \Gamma(\boldsymbol{\alpha}_{f \rightarrow \boldsymbol{\theta}_i, j})} \prod_j \theta_{i,j}^{\boldsymbol{\alpha}_{f \rightarrow \boldsymbol{\theta}_i, j} - 1} \quad (3.12)$$

kde Γ je Gamma funkce (zobecnění faktoriálu):

$$\Gamma(z) = \int_0^\infty t^{z-1} \exp(-t) dt \quad (3.13)$$

Dirichletovská distribuce byla zvolena, protože má důležité vlastnosti pro součin, které budou využity dále pro výpočet neúplné distribuce a celkové aproximace. Pokud označíme aproximované faktory indexem β a každý bude mít vlastní parametry $\boldsymbol{\alpha}_{f\beta \rightarrow \boldsymbol{\theta}_i}$, tak výsledná aproximace bude tvaru:

$$q(\boldsymbol{\theta}_i) \propto \prod_{\beta} m_{f\beta \rightarrow \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i) \propto \prod_{\beta} \prod_j \theta_{i,j}^{\boldsymbol{\alpha}_{f\beta \rightarrow \boldsymbol{\theta}_i, j} - 1} \quad (3.14)$$

$$\propto \text{Dir}(\boldsymbol{\theta}_i; \sum_{\beta} \boldsymbol{\alpha}_{f\beta \rightarrow \boldsymbol{\theta}_i} - (|\beta| - 1) \mathbf{1}) \quad (3.15)$$

$$= \text{Dir}(\boldsymbol{\theta}_i; \boldsymbol{\alpha}_i) \quad (3.16)$$

kde $\alpha_i = \sum_{\beta} \alpha_{f_{\beta} \rightarrow \theta_i} - (|\beta| - 1)\mathbf{1}$.

Při aktualizaci faktoru $\tilde{\beta}$ tedy neúplná distribuce bude:

$$q^{\setminus \tilde{\beta}}(\theta_i) \propto \prod_{\beta \neq \tilde{\beta}} m_{f_{\beta} \rightarrow \theta_i}(\theta_i) \quad (3.17)$$

$$\propto \text{Dir}(\theta_i; \alpha_i - \alpha_{f_{\tilde{\beta}} \rightarrow \theta_i} + \mathbf{1}) \quad (3.18)$$

Naším cílem je nalézt parametry α^* aproximované marginální pravděpodobnosti (3.16), které minimalizují vzdálenost od skutečné marginální pravděpodobnosti (3.9).

Jednotlivé faktory musíme vždy aproximovat s ohledem na všechny ostatní. U našeho grafického modelu využijeme nezávislostí a tak nám stačí pouze pracovat se zprávami ze sousedních vrcholů. Všechna informace se v modelu šíří přes zprávy mezi vrcholy a proto je třeba několika iterací pro přesun informace z jedné části modelu do druhé.

Pro měření vzdálenosti mezi dvěma pravděpodobnostními rozloženími se používá Kullback-Leiblerova divergence:

$$KL(p||q) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx \quad (3.19)$$

Pro nalezení minima použijeme algoritmus srovnání momentů a budeme minimalizovat $KL(p^*||q)$.

Pokud se podíváme na skutečnou marginální pravděpodobnost $p^*(\theta_i)$, zjistíme, že můžeme některé její členy upravit. Využijeme také vlastnosti gamma funkce $\Gamma(x) = (x-1)\Gamma(x-1)$.

$$w_j \theta_j \text{Dir}(\theta; \alpha) \propto w_j \theta_j \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \prod_i \theta_i^{\alpha_i-1} \quad (3.20)$$

$$\propto w_j \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \theta_j^{\alpha_j} \prod_{i \neq j} \theta_i^{\alpha_i-1} \quad (3.21)$$

$$\propto w_j \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \frac{\Gamma(\alpha_j + 1)}{\Gamma(1 + \sum_i \alpha_i)} \prod_{i \neq j} \Gamma(\alpha_i) \text{Dir}(\theta; \alpha + \delta_j) \quad (3.22)$$

$$\propto w_j \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \frac{\alpha_j \Gamma(\alpha_j)}{(\sum_i \alpha_i) \Gamma(\sum_i \alpha_i)} \prod_{i \neq j} \Gamma(\alpha_i) \text{Dir}(\theta; \alpha + \delta_j) \quad (3.23)$$

$$\propto w_j \frac{\alpha_j}{\sum_i \alpha_i} \text{Dir}(\theta; \alpha + \delta_j) \quad (3.24)$$

$$(3.25)$$

Díky této úpravě lze p^* vyjádřit jako směs Dirichletovských rozdělení.

$$p^*(\boldsymbol{\theta}) = w_0^* \text{Dir}(\boldsymbol{\theta}; \boldsymbol{\alpha}) + \sum_j w_j^* \text{Dir}(\boldsymbol{\theta}; \boldsymbol{\alpha} + \boldsymbol{\delta}_j) \quad (3.26)$$

kde

$$w_0^* \propto w_0, \quad w_j^* \propto w_j \frac{\alpha_j}{\sum_i \alpha_i}, \quad \sum_{i=0}^k w_i^* = 1 \quad (3.27)$$

Jak bylo ukázáno v sekci 2.3.1, pro minimalizaci KL divergence mezi dvěma rozděleními z exponenciální rodiny stačí, pokud se budou rovnat jejich postačující statistiky. Dokážeme jednoduše spočítat první dva momenty Dirichletovského rozdělení a proto použijeme aproximaci a budeme počítat pouze s nimi a zbylé momenty zanedbáme. Je tedy třeba nalézt střední hodnotu a rozptyl proměnných z $p^*(\boldsymbol{\theta})$.

$$\mathbb{E}_{p^*}[\boldsymbol{\theta}] = \int \boldsymbol{\theta} p^*(\boldsymbol{\theta}) \, d\boldsymbol{\theta} \quad (3.28)$$

$$= \int \boldsymbol{\theta} (w_0^* \text{Dir}(\boldsymbol{\theta}; \boldsymbol{\alpha}) + \sum_j w_j^* \text{Dir}(\boldsymbol{\theta}; \boldsymbol{\alpha} + \boldsymbol{\delta}_j)) \, d\boldsymbol{\theta} \quad (3.29)$$

$$= w_0^* \int \boldsymbol{\theta} \text{Dir}(\boldsymbol{\theta}; \boldsymbol{\alpha}) \, d\boldsymbol{\theta} + \sum_j w_j^* \int \boldsymbol{\theta} \text{Dir}(\boldsymbol{\theta}; \boldsymbol{\alpha} + \boldsymbol{\delta}_j) \, d\boldsymbol{\theta} \quad (3.30)$$

$$= w_0^* \mathbb{E}_{\text{Dir}(\boldsymbol{\alpha})}[\boldsymbol{\theta}] + \sum_j w_j^* \mathbb{E}_{\text{Dir}(\boldsymbol{\alpha} + \boldsymbol{\delta}_j)}[\boldsymbol{\theta}] \quad (3.31)$$

Střední hodnotu proměnných $\boldsymbol{\theta}$ podle rozdělení p^* lze tedy spočítat jako vážený součet středních hodnot $\boldsymbol{\theta}$ podle jednotlivých Dirichletovských distribucí, z kterých se p^* skládá.

První moment tedy máme spočítaný, pro výpočet rozptylu můžeme využít definici:

$$\text{Var}[\boldsymbol{\theta}] = \mathbb{E}[\boldsymbol{\theta}^2] - \mathbb{E}[\boldsymbol{\theta}]^2 \quad (3.32)$$

Chybí nám tedy ještě výpočet střední hodnoty druhé mocniny proměnné $\boldsymbol{\theta}$ podle p^* . Můžeme ji vyjádřit z definice střední hodnoty.

$$\mathbb{E}_{p^*}[\boldsymbol{\theta}^2] = \int \boldsymbol{\theta}^2 p^*(\boldsymbol{\theta}) d\boldsymbol{\theta} \quad (3.33)$$

$$= w_0^* \int \boldsymbol{\theta}^2 Dir(\boldsymbol{\theta}; \boldsymbol{\alpha}) d\boldsymbol{\theta} + \sum_j w_j^* \int \boldsymbol{\theta}^2 Dir(\boldsymbol{\theta}; \boldsymbol{\alpha} + \boldsymbol{\delta}_j) d\boldsymbol{\theta} \quad (3.34)$$

$$= w_0^* \mathbb{E}_{Dir(\boldsymbol{\alpha})}[\boldsymbol{\theta}^2] + \sum_j w_j^* \mathbb{E}_{Dir(\boldsymbol{\alpha} + \boldsymbol{\delta}_j)}[\boldsymbol{\theta}^2] \quad (3.35)$$

Opět získáváme vážený součet středních hodnot podle Dirichletovských rozdělení. Střední hodnotu druhé mocniny proměnné podle Dirichletovského rozdělení lze opět jednoduše odvodit z definice.

$$\mathbb{E}_{Dir(\boldsymbol{\alpha})}[x_i^2] = \int x_i^2 Dir(\mathbf{x}; \boldsymbol{\alpha}) d\mathbf{x} \quad (3.36)$$

$$= \int x_i^2 \frac{\Gamma(\alpha_0)}{\prod_{j=1}^N \Gamma(\alpha_j)} \prod_{j=1}^N x_j^{\alpha_j-1} d\mathbf{x} \quad (3.37)$$

Nyní jsme v podobné situaci jako v (3.20). Budeme postupovat stejně, vyjádříme novou Dirichletovskou distribuci.

$$\mathbb{E}_{Dir(\boldsymbol{\alpha})}[x_i^2] = \int \frac{\Gamma(\alpha_0 + 2) \alpha_i (\alpha_i + 1)}{\alpha_0 (\alpha_0 + 1) \Gamma(\alpha_i + 2) \prod_{j \neq i} \Gamma(\alpha_j)} x_i^{\alpha_i+1} \prod_{j \neq i} x_j^{\alpha_j-1} d\mathbf{x} \quad (3.38)$$

$$= \frac{\alpha_i (\alpha_i + 1)}{\alpha_0 (\alpha_0 + 1)} \int \frac{\Gamma(\beta_0)}{\prod_i \Gamma(\beta_i)} \prod_i x_i^{\beta_i-1} d\mathbf{x} \quad (3.39)$$

$$= \frac{\alpha_i (\alpha_i + 1)}{\alpha_0 (\alpha_0 + 1)} \int Dir(\mathbf{x}; \boldsymbol{\beta}) d\mathbf{x} \quad (3.40)$$

$$= \frac{\alpha_i (\alpha_i + 1)}{\alpha_0 (\alpha_0 + 1)} \quad (3.41)$$

Vyjádřili jsme $\Gamma(\alpha_0)$ a $\Gamma(\alpha_i)$ s pomocí $\Gamma(\alpha_0 + 2)$ a $\Gamma(\alpha_i + 2)$

$$\Gamma(\alpha_0) = \frac{\Gamma(\alpha_0 + 2)}{\alpha_0 (\alpha_0 + 1)} \quad \Gamma(\alpha_i) = \frac{\Gamma(\alpha_i + 2)}{\alpha_i (\alpha_i + 1)} \quad (3.42)$$

Následně jsme vytvořili nové parametry $\boldsymbol{\beta}$:

$$\beta_i = \alpha_i + 2 \quad \beta_{j \neq i} = \alpha_j \quad \beta_0 = \sum_i \beta_i \quad (3.43)$$

Nyní tedy dokážeme spočítat $\mathbb{E}_{p^*}[\boldsymbol{\theta}]$ a $\mathbb{E}_{p^*}[\boldsymbol{\theta}^2]$. Parametry aproximovaného

rozdělení nalezneme následovně

$$\frac{\mathbb{E}[X_1] - \mathbb{E}[X_1^2]}{\mathbb{E}[X_1^2] - \mathbb{E}[X_1]^2} = \frac{\frac{\alpha_1}{\alpha_0} - \frac{\alpha_1(\alpha_1+1)}{\alpha_0(\alpha_0+1)}}{\frac{\alpha_1(\alpha_1+1)}{\alpha_0(\alpha_0+1)} - \frac{\alpha_1^2}{\alpha_0^2}} \quad (3.44)$$

$$= \frac{\frac{\alpha_1(\alpha_0+1) - \alpha_1(\alpha_1+1)}{\alpha_0(\alpha_0+1)}}{\frac{\alpha_0\alpha_1(\alpha_1+1) - \alpha_1^2(\alpha_0+1)}{\alpha_0^2(\alpha_0+1)}} \quad (3.45)$$

$$= \frac{\alpha_0\alpha_1(\alpha_0 - \alpha_1)}{\alpha_1(\alpha_0\alpha_1 + \alpha_0 - \alpha_0\alpha_1 - \alpha_1)} \quad (3.46)$$

$$= \alpha_0 \quad (3.47)$$

$$\alpha_i = \mathbb{E}[X_i] \alpha_0 \quad (3.48)$$

Z rovnice (3.44) vypočítáme sumu všech parametrů α_0 . Protože střední hodnota proměnné z Dirichletovského rozdělení je právě $\frac{\alpha_i}{\alpha_0}$, tak jednotlivé parametry získáme z rovnice (3.48).

3.4 Algoritmus

V algoritmu 4 je předveden algoritmus Expectation Propagation upravený pro potřeby učení parametrů.

Algoritmus 4 Expectation Propagation pro učení parametrů

Parametry zpráv z faktoru β do množiny parametrů θ_i označíme $\alpha_{f_\beta \rightarrow \theta_i}$.

Parametry zpráv z množiny parametrů θ_i do faktoru β označíme $\alpha_{\theta_i \rightarrow f_\beta}$.

Parametry marginální distribuce množiny parametrů θ_i označíme α_i .

init

Nastav zprávy mezi faktory a proměnnými na 1.

Nastav parametry $\alpha_{f_\beta \rightarrow \theta_i}$ na 1.

Nastav parametry α_i na apriorní hodnotu.

end init

repeat

Vyber faktor $f_{\tilde{\beta}}$, který se bude aktualizovat.

Spočítej všechny zprávy z parametrů:

for každý parametr θ_i spojený s faktorem $f_{\tilde{\beta}}$ **do**

Parametry zprávy z θ_i do $f_{\tilde{\beta}}$: $\alpha_{\theta_i \rightarrow f_{\tilde{\beta}}} = \alpha_i - \alpha_{f_{\tilde{\beta}} \rightarrow \theta_i} + 1$.

end for

Aktualizuj zprávy z faktoru do proměnných:

for každou proměnnou X_i , spojenou s faktorem $f_{\tilde{\beta}}$ **do**

Zpráva z $f_{\tilde{\beta}}$ do X_i podle (3.5):

$$m_{f_{\tilde{\beta}} \rightarrow x_j} = \sum_{\mathbf{x}: x_j = \tilde{x}_j} \mathbb{E}_{q \setminus (\theta_{\rho(\mathbf{x}'), x_0})} \prod_{i \neq j} q \setminus (x_i)$$

end for

Aktualizuj marginální pravděpodobnost parametrů:

for každý parametr θ_i spojený s faktorem $f_{\tilde{\beta}}$ **do**

Spočítej parametry α_i^* pro Dirichletovské rozdělení, které nejlépe aproximuje cílovou marginální distribuci (3.26). Metoda popsána v předchozí sekci.

Parametry zprávy z $f_{\tilde{\beta}}$ do θ_i :

$$\alpha_{f_{\tilde{\beta}} \rightarrow \theta_i} = \alpha_i^* - \alpha_{\theta_i \rightarrow f_{\tilde{\beta}}} + 1$$

Aktualizuj parametry marginální distribuce $q(\theta_i)$

$$\alpha_i = \alpha_i^* = \alpha_{f_{\tilde{\beta}} \rightarrow \theta_i} + \alpha_{\theta_i \rightarrow f_{\tilde{\beta}}}$$

end for

for každou proměnnou X_i , spojenou s faktorem $f_{\tilde{\beta}}$ **do**

Aktualizuj zprávy z proměnných do faktoru:

$$q \setminus (x_i) = \prod_{\beta \neq \tilde{\beta}} m_{f_\beta \rightarrow x_i}(x_i)$$

end for

until konvergence

4. Implementace

V předchozích kapitolách byly popsány teoretické základy nutné pro implementaci inference v bayesovských sítích. V této kapitole bude popsána vytvořená knihovna pro dialogové systémy. Celá knihovna se skládá z několika vrstev, každá z nich staví na předchozí a poskytuje větší úroveň abstrakce. Nejnižší vrstva implementuje efektivní počítání s faktory. Nad ní stojí vrstva reprezentující jednotlivé vrcholy ve faktor grafu. Tato vrstva také obsahuje funkcionalitu pro počítání a posílání zpráv. Nejvyšší vrstva se zaměřuje na samotnou inferenci v grafickém modelu.

4.1 Diskrétní faktor

Faktor je základním stavebním kamenem. Operace s faktory musí být efektivní, pro výpočet jedné zprávy je potřeba několik násobení faktorů, následně je třeba je marginalizovat, atd. Faktory je třeba úsporně reprezentovat, každá zpráva, každé pravděpodobnostní rozdělení je samo o sobě faktor. Faktory jsou reprezentovány třídou **Factor** v modulu `alex/infer/factor.py`, která podporuje všechny základní matematické operace a také řadu specifických operací jako je marginalizace, normalizace, nastavení faktoru na pozorovanou hodnotu, vybrání nejpravděpodobnějšího přiřazení, atd.

4.1.1 Reprezentace faktorů

Každý diskretní faktor má seznam diskretních proměnných, které tvoří jeho doménu. Každá z těchto proměnných může mít jinou kardinalitu, některé proměnné jsou binární, jiné mají mnohem více hodnot. Faktor je tedy ve své podstatě multidimenzionální tabulka. Pro implementaci je tato tabulka zploštěná do jednoduchého pole.

Knihovna je napsaná v Pythonu a pro matematické operace používá knihovnu Numpy [18]. Pole pak využívají implementaci z knihovny Numpy, díky které jsou matematické operace napsané v rychlejším jazyku (C, Fortran) než je Python a jsou navíc vektorizované.

Jak je tedy možné reprezentovat multidimenzionální tabulku jednodimenzionálním polem? Proměnné jsou seřazené a pro zjednodušení můžeme předpokládat, že hodnoty proměnných jsou čísla z $\{0, \dots, n - 1\}$, kde n je kardinalita proměnné. Každá hodnota v poli je pak hodnotou faktoru pro nějaké přiřazení hodnot proměnným, např. $(0, 1, 0)$. Jednotlivé přiřazení jsou v poli seřazeny lexikograficky. Pro každou proměnnou si pamatujeme její kardinalitu a také tzv. krok. Krok určuje pro každou proměnnou o kolik hodnot v tabulce se musíme posunout,

X	Y	Hodnota
0	0	0.2
0	1	0.3
1	0	0.1
1	1	0.4

Tabulka 4.1: Příklad faktoru s dvěma proměnnými X a Y .

abychom se dostali na další hodnotu této proměnné se zachováním hodnot všech následujících proměnných.

Příklad faktoru je v tabulce 4.1. Doménu faktoru tvoří dvě binární proměnné X a Y , jejich kardinalita je tedy 2. Pro proměnnou X je krok 2, pro proměnnou Y je krok 1.

4.1.2 Operace s faktory

Implementace diskrétního faktoru obsahuje všechny základní matematické operace, ale také speciální operace, které jsou využity specificky pro pravděpodobnostní distribuce, např. marginalizace anebo normalizace. Operace jako násobení a marginalizace jsou používány při každém výpočtu zprávy a tedy je třeba je napsat tak, aby fungovaly co nejefektivněji.

Matematické operace s faktory musí fungovat ve třech různých situacích, příklady uvedeme na násobení.

1. Násobení faktoru s faktorem, oba se stejnou doménou,
2. násobení dvou faktorů, které sdílí jen některé proměnné,
3. násobení faktoru konstantou.

Násobení faktoru konstantou je triviální, každá položka faktoru bude vynásobena konstantou. Tato operace může být jednoduše vektorizována.

Při násobení dvou faktorů se stejnou doménou je třeba pronásobit prvky se stejným přiřazením proměnných. Což znamená pronásobit hodnoty na stejných místech v poli. Opět se tedy jedná o operaci, která je jednoduše vektorizovatelná.

Operace s různými doménami

Poslední možnost je, že se snažíme provést matematickou operaci s dvěma faktory, které ovšem nemají stejnou doménu. Pak musí výsledkem být nový faktor, jehož doména je sjednocením domén vstupních faktorů. Jednotlivé prvky nového

f_1			\times	f_2			$=$	f_r			
X	Y	$Hodnota$		Y	Z	$Hodnota$		X	Y	Z	$Hodnota$
0	0	0.2		0	0	0.2		0	0	0	0.04
0	1	0.3		0	1	0.2		0	0	1	0.04
1	0	0.1		1	0	0.2		0	1	0	0.06
1	1	0.4		1	1	0.4		0	1	1	0.12
								1	0	0	0.02
								1	0	1	0.02
								1	1	0	0.08
								1	1	1	0.16

Tabulka 4.2: Příklad násobení dvou faktorů, které sdílí proměnnou.

faktoru jsou pak výsledkem aplikace operace na prvky ze vstupních faktorů, které sdílí ohodnocení společných proměnných. Příklad s násobením dvou faktorů, které sdílí proměnnou je v tabulce 4.2.

Výsledek násobení faktorů f_1 a f_2 je ve faktoru f_r . Faktory sdílí pouze proměnnou Y , takže je potřeba pronásobit všechny přiřazení z f_1 se všemi přiřazeními z f_2 , které mají stejnou hodnotu Y . Příkladem je například přiřazení $(0, 1)$ s hodnotou 0.3 vynásobené s přiřazením $(1, 1)$ s hodnotou 0.4. Výsledek je uložen ve faktoru f_r s přiřazením $(0, 1, 1)$ a správnou hodnotou $0.3 \cdot 0.4 = 0.12$.

4.1.3 Algoritmus pro operace s různými doménami

Předvedli jsme možné případy operací s faktory a ukázali, že dva ze tří jsou triviální na implementaci. Nyní představíme efektivní implementaci třetí možnosti, tedy aplikace operace na dva faktory s různými doménami (algoritmus 5).

Ze vstupních faktorů vytvoříme prázdný faktor pro výsledek. Jeho doména je sjednocením domén vstupních faktorů. Kardinalita proměnných zůstává stejná. Krok jednotlivých proměnných je třeba přepočítat. Spočítáme jej jako součin kardinalit proměnných, které následují po té aktuální. Velikost pole pro všechny hodnoty je rovna součin všech kardinalit.

Následně přistoupíme k vyplňování tabulky. Pro oba vstupní faktory si budeme udržovat index na pozici s ohodnocením proměnných, které odpovídá aktuálně vyplňovanému ohodnocení ve výsledném faktoru. Po provedení operace tyto indexy aktualizujeme.

Pokud reprezentujeme ohodnocení proměnných jako číslo (kde každá cifra může mít jinou kardinalitu), pak se přesuneme k dalšímu ohodnocení v řadě tak, že zvýšíme nejméně signifikantní cifru (proměnnou) o jedna. Může se stát, že jsme dosáhli kardinality dané proměnné a pak se musíme vrátit na ohodnocení 0 a aplikovat přesun na vyšší cifru. Opakovanou aplikací přesunu můžeme upravit

všechny proměnné, příkladem je přechod z přiřazení $(0, 1, 1)$ na $(1, 0, 0)$, všechny proměnné binární. Při každé úpravě proměnné také aktualizujeme indexy do vstupních faktorů.

Při přesunu na další hodnotu proměnné stačí k indexu do faktoru přičíst krok upravené proměnné v tomto faktoru. Pokud je třeba se vrátit u proměnné v na ohodnocení 0, pak od indexu pro vstupní faktor odečteme $(c_v - 1) \cdot s_v$, kde c_v je kardinalita proměnné v a s_v je krok proměnné v . Pro proměnnou, která není obsažena v doméně faktoru, je její krok roven 0. Celá metoda pro aplikaci matematické operace je v algoritmu 5.

Algoritmus 5 Aplikace operace na faktory s různými doménami

function APPLY-OP(f_1, f_2, op)

f_1, f_2 – vstupní faktory

op – operace

$f_r \leftarrow$ nový faktor pro výsledek operace na f_1 a f_2 s prázdným polem

index[f_1] \leftarrow 0

index[f_2] \leftarrow 0

přiřazení[v] \leftarrow 0 pro každou proměnnou $v \in$ proměnné[f_r]

for $i \in \{0, \dots, \text{LENGTH}(\text{pole}[f_r]) - 1\}$ **do**

 pole[f_r][i] = op(pole[f_1][index[f_1]], pole[f_2][index[f_2]])

for $v \in \text{REVERSED}(\text{proměnné}[f_r])$ **do**

 přiřazení[v] += 1

if přiřazení[v] = kardinalita[v] **then**

 přiřazení[v] \leftarrow 0

 index[f_1] -= (kardinalita[v] - 1) · krok[f_1][v]

 index[f_2] -= (kardinalita[v] - 1) · krok[f_2][v]

else

 index[f_1] += krok[f_1][v]

 index[f_2] += krok[f_2][v]

break

end if

end for

end for

return f_r

end function

Tento algoritmus pracuje přímo nad polem a jeho výhodou je, že nepotřebuje ke svému výpočtu znát jednotlivé hodnoty proměnných. Na rozdíl třeba od algoritmu, který by generoval všechny možné kombinace přiřazení a pak pomocí nich přistupoval k odpovídajícím hodnotám ve faktorech, zde pracujeme pouze s indexy do polí a používáme jednoduché matematické operace.

4.1.4 Marginalizace proměnných

Další důležitou operací často používanou při počítání s faktory je marginalizace. Na vstupu je faktor a podmnožina proměnných z domény faktoru, které mají zůstat pro vysčítání zbytku. Algoritmus je podobný aplikaci matematické operace z předchozí sekce. V tomto případě procházíme původní faktor a každou hodnotu přičteme na správnou pozici v novém faktoru. Pro aktualizaci indexu v tomto případě musíme projít všechny proměnné, které mají zůstat a u každé zkontrolovat, zda-li se v následujícím kroku změní. Postup marginalizace je zapsán v algoritmus 6.

Algoritmus 6 Marginalizace faktoru

```
function MARGINALIZE( $f$ ,  $vars$ )  
   $f$  – vstupní faktor  
   $vars$  – seznam proměnných, které mají zůstat  
  
   $f_r \leftarrow$  nový faktor, obsahující pouze proměnné z  $vars$   
  přiřazení[ $v$ ]  $\leftarrow 0$  pro každou proměnnou  $v \in vars$   
   $index \leftarrow 0$  ▷ Index do nového faktoru  
  
  for  $i \in \{0, \dots, \text{LENGTH}(\text{pole}[f]) - 1\}$  do  
     $\text{pole}[f_r][index] += \text{pole}[f][i]$   
  
    for  $v \in vars$  do  
      if  $(i + 1) \bmod \text{krok}[f][v] = 0$  then  
        přiřazení[ $v$ ]  $+= 1$   
         $index += \text{krok}[f_r][v]$   
      end if  
  
      if přiřazení[ $v$ ] = kardinalita[ $v$ ] then  
        přiřazení[ $v$ ]  $\leftarrow 0$   
         $index -= \text{kardinalita}[v] \cdot \text{krok}[f_r][v]$   
      end if  
    end for  
  end for  
  return  $f_r$   
end function
```

4.2 Vrcholy faktor grafu

Předchozí sekce hovořila o práci s faktory, tato sekce se bude zabývat implementací jednotlivých vrcholů ve faktor grafu. Vrcholy se dělí na vrcholy pro proměnné a vrcholy pro faktory. Zde může být názvosloví matoucí, protože implementace faktorů z předchozí sekce se používá pro oboje. Vrchol pro proměnnou reprezentuje

marginální distribuci proměnné. Vrcholy pro faktory reprezentují pouze samotné faktory, z kterých se skládá sdružená distribuce. Vrcholy slouží k vytvoření reprezentace grafického modelu a obsahují metody pro výpočet a posílání zpráv. Implementace vrcholů je v modulu `alex/infer/node.py`.

4.2.1 Rozhraní vrcholů

Základní funkcionalita vrcholů je popsána v abstraktní třídě **Node**. Vlastností všech vrcholů je jejich sdružování do sítě. K tomu slouží metoda **connect**, kterou musí obsahovat každá implementace vrcholu a ta informuje oba vrcholy, že spolu sousedí. Většina vrcholů si pamatuje své sousedy pro počítání zpráv. Při připojování proměnných k faktorům je také možné označit proměnnou za rodiče daného faktoru. Tato informace je důležitá při normalizaci a učení parametrů.

Dále třída obsahuje metody sloužící pro posílání zpráv. Nejdůležitější jsou metody **message_to** a **message_from**. Vrchol spočítá v metodě **message_to** zprávu pro svého souseda. Má přístup ke všem ostatním vrcholům a tak pro něj není problém zprávu spočítat. Sousední vrchol zprávu přijme tak, že bude zavolána jeho metoda **message_from** a v ní mu bude zpráva předána. Díky tomu, že na posílání zpráv se podílí odesílatel i příjemce, můžeme kombinovat různé vrcholy, stačí pokud se dohodnou na stejném formátu zprávy. Dochází tak k oddělení odesílatele od příjemce.

Další metodou, kterou obsahuje každý vrchol, je metoda pro inicializaci zpráv **init_messages**. Pro stávající implementace vrcholů není třeba ji volat před posíláním zpráv, protože zprávy mezi sousedními vrcholy jsou inicializovány vždy při zavolání metody **connect** pro jejich propojení. Pokud ovšem budeme provádět více výpočtů nad jedním grafickým modelem, je třeba zprávy nastavit na původní hodnoty před dalším výpočtem, jinak by hodnota zpráv z minulého výpočtu ovlivnila ten následující.

Z optimalizačních důvodů se při odesílání zpráv nepočítá součin všech příchodů vždy znovu, ale je předpočítán a pouze se aktualizuje před odesláním zpráv. Pro aktualizaci slouží metoda **update**. Vzhledem k tomu, že při inferenci je každý faktor vybrán jen jednou a pak jsou z něj odeslány zprávy do všech ostatních vrcholů, stačí metodu **update** volat pro každý faktor v každé iteraci jen jednou. Pro odeslání zpráv do všech vrcholů slouží jako zkratka metoda **send_messages**.

Třída **DiscreteVariableNode** implementuje funkcionalitu vrcholů pro diskrétní proměnné. Implementace vrcholů pro faktory s diskrétními proměnnými je ve třídě **DiscreteFactorNode**.

4.2.2 Rozhraní vrcholů pro proměnné

Smyslem vrcholů pro proměnné je reprezentovat aposteriorní marginální pravděpodobnost proměnných. Mají navíc několik metod, které jsou potřeba pro výpočty. Nejprve je třeba mít možnost nastavit pozorované proměnné, k tomu slouží metoda **observed**. Tato metoda může být zavolána se slovníkem, kde klíčem jsou pozorovaná přiřazení a hodnotou je pravděpodobnost pozorování. Zbylá přiřazení dostanou nulovou hodnotu. S tímto pozorováním se pak počítá při posílání zpráv.

Po ukončení výpočtu je třeba zjistit, která hodnota nebo více hodnot patří mezi nejpravděpodobnější. Některé dialogové strategie totiž potřebují jen jednu nejpravděpodobnější hodnotu, ale jiné mohou počítat se seznamem nejpravděpodobnějších možností. K tomu slouží metoda **most_probable**. Vrátí seznam nejpravděpodobnějších přiřazení a jejich pravděpodobností.

4.3 Vrcholy pro Dirichletovské parametry

V kapitole 3 jsme ukázali jakým způsobem je možné pro diskrétní proměnné učit parametry. V modulu **alex/infer/node.py** jsou implementované vrcholy pro práci s dirichletovskými parametry. Jedná se o třídy **DirichletParameterNode** a **DirichletFactorNode**. V rozhraní se neliší od standardních implementací vrcholů.

Vrchol **DirichletParameterNode** musí být vždy spojen pouze s vrcholem **DirichletFactorNode**, který už ale může být dále propojen s vrcholy pro diskrétní proměnné. Díky tomu, že tyto vrcholy o své existenci navzájem ví, tak je možné použít standardní rozhraní pro posílání zpráv. Při připojení vrcholu pro parametr k vrcholu pro faktor tento pozná, že se k němu připojil speciální vrchol a bude se tak k němu chovat. Zprávy od něj bude interpretovat jako parametry pro distribuci zbytku proměnných. Pro zprávu z faktoru do parametru je třeba provést aproximaci pravděpodobnostní distribuce proměnných a z ní odvodit nové parametry (popsáno v algoritmu 4 na straně 39).

4.4 Inferenční algoritmus

V této sekci si popíšeme poslední vrstvu implementovanou v knihovně a to vrstvu starající se o inferenci v grafických modelech. Předpokládáme, že již máme vytvořený graf z vrcholů, které byly představeny v předchozí sekci. V modulu **alex/infer/lbp.py** je implementována třída **LBP**, která implementuje iterativní algoritmus pro aktualizaci grafického modelu a také obsahuje několik implementací strategií pro různé typy grafů.

Pro výpočet v grafickém modelu je nejprve třeba zaregistrovat všechny vrcholy z grafu. K tomu existuje několik metod, odlišujících se podle typu grafu. Pro dynamické bayesovské sítě je možné přidávat vrcholy po vrstvách. K tomu slouží metody `add_layer` a `add_layers`, první metoda přidá jednu vrstvu na konec sítě. Druhá metoda umožňuje přidat více vrstev naráz. Pro obecné grafy slouží metoda `add_nodes`, která umožňuje přidat vrcholy bez informace o jejich organizaci uvnitř sítě.

K mazání vrcholů ze sítě slouží metody `clear_nodes` a `clear_layers`.

Typ inference je možné zadat při vytváření třídy, příslušná strategie je pak použita při samotném výpočtu uvnitř metody `run`. Tato metoda pak provádí samotnou inferenci, je možné zadat počet iterací, v případě inference v dynamickém modelu i zadat od které vrstvy se má inference provádět.

Pro více výpočtů nad jedním grafickým modelem je možné místo inicializace zpráv pro každý vrchol zvlášť použít metodu `init_messages`, která tuto inicializaci provede u všech vrcholů.

4.4.1 Strategie výběru vrcholu v LBP algoritmu

Inference v grafu se může lišit podle typu faktor grafu, ale také podle nároků, které na výsledek máme. Nejjednodušší metodou pro výběr je nechat pořadí na uživateli algoritmu. Pro stromy máme speciální strategii, která zaručí konvergenci po jednom dopředném a jednom zpětném kroku propagace. V dialogových systémech je často používána dynamická bayesovská síť a nás zajímá pravděpodobnost proměnných v poslední vrstvě sítě. V takovém případě můžeme některé zprávy zanedbat, protože už příliš neovlivní proměnné, které nás zajímají.

Inference na stromě

Pro efektivní inferenci je třeba si pro každý vrchol pamatovat, kolik mu chybí zpráv, aby mohl jednu sám odeslat. Pro každý vrchol v s k sousedy je na začátku počet chybějících zpráv $k - 1$. Strom na alespoň dvou vrcholech obsahuje alespoň dva listy. Budeme tedy postupně odebírat vrcholy, jejichž počet chybějících zpráv je nulový. Rozešleme z nich zprávu do souseda, z kterého ještě zpráva nepřišla a snížíme jeho počet chybějících zpráv. Odebráním listu ze stromu vždy dostaneme opět strom. Postupně tak zmenšujeme strom, až dostaneme právě jeden vrchol, který získal všechny zprávy.

Po vypočtení marginální pravděpodobnosti ve stromě o jednom vrcholu můžeme zase přidávat vrcholy v obráceném pořadí než v jakém jsme je odebírali. Do každého přidaného vrcholu pak můžeme poslat zprávu a spočítat jeho marginální pravděpodobnost.

Inference v dynamické bayesovské síti

Dynamická bayesovská síť je nejčastější reprezentace dialogového stavu. V jedné vrstvě sítě je popsána jedna obrátka dialogu. Vrcholy v jedné vrstvě většinou závisí pouze na vrcholech ve stejné nebo předchozí vrstvě. Po každé obrátce nás pro účely komponenty řízení dialogu zajímají hlavně pravděpodobnosti proměnných v poslední vrstvě, tedy po aktuální obrátce.

Inferenci provádíme způsobem podobným indukci. Pro první vrstvu provedeme inferenci libovolným způsobem, může dokonce platit, že v rámci jedné vrstvy se jedná o strom. Po přidání k -té vrstvy předpokládáme, že byla provedena inference na předchozích $k - 1$ vrstvách a tedy zprávy v této části grafu jsou správné. Zprávy ve směru k nové vrstvě jsou stále správné, neznáme ovšem hodnotu zpráv z předposlední vrstvy do nové vrstvy a hodnotu zpráv v nové vrstvě. Pošleme tedy zprávy z předposlední vrstvy a provedeme inferenci v nové vrstvě.

Stále ještě zbývají zprávy z nové vrstvy zpět v síti. Čím dál do historie ovšem jdeme, tím menší vliv naše nová pozorování budou mít na vrcholy v dané vrstvě. Proto se většinou omezíme jen na posledních několik vrstev (1 až 3).

4.5 Příklady

Po popisu jednotlivých vrstev v předchozí sekci nyní přejdeme k příkladům použití knihovny. Nejprve ukážeme použití jednotlivých tříd a metod. Nakonec bude představen příklad systému použitého v Dialog State Tracking Challenge (DSTC) 2013 [31].

4.5.1 Použití jednotlivých komponent

Představíme příklady vytvoření a používání jednotlivých komponent, od implementace faktorů až po inferenci v grafickém modelu.

Faktor

Třída **Faktor** je popsána v sekci 4.1.1. Vytvoření faktoru je ukázáno v příkladu 1, zde máme jednoduchý generativní model pro výběr restaurace podle typu jídla. Vytvořený faktor obsahuje dvě proměnné, první je požadovaný typ jídla, druhá je pozorovaný požadovaný typ. Faktor může pocházet z generativního modelu, kde pravděpodobnost pozorování závisí na skutečné hodnotě. Zde vidíme, že je větší pravděpodobnost pozorování typu jídla, které uživatel opravdu chce. Pro indické jídlo je zde pravděpodobnost větší než pro čínské, v reálném dialogovém systému na tyto pravděpodobnosti může mít vliv například jazykový model, anebo

podobnost slov.

Příklad 1 Vytvoření faktoru

```
from factor import Factor

factor = Factor(
    ['food', 'food_obs'],
    {
        'food': ['chinese', 'indian'],
        'food_obs': ['obs_chinese', 'obs_indian'],
    },
    {
        ('indian', 'obs_indian'): 0.9,
        ('indian', 'obs_chinese'): 0.1,
        ('chinese', 'obs_indian'): 0.2,
        ('chinese', 'obs_chinese'): 0.8,
    })
```

Vytvořený faktor můžeme zobrazit (příklad 2), stačí použít příkaz `print`, popřípadě je možné použít metodu `pretty_print`, která vrátí řetězec s formátovaným výpisem faktoru. Nabízí možnost nastavení šířky tabulky a počtu desetinných míst.

Příklad 2 Zobrazení faktorů

```
>>> print factor
```

food	food_obs	Value
chinese	obs_chinese	0.8000000119
chinese	obs_indian	0.1999999881
indian	obs_chinese	0.09999999404
indian	obs_indian	0.8999999762

```
>>> print factor.pretty_print(width=40, precision=2)
```

food	food_obs	Value
chinese	obs_chinese	0.8
chinese	obs_indian	0.2
indian	obs_chinese	0.1
indian	obs_indian	0.9

Dále obsahuje faktor implementaci matematických operací, je možné používat standardní operátory. V příkladu 3 je ukázáno násobení dvou faktorů, již vytvořený faktor s faktorem, který reprezentuje přechodovou pravděpodobnost. Operace fungují i s konstantami (příklad 4).

Příklad 3 Násobení faktorů

```
>>> factor_trans = Factor(  
...     ['food', 'food_next'],  
...     {  
...         'food': ['chinese', 'indian'],  
...         'food_next': ['chinese', 'indian'],  
...     },  
...     {  
...         ('indian', 'indian'): 0.9,  
...         ('indian', 'chinese'): 0.1,  
...         ('chinese', 'indian'): 0.1,  
...         ('chinese', 'chinese'): 0.9,  
...     })  
>>> result = factor * factor_trans  
>>> print result.pretty_print(50, 2)
```

food	food_next	food_obs	Value

chinese	chinese	obs_chinese	0.72
chinese	chinese	obs_indian	0.18
chinese	indian	obs_chinese	0.08
chinese	indian	obs_indian	0.02
indian	chinese	obs_chinese	0.01
indian	chinese	obs_indian	0.09
indian	indian	obs_chinese	0.09
indian	indian	obs_indian	0.81

Příklad 4 Násobení faktorů konstantou

```
>>> result = factor * 0.5  
>>> print result.pretty_print(50, 2)
```

food	food_obs	Value

chinese	obs_chinese	0.4
chinese	obs_indian	0.1
indian	obs_chinese	0.05
indian	obs_indian	0.45

Další důležitou metodou, kterou faktory nabízí, je marginalizace proměnných. Předvedeme si ji na faktoru pro pozorování (příklad 5), z kterého chceme získat pouze marginální pravděpodobnosti pozorování.

Příklad 5 Marginalizace faktoru

```
>>> marginalized = factor.marginalize(['food_obs'])
>>> print marginalized.pretty_print(50, 2)
```

```
-----
```

food_obs	Value
obs_chinese	0.9
obs_indian	1.1

```
-----
```

Pokud je faktor použit pro reprezentaci pravděpodobnostního rozdělení, pak je možné, že při některých úkonech bude výsledkem nenormalizované pravděpodobnostní rozdělení. Faktor nabízí metodu pro normalizaci hodnot, která navíc bere v úvahu i podmíněné pravděpodobnosti (příklad 6).

Příklad 6 Normalizace faktoru

```
>>> uniform = Factor(
...     ['food', 'food_next'],
...     {
...         'food': ['chinese', 'indian'],
...         'food_next': ['chinese', 'indian'],
...     },
...     {
...         ('indian', 'indian'): 1,
...         ('indian', 'chinese'): 1,
...         ('chinese', 'indian'): 2,
...         ('chinese', 'chinese'): 2,
...     })
>>> uniform.normalize(parents=['food'])
>>> print uniform.pretty_print(50, 2)
```

```
-----
```

food	food_next	Value
chinese	chinese	0.5
chinese	indian	0.5
indian	chinese	0.5
indian	indian	0.5

```
-----
```

Nakonec faktor nabízí možnost zjistit, které ohodnocení jsou nejpravděpodobnější (příklad 7).

Příklad 7 Nejpravděpodobnější hodnoty

```
>>> food = Factor(  
...     ['food'],  
...     {  
...         'food': ['chinese', 'indian', 'french'],  
...     },  
...     {  
...         ('chinese',): 0.3,  
...         ('indian',): 0.6,  
...         ('french',): 0.1,  
...     })  
>>> food.most_probable(n=2)  
[('indian', 0.6), ('chinese', 0.3)]
```

Vrcholy

V této sekci ukážeme vytváření jednoduchého grafického modelu a následně posílání zpráv z jednoho vrcholu do druhého. V příkladu 8 vytvoříme jednoduchý skrytý Markovský model pro výběr typu jídla. Model bude mít 2 obrátky, v každé bude jedna skrytá a jedna pozorovaná proměnná. Tento model bude generativní, bude obsahovat faktor pro generování pozorování na základě skutečné hodnoty a také faktor pro přechodovou pravděpodobnost z jedné obrátky do druhé.

Po vytvoření grafického modelu je dalším krokem nastavení pozorovaných hodnot proměnných (příklad 9). Předpokládejme, že v první obrátce bylo pozorováno čínské jídlo s pravděpodobností 0.6 a indické s pravděpodobností 0.4. V druhé obrátce bylo pozorováno čínské jídlo s pravděpodobností 0.5 a indické s pravděpodobností 0.5. Tedy, v první obrátce si myslíme, že uživatel spíše preferuje čínské jídlo, ale z druhé obrátky už nevíme nic. Podíváme se, co z těchto informací zjistí náš grafický model.

Strom zakořeníme ve vrcholu `trans_factor` a začneme posílat zprávy od listů ke kořeni a pak zpět (příklad 10). Naším cílem je zjistit aposteriorní marginální pravděpodobnosti skrytých proměnných. Před odesláním je vždy třeba aktualizovat vnitřní reprezentaci vrcholů. Nakonec musíme normalizovat vrcholy s proměnnými.

Ve výsledku (příklad 11) tedy vidíme, že v druhé obrátce z pozorování sice nezískáme žádnou informaci, ale díky vysoké pravděpodobnosti přechodu zůstala pravděpodobnost čínského jídla stále vysoká. Toto pozorování zároveň snížilo pravděpodobnost v první obrátce.

Příklad 8 Jednoduchý generativní model

```
from alex.ml.bn.node import DiscreteVariableNode, DiscreteFactorNode
from alex.ml.bn.factor import Factor

hid_1 = DiscreteVariableNode('food_1', ['chinese', 'indian'])
obs_1 = DiscreteVariableNode('food_obs_1', ['chinese', 'indian'])
obs_factor_1 = DiscreteFactorNode('food_obs_factor_1', Factor(
    ['food_1', 'food_obs_1'],
    {
        'food_1': ['chinese', 'indian'],
        'food_obs_1': ['chinese', 'indian'],
    },
    {
        ('chinese', 'chinese'): 0.9,
        ('chinese', 'indian'): 0.1,
        ('indian', 'chinese'): 0.1,
        ('indian', 'indian'): 0.9,
    }
)))

hid_2 = DiscreteVariableNode('food_2', ['chinese', 'indian'])
obs_2 = DiscreteVariableNode('food_obs_2', ['chinese', 'indian'])
obs_factor_2 = DiscreteFactorNode('food_obs_factor_2', Factor(
    ['food_2', 'food_obs_2'],
    {
        'food_2': ['chinese', 'indian'],
        'food_obs_2': ['chinese', 'indian'],
    },
    {
        ('chinese', 'chinese'): 0.9,
        ('chinese', 'indian'): 0.1,
        ('indian', 'chinese'): 0.1,
        ('indian', 'indian'): 0.9,
    }
)))

trans_factor = DiscreteFactorNode('food_trans_factor', Factor(
    ['food_1', 'food_2'],
    {
        'food_1': ['chinese', 'indian'],
        'food_2': ['chinese', 'indian'],
    },
    {
        ('chinese', 'chinese'): 0.99,
        ('chinese', 'indian'): 0.01,
        ('indian', 'chinese'): 0.01,
        ('indian', 'indian'): 0.99,
    }
)))

obs_factor_1.connect(hid_1, parent=True)
obs_factor_1.connect(obs_1, parent=False)

obs_factor_2.connect(hid_2, parent=True)
obs_factor_2.connect(obs_2, parent=False)

trans_factor.connect(hid_1, parent=True)
trans_factor.connect(hid_2, parent=False)
```

Příklad 9 Nastavení pozorovaných hodnot

```
obs_1.assigned({
    ('chinese',): 0.6,
    ('indian',): 0.4,
})

obs_2.assigned({
    ('chinese',): 0.5,
    ('indian',): 0.5,
})
```

Příklad 10 Posílání zpráv

```
obs_1.message_to(obs_factor_1)
obs_2.message_to(obs_factor_2)

obs_factor_1.update()
obs_factor_1.message_to(hid_1)

obs_factor_2.update()
obs_factor_2.message_to(hid_2)

hid_1.update()
hid_1.message_to(trans_factor)

hid_2.update()
hid_2.message_to(trans_factor)

trans_factor.update()
trans_factor.send_messages()

hid_1.update()
hid_1.normalize()

hid_2.update()
hid_2.normalize()
```

Příklad 11 Výsledek inference

```
>>> print hid_1.belief.pretty_print(50, 2)
```

food_1	Value
chinese	0.58
indian	0.42

```
>>> print hid_2.belief.pretty_print(50, 2)
```

food_2	Value
chinese	0.58
indian	0.42

Inference

Nyní se zbavíme manuálního posílání zpráv z příkladu 10 a nahradíme jej použitím třídy pro Loopy Belief Propagation (příklad 12). Zvolíme strategii pro stromy a výsledek bude stejný jako v případě s manuálním posíláním zpráv.

Příklad 12 Inference s pomocí LBP

```
from alex.ml.bn.lbp import LBP
```

```
lbp = LBP(strategy='tree')
```

```
lbp.add_nodes([obs_1, obs_2, hid_1, hid_2, obs_factor_1, obs_factor_2,  
               trans_factor])
```

```
lbp.run()
```

4.5.2 Učení dirichletovských parametrů

Nyní grafický model z minulé sekce upravíme tak, aby parametry přechodové pravděpodobnosti pocházely z dirichletovského rozdělení. Vše zůstane stejné, pouze faktor pro přechodovou pravděpodobnost nahradíme **DirichletFactorNode** a k němu připojíme nový vrchol **DirichletParameterNode**, který bude reprezentovat parametry. Vytvoření grafického modelu je v příkladu 13.

Druhý parametr konstruktoru **DirichletParameterNode** nyní neurčuje pravděpodobnosti, ale parametry α dirichletovských rozdělení. Pro každou kombinaci hodnot rodičů existuje jedno dirichletovské rozdělení popisující apriorní rozdělení nad pravděpodobnostmi proměnné, která je potomkem faktoru. V tomto případě je to skrytá proměnná v druhé obrátce. Parametry můžou být buď nastaveny bez

apriorní znalosti, pak jsou všechny rovny 1, anebo můžeme nějakou naši apriorní znalost přidat. V tomto případě budeme předpokládat, že se spíše cíl nemění.

Tentokrát budou pozorování nastavena tak, že je zřejmé, který typ jídla uživatel požaduje. Inferenci provedeme naprosto stejně jako v předchozím případě (příklad 14). Zajímavý je výsledek inference, protože se nezměnily pouze hodnoty skrytých proměnných, ale také dirichletovský parametr. V příkladu 15 je vidět, že vypočítané hodnoty skrytých proměnných jsou blízké pozorovaným hodnotám v každé obrátce, to je způsobeno tím, že parametry pro přechodovou pravděpodobnost jsme nenastavili příliš informativně. Zároveň je vidět, že byly parametry modifikovány, pokud uživatel v první obrátce chce čínské jídlo, pak je větší pravděpodobnost, že v druhé obrátce jej bude chtít také.

Nyní ještě ukážeme jak je možné použít stejný model vícekrát, s různými pozorováními. Představme si, že nyní máme další hovor o dvou obrátcích, avšak s jinými pozorovanými hodnotami. Stačí nastavit nová pozorování, znovu inicializovat zprávy a spustit inferenci. Výhodou je, že není třeba znovu vytvářet síť, navíc počítáme už s upraveným dirichletovským parametrem pro přechodovou pravděpodobnost. Výsledek je v příkladu 16.

4.6 Dialog State Tracking Challenge

Knihovna pro inferenci byla použita v Dialog State Tracking Challenge (DSTC) 2013 [31]. Nejprve popíšeme problém, následně představíme použitý model a nakonec porovnáme výsledky modelu s ostatními systémy.

Cílem DSTC bylo vytvořit prostředky pro porovnání různých přístupů k inferenci dialogového stavu a vyhodnotit jejich úspěšnost pomocí společné množiny metrik. Organizátoři poskytli několik anotovaných množin dat, které pocházejí z reálného použití tří různých dialogových systémů pro úlohu Let's Go! [23]. Zároveň poskytli i baseline dialogový systém. Cílem účastníků soutěže bylo vytvořit systém pro odhad dialového stavu, který správně predikuje dialogový stav na základě vstupu od uživatele a minulé akce systému.

4.6.1 Let's Go!

Úlohou dialogového systému Let's Go! je poskytnout telefonní službu pro zjišťování autobusového spojení v Pittsburghu. Systém rozpoznává 9 různých slotů, některé z nich sestávající z podslotů. Tyto sloty jsou

- *route* – linka,
- *time* – čas odjezdu nebo příjezdu,

Příklad 13 Vytvoření grafického modelu s dirichletovskými parametry

```
from alex.ml.bn.node import (DiscreteVariableNode, DiscreteFactorNode,
                             DirichletParameterNode,
                             DirichletFactorNode)
from alex.ml.bn.factor import Factor
from alex.ml.bn.lbp import LBP

obs_probability = {
    ('chinese', 'chinese'): 0.9,
    ('chinese', 'indian'): 0.1,
    ('indian', 'chinese'): 0.1,
    ('indian', 'indian'): 0.9,
}

hid_1 = DiscreteVariableNode('food_1', ['chinese', 'indian'])
obs_1 = DiscreteVariableNode('food_obs_1', ['chinese', 'indian'])
obs_factor_1 = DiscreteFactorNode('food_obs_factor_1', Factor(
    ['food_1', 'food_obs_1'],
    {
        'food_1': ['chinese', 'indian'],
        'food_obs_1': ['chinese', 'indian'],
    },
    obs_probability))

hid_2 = DiscreteVariableNode('food_2', ['chinese', 'indian'])
obs_2 = DiscreteVariableNode('food_obs_2', ['chinese', 'indian'])
obs_factor_2 = DiscreteFactorNode('food_obs_factor_2', Factor(
    ['food_2', 'food_obs_2'],
    {
        'food_2': ['chinese', 'indian'],
        'food_obs_2': ['chinese', 'indian'],
    },
    obs_probability))

trans_factor = DirichletFactorNode('food_trans_factor')
trans_param = DirichletParameterNode('food_trans_param', Factor(
    ['food_1', 'food_2'],
    {
        'food_1': ['chinese', 'indian'],
        'food_2': ['chinese', 'indian'],
    },
    {
        ('chinese', 'chinese'): 2,
        ('chinese', 'indian'): 1,
        ('indian', 'chinese'): 1,
        ('indian', 'indian'): 2,
    }
)))

obs_factor_1.connect(hid_1, parent=True)
obs_factor_1.connect(obs_1, parent=False)

obs_factor_2.connect(hid_2, parent=True)
obs_factor_2.connect(obs_2, parent=False)

trans_factor.connect(hid_1, parent=True)
trans_factor.connect(hid_2, parent=False)
trans_factor.connect(trans_param)
```

Příklad 14 Nastavení pozorovaných proměnných a inference

```
obs_1.assigned({
    ('chinese',): 0.9,
    ('indian',): 0.1,
})

obs_2.assigned({
    ('chinese',): 0.8,
    ('indian',): 0.2,
})

lbp = LBP(strategy='tree')
lbp.add_nodes([obs_1, obs_2, hid_1, hid_2, obs_factor_1, obs_factor_2,
               trans_factor, trans_param])
lbp.run()
```

Příklad 15 Vypočtené hodnoty skrytých proměnných

```
>>> print hid_1.belief.pretty_print(50, 2)
```

```
-----
              food_1              Value
-----
              chinese              0.86
              indian              0.14
-----
```

```
>>> print hid_2.belief.pretty_print(50, 2)
```

```
-----
              food_2              Value
-----
              chinese              0.81
              indian              0.19
-----
```

```
>>> print trans_param.alpha.pretty_print(50, 2)
```

```
-----
              food_1      food_2      Value
-----
              chinese      chinese      2.3
              chinese      indian      0.96
              indian      chinese      1.0
              indian      indian      2.0
-----
```

Příklad 16 Využití modelu pro nová pozorování

```
>>> obs_1.observed({
...     ('chinese',): 0.1,
...     ('indian',): 0.9,
... })
>>> obs_2.observed({
...     ('chinese',): 0.2,
...     ('indian',): 0.8,
... })
>>> lbp.init_messages()
>>> lbp.run()
>>> print trans_param.alpha.pretty_print(50, 2)
```

food_1	food_2	Value
chinese	chinese	2.3
chinese	indian	0.97
indian	chinese	0.97
indian	indian	2.3

- *date* – datum odjezdu,
- *from.description* – popis místa odjezdu,
- *to.description* – popis místa příjezdu,
- *from.monument* – významný monument v místě odjezdu,
- *to.monument* – významný monument v místě příjezdu,
- *from.neighborhood* – část města (sousedství) z které chce uživatel jet,
- *to.neighborhood* – část města (sousedství) kam chce uživatel jet.

Navíc se sloty *time* a *date* skládají z dalších 5, respektive 4 slotů (např. *time.hour*, *time.minute*). Systém Let's Go celkem rozpoznával více než 5000 míst a zhruba 150 různých linek.

Jednotlivé dialogy se skládají z obrátek, ve kterých se střídá uživatel a systém. Příklad rozhovoru je v tabulce 4.3. Uživatel může informovat systém o hodnotě libovolného slotu. Systém může uživatele požádat o hodnotu libovolného slotu, popř. o potvrzení hodnoty slotu. Vstup a výstup je reprezentován dialogovými akty. Používány jsou pouze dialogové akty *inform*, *deny*, *affirm* a *negate*. Zbytek dialogových aktů je ignorován, protože nemění cíle uživatele.

Organizátoři poskytli 4 datové sady, které pocházely ze 3 různých dialogových systémů. Každá datová sada obsahovala hypotézy z živého běhu dialogového

Obrátka	Aktér	Přepis
1	Systém dact	<i>Welcome to the Lets Go! bus information system for Pittsburgh. Where would you like to leave from?</i> hello()&request(from.desc)
	Uživatel dact	<i>The South Side.</i> inform(from.desc=„the south side“)
2	Systém dact	<i>Leaving from the south side neighborhood. Did I get that right?</i> confirm(from.desc=„the south side“)
	Uživatel dact	<i>Correct.</i> affirm()
3	Systém dact	<i>Where do you wanna go?</i> request(to.desc)
	Uživatel dact	<i>Oakland.</i> inform(to.desc=„Oakland“)
4	Systém dact	<i>When do you wanna travel?</i> request(time)
	Uživatel dact	<i>Next.</i> inform(time=next)
5	Systém dact	<i>The next bus is the 54C, leaving from south eighteen-th street at east Carson in the south side at 10:25pm. It arrives at fifth avenue at Bigelow in Oakland at 10:35pm. You may now say. next bus, previous bus, re-start, or goodbye.</i> ...

Tabulka 4.3: Příklad dialogu mezi uživatelem a dialogovým systémem Let's Go! *dact* znamená dialogový akt.

systému (live data). Dva dialogové systémy produkovaly seznam n nejlepších hypotéz. Jeden systém produkoval pouze nejlepší hypotézu, záznamy hovorů proto byly zpracovány zpětně pro vygenerování seznamů n nejlepších hypotéz (batch data). Vygenerované hypotézy ovšem obsahovaly pouze skóre, které muselo být přepočítáno pro získání pravděpodobností. Organizátoři nevydali žádné informace o povaze skóre.

Data obsahovala zjevně chybné hypotézy, např. v jedné hypotéze se objevily dvě různé hodnoty pro jeden slot. Tyto hypotézy byly z dat odstraněny ještě před samotnou inferencí.

4.6.2 Popis systému

Dialogový stav byl v použitém systému modelován jednoduchým generativním modelem, kde pro každý slot v jedné obrátce existují dva vrcholy, skutečná hodnota slotu s_t a pozorovaná hodnota o_t . Stav důvěry skutečné hodnoty slotu $b(s_t)$ závisí pouze na hodnotě v minulé obrátce s_{t-1} a na poslední systémové akci a_{t-1} . Pozorovaná hodnota o_t závisí pouze na skutečné hodnotě s_t .

Stav důvěry můžeme vyjádřit ze sdružené pravděpodobnosti generativního modelu:

$$b(s_t) = \sum_{s_{t-1}, o_t} p(s_t | a_{t-1}, s_{t-1}) p(o_t | s_t) b(s_{t-1}) \quad (4.1)$$

Pro zrychlení výpočtu bylo využito stažených parametrů s manuálně nastavenými hodnotami [27]:

$$p(s_t | a_{t-1}, s_{t-1}) = \begin{cases} \theta_t & \text{pokud } s_t = s_{t-1} \\ \frac{1-\theta_t}{|hodnoty|-1} & \text{jinak} \end{cases}, \quad (4.2)$$

kde θ_t je pravděpodobnost, že hodnota slotu se nemění a $|hodnoty|$ je počet hodnot pro slot.

Pro model pozorování byly také použity stažené pravděpodobnosti:

$$p(o_t | s_t) = \begin{cases} \theta_o & \text{pokud } o_t = s_t \\ \frac{1-\theta_o}{|hodnoty|-1} & \text{jinak} \end{cases}, \quad (4.3)$$

kde θ_o je pravděpodobnost, že pozorování bude odpovídat skutečné hodnotě slotu.

Parametr θ_t určuje jak moc bude systém zapomínat, v případě hodnoty blízké 1 systém téměř nezapomíná a jakmile nějakou hodnotu pozoruje, bude jej těžké přesvědčit o čemkoliv jiném, v opačném případě naopak systém zapomíná a i z téměř jisté důvěry v hodnotu stavu může během pár obrátek dojít do stavu, kdy mají všechny možné hodnoty stejnou pravděpodobnost. Parametr θ_o vyjadřuje

důvěru systému v SLU. Pokud je jeho hodnota vysoká, předpokládáme, že SLU téměř nedělá chyby. V opačném případě je systém tolerantní k chybám SLU. Na základě výsledků dialogového systému na trénovacích dat byly manuálně zvoleny vhodné hodnoty parametrů: $\theta_t = 0.8$ a $\theta_o = 0.8$

Představený model pro pozorování platí pro *inform* dialogové akty, dialogové sloty *affirm* a *negate* byly převedeny na *inform*.

Pro inferenci bylo použito LBP a protože možných hodnot slotů bylo v řádu stovek, všechny nepozorované hodnoty byly staženy do jedné speciální hodnoty a jejich pravděpodobnost byla počítána pouze dohromady.

4.6.3 Evaluace

Použitý model byl testován na 4 testovacích datových sadách s live daty a na dvou testovacích datových sadách s batch daty. Výsledky byly porovnány s baseline dialogovým systémem vytvořeným organizátory. Použité metriky byly přesnost (accuracy) dialogového systému a Brier score. Přesnost je podíl obrátek, ve kterých byla nejlepší hypotéza systému pro odhad stavu správná, oproti celkovému počtu obrátek. Brier score měří přesnost prediktivní pravděpodobnostní distribuce systému pro odhad stavu [8] (čím menší, tím lepší).

Výsledky ukazují (viz tabulka 4.4), že generativní model překonává baseline na všech datových sadách, kromě batch dat. Manuální inspekci bylo zjištěno, že generativní model je velmi citlivý na vstupní pravděpodobnosti. Pravděpodobnosti hypotéz z batch dat, zřejmě kvůli způsobu přepočítání skóre na pravděpodobnosti, jsou velmi blízko sebe a tedy nedávají dostatek informací pro pravděpodobnostní model.

Generativní systém byl také porovnán s ostatními dialogovými systémy přihlášenými do DSTC. Zde byl rovnocenným soupeřem většině ostatních týmů (viz tabulka 4.5). Ve výsledcích bude generativní systém veden jako 2. systém týmu číslo 3.

Tabulky výsledků a příklad dialogu byly převzaty z článku popisujícího přihlášené systémy do DSTC a porovnávajícího generativní model s jednoduchým diskriminativním modelem [39].

Popis jednotlivých součástí systému a jeho používání je v příloze A.

4.7 Existující řešení

Většina existujících knihoven pro inferenci v Bayesovských sítích je vytvořena pro obecné grafické modely. Použití v dialogových systémech se ovšem liší od předpokládaného využití většiny knihoven. Při řešení problémů je jen málokdy

live data	metric	BT	GT
test1	přesnost	0.77	0.88
	Brier score	0.29	0.21
test2	přesnost	0.79	0.85
	Brier score	0.27	0.23
test3	přesnost	0.92	0.93
	Brier score	0.14	0.16
test4	přesnost	0.82	0.87
	Brier score	0.24	0.20
ALL	přesnost	0.83	0.88
	Brier score	0.24	0.20
batch data	metric	BT	GT
test1	přesnost	0.75	0.74
	Brier score	0.35	0.39
test2	přesnost	0.79	0.77
	Brier score	0.30	0.33
ALL	přesnost	0.77	0.76
	Brier score	0.32	0.36

Tabulka 4.4: Přesnost systému pro odhad stavu na live a batch testovacích datech, kde BT je baseline systém a GT je generativní systém. ALL značí průměrné skóre přes všechny testovací sady.

potřeba získat výsledek v řádu milisekund, i na úkor kvality aproximace. Ovšem při hovoru s uživatelem není možné si dovolit dlouhé pauzy, působí rušivě. Častým problémem pak u těchto knihoven je nemožnost volby aproximace.

Populární knihovna, která umožňuje použití Bayesovských modelů pro širokou škálu problémů, je Infer.Net, pocházející z Microsoft Research [15]. Tato knihovna nabízí velkou řadu inferenčních algoritmů. Jak název napovídá, je napsaná v prostředí .Net a tedy je nutné ji používat z jazyka, který běží v tomto prostředí (C#, F# atd.)

Dále existuje řada knihoven, které lze použít pro inferenci v Bayesovských sítích, ovšem namísto variačních metod používají vzorkování. Příkladem je knihovna PyMC [19]. Díky využití metod Monte Carlo je jednodušší počítání se složitějšími distribucemi, ovšem na druhou stranu platíme mnohdy pomalou konvergencí.

tým / systém	přesnost	Brier score
BT - C	0.81	0.27
BT	0.83	0.24
GT	0.88	0.20
team1	0.88	0.23
team2	0.88	0.21
team4	0.81	0.28
team5	0.88	0.21
team6	0.91	0.18
team7	0.85	0.23
team8	0.83	0.24
team9	0.89	0.20

Tabulka 4.5: Přesnost systémů přihlášených do DSTC. BT - C značí baseline systém bez mazání chybných hypotéz, BT je baseline systém, GT je generativní systém a dále následují systémy ostatních týmů. Skóre jsou zprůměrována přes všechny 4 testovací sady dat.

Závěr

Práce splnila všechny vytyčené cíle. Výsledkem je implementace metod pro odhad stavu a parametrů v dialogových systémech. Byly představeny metody pro inferenci v grafických modelech založené na posílání zpráv, navíc byl odvozen algoritmus pro inferenci parametrů dirichletovského rozdělení srovnáním momentů. Práce obsahuje příklady použití vytvořené knihovny a otestování na reálných datech v rámci soutěže Dialog State Tracking Challenge, kde generativní systém pro odhad stavu používající implementovanou knihovnu dosáhl výsledků srovnatelných s ostatními přihlášenými týmy, které se dají považovat za state of the art. Bližší detaily podle cílů vytyčených v úvodu práce:

1. V kapitole 1 byly představeny dialogové systémy a jejich jednotlivé součásti. V sekci 2.1 pak byly představeny bayesovské sítě, které se nabízejí jako ideální model pro reprezentaci dialogového stavu.
2. V kapitole 2 jsou prezentovány možné inferenční algoritmy pro odhad stavu v bayesovských sítích. V sekci 2.2.4 je představen algoritmus Loopy Belief Propagation. Jeho použití je ukázáno v sekci 4.5.1.
3. Implementace LBP byla použita pro DSTC a popis celé soutěže i s výsledky je v sekci 4.6. Implementovaný systém pro odhad stavu byl lepší než baseline systém poskytnutý organizátory (tabulka 4.4) a systém se navíc umístil příznivě i vzhledem k ostatním přihlášeným týmům (tabulka 4.5).
4. Implementované strategie pro výběr vrcholu v LBP algoritmu jsou popsány v sekci 4.4.1.
5. Algoritmus Expectation Propagation pro inferenci v obecných grafických modelech je představen v sekci 2.3.2.
6. Knihovna a jednotlivé její součásti jsou popsány v kapitole 4. Příklad vrcholů vytvořených speciálně pro použití v EP je v sekci 4.3. Učení parametrů pro odhad stavu v dialogových systémech je odvozeno v kapitole 3.

A. Zpracování dialogů z DSTC

A.1 Soubory

Předvedeme použití vytvořené knihovny na data z Dialog State Tracking Challenge. Všechny potřebné data a zdrojové kódy jsou na přiloženém cd. V adresáři `src/dstc` je velké množství modulů a dalších adresářů:

- `common.py`, `dstcutils.py`, `jsonpath.py`, `ontology.py`, `pddiscrete.py`, `preprocessing.py` – pomocné moduly pro práci s daty, vytvořené členy výzkumné skupiny pracující na projektu Vystadial,
- `loader.py` – modul pro načítání dat z formátu JSON [9] do interní reprezentace,
- `system_dm.py` – skript pro zpracování dat používající vytvořenou knihovnu pro odhad stavu v jednotlivých dialogích. Použitá datová sada, vstupní a výstupní soubory, parametry grafického modelu a další parametry nutné pro výpočet je možné skriptu zadat buď pomocí přepínačů na příkazové řádce, anebo v konfiguračním souboru.
- `test_run.sh` – bashový skript pro spuštění celé pipeline od zpracování dat až po zobrazení výsledků,
- `test1.conf`, `test2.conf`, `test3.conf`, `test4.conf` – konfigurační soubory pro spuštění skriptu `system_dm.py` na testovací datové sady ze všech čtyř dialogových systémů.
- V adresáři `src/dstc/bin/` je baseline dialogový systém a skripty pro zpracování výsledků od organizátorů soutěže:
 - `baseline` – baseline dialogový systém,
 - `score` – skript pro vyhodnocení výstupu soutěžního systému,
 - `report` – skript pro zobrazení výsledků soutěžního systému v čitelné podobě,
 - `validate` – skript pro validaci korektního výstupu soutěžního systému,
 - Pomocné moduly pro tyto skripty jsou v adresáři `src/dstc/lib`.
- V adresáři `src/dstc/data/` jsou všechny 4 testovací datové sady, každá ve vlastním adresáři (`test1/`, `test2/`, `test3/`, `test4/`). Každá datová sada

se pak skládá z podadresářů pro jednotlivé hovory. Každý hovor se skládá ze dvou souborů:

- **dstc.log.json** – záznam z dialogového systému obsahující všechny informace o dialogu ve formátu JSON,
 - **dstc.labels.json** – soubor obsahující správné hypotézy, tedy řešení úlohy, tento soubor samozřejmě před skončením soutěže v datech chyběl.
- V adresáři **src/dstc/config/** jsou soubory obsahující cesty k jednotlivým dialogům pro každou testovací sadu.

A.2 Konfigurace

Skript **system_dm.py** očekává řadu parametrů pro správné fungování. Parametry je možné zadat na příkazové řádce, například:

```
$ python system_dm.py \  
  --dataroot=data/ \  
  --dataset=config/test1.sessions \  
  --scorefile=track_test1.json
```

Popřípadě je možné vytvořit konfigurační soubor obsahující pouze parametry:

```
--dataroot=data/  
--dataset=config/test1.sessions  
--scorefile=track_test1.json
```

a tento soubor pak předat jako jediný parametr:

```
$ python system_dm.py --flagfile test1.conf
```

Většina vlastností systému pro odhad stavu je konfigurovatelná, ale jen malý počet parametrů je povinný. Představíme si ty nejdůležitější parametry:

- *dataroot* – povinný, adresář obsahující datové sady,
- *dataset* – povinný, konfigurační soubor obsahující seznam cest k jednotlivým dialogům,
- *scorefile* – povinný, název souboru, do kterého má být zapsán výsledek zpracování dialogů,
- *slu* (live / batch) – přepínač, zda-li pracovat s hypotézami z live nebo batch dat,

- *remove_inconsistent* – mazat nekonzistentní hypotézy (např. v jedné hypotéze více hodnot pro jeden slot),
- *processes* – systém podporuje zpracování na více jádrech, pomocí tohoto přepínače je možné nastavit počet jader, pro jedno jádro systém přepne do ladícího módu, kdy si udržuje v paměti všechny dialogy; nehodí se pro velké datové sady,
- *prob_same* – hodnota parametru θ_o , tedy pravděpodobnost, že pozorování odpovídá skryté proměnné,
- *prob_trans* – hodnota parametru θ_t , tedy pravděpodobnost, že se hodnota skryté proměnné nezmění v další obrátce.

A.3 Výpočet

V předchozí sekci bylo ukázáno jakým způsobem lze nastavit a spustit skript `system_dm.py`, který provede odhad dialogového stavu. Výsledkem je soubor ve formátu JSON, kde pro každý dialog a každou jeho obrátku jsou vyznačeny hypotézy pro každý slot. V příkladu 17 je ukázáno jak vypadá výstup pro jeden slot, v tomto případě se jedná o počáteční zastávku, kde hodnota „shadyside“ má pravděpodobnost přibližně 0.56.

Příklad 17 Část výsledku systému pro odhad stavu.

```
"from.neighborhood": {
  "hyps": [
    {
      "score": 0.5637651968841264,
      "slots": {
        "from.neighborhood": "shadyside"
      }
    }
  ]
},
```

Nyní by bylo třeba výsledky zpracovat a následně zobrazit. Všechny tři kroky je možné provést naráz použitím skriptu `test_run.sh`:

```
$ ./test_run.sh test1.conf
```

Ukázka výsledků je v příklad 18 a 19. Ve výsledku jsou zobrazeny tři tabulky s výsledky. V první tabulce („schedule1“) jsou statistiky počítány po každé obrátce. V druhé tabulce („schedule2“) jsou statistiky počítány pro slot pouze pokud se o

něm v dané obrátce mluvílo. Nakonec ve třetí tabulce („schedule3“) jsou statistiky počítány pouze na konci dialogu.

Jednotlivé sloupce reprezentují různé sloty, navíc je zde sloupec *joint*, který hodnotí sdruženou informaci o všech slotech v dané obrátce, a sloupec *all*, který obsahuje průměrné skóre.

Jednotlivé řádky obsahují statistiky o slotech, zvlášť zajímavé jsou:

- accuracy – přesnost,
- avgp – průměrné skóre přiřazené správné hypotéze,
- l2 – Brier skóre.

Nakonec jsou zobrazeny i základní informace o celém výpočtu, například jaká datová sada byla zpracována, počet hovorů, počet obrátek, celkový čas a průměrný čas na jednu obrátku.

Příklad 18 Výsledek systému pro odhad stavu podle prvních dvou typů hodnocení.

schedule1									

	route	from.d	from.m	from.n	to.des	to.mon	to.nei	joint	all
N	10085	10085	10085	10085	10085	10085	10085	10085	90765
accuracy	0.8817	0.6969	0.9841	0.7771	0.8256	0.9845	0.7327	0.2642	0.8534
avgp	0.8794	0.6779	0.9811	0.6765	0.8235	0.9828	0.7185	0.2465	0.8370
hypcount	0.0773	0.2038	0.0103	0.2442	0.0435	0.0045	0.0353	0.4943	0.0729
l2	0.1703	0.4542	0.0268	0.4575	0.2495	0.0243	0.3981	1.0655	0.2304
mrr	0.8977	0.7239	0.9850	0.7838	0.8329	0.9846	0.7333	0.2888	0.8606
nonempty	0.0682	0.1660	0.0103	0.2442	0.0367	0.0045	0.0348	0.4943	0.0669
roc.v1_ca05	0.0000	0.0000	0.9841	0.0000	0.0000	0.9845	0.0000	0.0000	0.0000
roc.v1_ca10	0.8617	0.0000	0.9841	0.0000	0.0000	0.9845	0.0000	0.0000	0.0000
roc.v1_ca20	0.8817	0.0000	0.9841	0.0000	0.8256	0.9845	0.0000	0.0000	0.8534
roc.v1_eer	0.8817	0.6969	0.9841	0.4288	0.8256	0.9845	0.7327	0.2642	0.8534
roc.v2_ca05	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
roc.v2_ca10	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
roc.v2_ca20	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

schedule2									

	route	from.d	from.m	from.n	to.des	to.mon	to.nei	joint	all
N	6100	4837	926	2706	4070	904	2283	9974	27536
accuracy	0.8590	0.6729	0.9471	0.6936	0.6784	0.9436	0.4717	0.2669	0.7309
avgp	0.8589	0.6685	0.9449	0.6414	0.6773	0.9351	0.4496	0.2490	0.7212
hypcount	0.0807	0.1784	0.0248	0.1778	0.0744	0.0243	0.0604	0.4927	0.0934
l2	0.1994	0.4670	0.0780	0.5071	0.4562	0.0918	0.7784	1.0621	0.3939
mrr	0.8769	0.6997	0.9514	0.7051	0.6907	0.9441	0.4733	0.2916	0.7443
nonempty	0.0723	0.1333	0.0248	0.1778	0.0585	0.0243	0.0600	0.4927	0.0812
roc.v1_ca05	0.0000	0.0000	0.9471	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
roc.v1_ca10	0.0000	0.0000	0.9471	0.0000	0.0000	0.9436	0.0000	0.0000	0.0000
roc.v1_ca20	0.8590	0.0000	0.9471	0.0000	0.0000	0.9436	0.0000	0.0000	0.0000
roc.v1_eer	0.8590	0.6729	0.9471	0.6936	0.6784	0.9436	0.4717	0.2669	0.7309
roc.v2_ca05	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
roc.v2_ca10	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
roc.v2_ca20	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Příklad 19 Výsledek systému pro odhad stavu podle třetího typu hodnocení a základní statistiky.

schedule3										

	route	from.d	from.m	from.n	to.des	to.mon	to.nei	joint	all	
N	897	874	422	767	767	354	620	901	5784	
accuracy	0.8863	0.6739	0.9645	0.7471	0.7249	0.9520	0.4048	0.1576	0.7076	
avgp	0.8827	0.6368	0.9496	0.5949	0.7159	0.9426	0.3581	0.1526	0.6692	
hypcount	0.0836	0.2574	0.0355	0.3494	0.0704	0.0198	0.1145	0.6071	0.1376	
l2	0.1656	0.5109	0.0713	0.5730	0.4012	0.0811	0.9077	1.1984	0.4673	
mrr	0.9036	0.7008	0.9656	0.7562	0.7347	0.9520	0.4073	0.1770	0.7184	
nonempty	0.0758	0.2082	0.0355	0.3494	0.0600	0.0198	0.1129	0.6071	0.1272	
roc.v1_ca05	0.0000	0.0000	0.9645	0.0000	0.0000	0.9520	0.0000	0.0000	0.0000	
roc.v1_ca10	0.8740	0.0000	0.9645	0.0000	0.0000	0.9520	0.0000	0.0000	0.0000	
roc.v1_ca20	0.8863	0.0000	0.9645	0.0000	0.0000	0.9520	0.0000	0.0000	0.0000	
roc.v1_eer	0.8863	0.6739	0.0664	0.5176	0.7249	0.9520	0.4048	0.1576	0.7076	
roc.v2_ca05	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
roc.v2_ca10	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
roc.v2_ca20	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	

basic stats										

dataset		: config/test1.sessions								
scorer_version		: 1.7								
sessions		: 715								
total_wall_time		: 114.580698967								
turns		: 10085								
wall_time_per_turn		: 0.0113614971707								

B. Dokumentace tříd

B.1 Balíček **alex.infer**

B.1.1 Modul **factor**

Modul **factor** obsahuje implementaci faktorů pro počítání s diskrétními pravděpodobnostními rozděleními.

exception alex.infer.factor.FactorError

Výjimka značící chybu při manipulaci s faktorem.

class alex.infer.factor.Factor

Třída reprezentující diskrétní faktor s definovanými operacemi.

Je možné použít všechny standardní matematické operátory (+, -, *, /, **). Operátor je možné aplikovat na dva faktory, anebo na faktor a skalární hodnotu.

```
__init__(variables, variable_values, prob_table[, logarithme-  
          tic=True])
```

Vytvoření nového faktoru.

```
f = Factor(['A', 'B'],  
          {'A': ['a1', 'a2'], 'B': ['b1', 'b2']},  
          {  
              ('a1', 'b1'): 0.8,  
              ('a1', 'b2'): 0.2,  
              ('a2', 'b1'): 0.3,  
              ('a2', 'b2'): 0.7  
          })
```

Parametry

- **variables** – seznam proměnných ve faktoru.
- **variable_values** – slovník obsahující pro každou proměnnou seznam možných hodnot.
- **prob_table** – slovník možných přiřazení proměnných a jejich ohodnocení.
- **logarithmetic** – přepínač určující, zda-li použít aritmetiku nad zlogaritmovanými hodnotami.

```
__getitem__(assignment)
```

Získání ohodnocení pro dané přiřazení.

```
>>> f = Factor(['A', 'B'],  
...          {'A': ['a1', 'a2'], 'B': ['b1', 'b2']},
```

```

...     {
...         ('a1', 'b1'): 0.8,
...         ('a1', 'b2'): 0.2,
...         ('a2', 'b1'): 0.3,
...         ('a2', 'b2'): 0.7
...     })
>>> f[('a2', 'b1')]
0.3

```

Parametr assignment – přiřazení proměnných.

Vrací hodnotu dotazovaného přiřazení.

__setitem__(*assignment, value*)

Nastavení hodnoty pro přiřazení proměnných.

```

>>> f = Factor(['A', 'B'],
...     {'A': ['a1', 'a2'], 'B': ['b1', 'b2']},
...     {
...         ('a1', 'b1'): 0.8,
...         ('a1', 'b2'): 0.2,
...         ('a2', 'b1'): 0.3,
...         ('a2', 'b2'): 0.7
...     })
>>> f[('a2', 'b1')] = 0.4
>>> f[('a2', 'b1')]
0.4

```

Parametry

- **assignment** – přiřazení proměnných.
- **value** – nová hodnota.

pretty_print(*width=79, precision=10*)

Vypsání faktoru ve formě tabulky.

```

>>> f = Factor(['A', 'B'],
...     {'A': ['a1', 'a2'], 'B': ['b1', 'b2']},
...     {
...         ('a1', 'b1'): 0.8,
...         ('a1', 'b2'): 0.2,
...         ('a2', 'b1'): 0.3,
...         ('a2', 'b2'): 0.7
...     })
>>> print f.pretty_print(50, 2)

```

```

-----
      A              B              Value

```

a1	b1	0.8
a1	b2	0.2
a2	b1	0.3
a2	b2	0.7

Parametry

- **width** – šířka tabulky.
- **precision** – počet desetinných míst.

Vrací řetězec s naformátovanou tabulkou.

`__iter__()`

Iterace přes jednotlivé hodnoty přiřazení.

```
>>> f = Factor(['A', 'B'],
...           {'A': ['a1', 'a2'], 'B': ['b1', 'b2']}},
...           {
...             ('a1', 'b1'): 0.8,
...             ('a1', 'b2'): 0.2,
...             ('a2', 'b1'): 0.3,
...             ('a2', 'b2'): 0.7
...           })
>>> for (assignment, value) in f:
...     print assignment, '->', value
('a1', 'b1') -> 0.8
('a1', 'b2') -> 0.2
('a2', 'b1') -> 0.3
('a2', 'b2') -> 0.7
```

`observed(assignment_dict)`

Nastavení pozorovaných hodnot.

```
>>> f = Factor(['A', 'B'],
...           {'A': ['a1', 'a2'], 'B': ['b1', 'b2']}},
...           {
...             ('a1', 'b1'): 0.8,
...             ('a1', 'b2'): 0.2,
...             ('a2', 'b1'): 0.3,
...             ('a2', 'b2'): 0.7
...           })
>>> f.observed({
...     ('a1', 'b1'): 0.6,
...     ('a1', 'b2'): 0.4,
```

```
... })
>>> print f.pretty_print(50, 2)
```

A	B	Value
a1	b1	0.6
a1	b2	0.4
a2	b1	0.0
a2	b2	0.0

Parametr `assignment_dict` – slovník s pozorovanými přiřazeními a jejich ohodnocením.

`normalize([parents])`

Normalizace hodnot ve faktoru, aby jejich součet byl roven 1.

Pokud je specifikován parametr **`parents`**, pak musí být součet všech proměnných, které mají stejné přiřazení pro proměnné z **`parents`**, roven 1.

```
>>> f = Factor(['A', 'B'],
...           {'A': ['a1', 'a2'], 'B': ['b1', 'b2']}},
...           {
...             ('a1', 'b1'): 1,
...             ('a1', 'b2'): 3,
...             ('a2', 'b1'): 2,
...             ('a2', 'b2'): 2,
...           })
>>> f.normalize(parents=['A'])
>>> print f.pretty_print(50, 2)
```

A	B	Value
a1	b1	0.25
a1	b2	0.75
a2	b1	0.5
a2	b2	0.5

Parametr `parents` – seznam rodičovských proměnných.

`marginalize(keep)`

Vysčítání proměnných tak, aby zůstaly pouze proměnné z **`keep`**.

```
>>> f = Factor(['A', 'B'],
...           {'A': ['a1', 'a2'], 'B': ['b1', 'b2']}},
```

```

...     {
...         ('a1', 'b1'): 0.8,
...         ('a1', 'b2'): 0.2,
...         ('a2', 'b1'): 0.3,
...         ('a2', 'b2'): 0.7,
...     })
>>> g = f.marginalize(keep=['A'])
>>> print g.pretty_print(50, 2)
-----
              A              Value
-----
              a1              1.0
              a2              1.0
-----

```

Parametr keep – seznam proměnných, které mají zůstat.

most_probable($[n]$)

Nalezení nejpravděpodobnějších přiřazení proměnných.

```

>>> f = Factor(['A'],
...     {'A': ['a1', 'a2']},
...     {
...         ('a1',): 0.8,
...         ('a2',): 0.2,
...     })
>>> f.most_probable(2)
[('a1', 0.8), ('a2', 0.2)]

```

Parametr n – počet požadovaných nejpravděpodobnějších přiřazení.

Vrací seznam nejpravděpodobnějších přiřazení a jejich hodnot.

rename_variables(*mapping*)

Přejmenování proměnných.

Může se stát, že je potřeba přejmenovat proměnné, aby odpovídaly proměnným v jiném faktoru.

```

>>> f = Factor(['A', 'B'],
...     {'A': ['a1', 'a2'], 'B': ['b1', 'b2']},
...     {
...         ('a1', 'b1'): 0.8,
...         ('a1', 'b2'): 0.2,
...         ('a2', 'b1'): 0.3,

```



```

...         ('a2', 'b2'): 0.7,
...     })
>>> f.rename_variables({'A': 'X', 'B': 'Y'})
>>> print f.pretty_print(50, 2)
-----
      X           Y           Value
-----
    a1          b1           0.8
    a1          b2           0.2
    a2          b1           0.3
    a2          b2           0.7
-----

```

Parametr mapping – slovník se zobrazením ze stávajících názvů proměnných do nových názvů.

Příklady matematických operací

Násobení:

```

>>> f1 = Factor(['A', 'B'],
...             {'A': ['a1', 'a2'], 'B': ['b1', 'b2']}},
...             {
...                 ('a1', 'b1'): 0.8,
...                 ('a1', 'b2'): 0.2,
...                 ('a2', 'b1'): 0.3,
...                 ('a2', 'b2'): 0.7
...             })
>>> f2 = Factor(['B', 'C'],
...             {'B': ['b1', 'b2'], 'C': ['c1', 'c2']}},
...             {
...                 ('b1', 'c1'): 0.3,
...                 ('b1', 'c2'): 0.7,
...                 ('b2', 'c1'): 0.5,
...                 ('b2', 'c2'): 0.5,
...             })
>>> fr = f1 * f2
>>> print fr.pretty_print(50, 2)
-----
      A           B           C           Value
-----
    a1          b1          c1           0.24
    a1          b1          c2           0.56
    a1          b2          c1           0.10

```

a1	b2	c2	0.10
a2	b1	c1	0.09
a2	b1	c2	0.21
a2	b2	c1	0.35
a2	b2	c2	0.35

Umocnění:

```
>>> f1 = Factor(['A', 'B'],
...             {'A': ['a1', 'a2'], 'B': ['b1', 'b2']}},
...             {
...                 ('a1', 'b1'): 0.8,
...                 ('a1', 'b2'): 0.2,
...                 ('a2', 'b1'): 0.3,
...                 ('a2', 'b2'): 0.7
...             })
>>> fr = f1 ** 2
>>> print fr.pretty_print(50, 2)
```

A	B	Value
a1	b1	0.64
a1	b2	0.04
a2	b1	0.09
a2	b2	0.49

B.1.2 Modul **node**

Modul obsahuje implementace vrcholů tvořících grafický model. Pro výpočty jsou používány třídy z modulu **factor**.

class alex.infer.node.Node

Abstraktní třída pro reprezentaci vrcholů.

Obsahuje definice základních metod, které musí obsahovat každá implementace vrcholu.

__init__(*name*[, *aliases*])

Vytvoření nového vrcholu.

Při vytvoření vrcholu je třeba zadat jeho jméno, pod kterým bude možné jej nalézt při inferenci a je možné předat aliasy pro proměnné.

Aliases pro proměnné slouží k přejmenování proměnných ve zprávách, které přicházejí do vrcholu.

Parametry

- **name** – jméno vrcholu.
- **aliases** – zobrazení ze jmen proměnných, které se mohou objevit ve zprávě, do proměnných v parametru.

add_neighbor(*node*, ***options*)

Přidání souseda.

Pro většinu vrcholů bude stačit pouze vrchol, zůstává tu ovšem možnost přidat implementačně závislé parametry.

Parametry

- **node** – sousední vrchol.
- **options** – implementačně závislé parametry.

connect(*node*, ***options*)

Propojení dvou vrcholů.

Propojení dvou vrcholů, stačí zavolat pouze jednou pro dvojici vrcholů.

Interně zavolá metodu **add_neighbor()** a předá jí parametry.

Parametry

- **node** – sousední vrchol.
- **options** – implementačně závislé parametry.

init_messages()

Inicializace všech zpráv mezi vrcholem a jeho sousedy.

message_from(*node*, *message*)

Přijmutí zprávy ze sousedního vrcholu.

Zpráva je uložena a použita při dalších výpočtech.

Parametry

- **node** – sousední vrchol.
- **message** – zpráva ze sousedního vrcholu.

message_to(*node*)

Poslání zprávy do sousedního vrcholu.

Parametr node – sousední vrchol, kterému má být zpráva zaslána.

normalize(*[parents]*)

Normalizace pravděpodobnostního rozdělení.

Při počítání nejsou zprávy normalizované, proto ani výsledek nemusí být normalizovaný a je tedy třeba jej normalizovat před jakýmkoliv dalším použitím. Pro normalizaci je možné zadat, které proměnné jsou rodiče, pak musí být suma ohodnocení všech přiřazení se stejnými rodiči rovna jedné.

Parametr `parents` – seznam proměnných, které jsou rodiči vrcholu.

`update()`

Aktualizace interní pravděpodobnostní distribuce.

Příchozí zprávy jsou pouze uloženy a je potřeba aktualizovat interní reprezentaci jejich součinu před počítáním odchozích zpráv.

`class alex.infer.node.FactorNode`

Předek: `alex.infer.node.Node`

Abstraktní třída reprezentující vrcholy pro faktory.

`class alex.infer.node.VariableNode`

Předek: `alex.infer.node.Node`

Abstraktní třída reprezentující vrcholy pro proměnné.

Vrcholy pro diskrétní vrcholy

`class alex.infer.node.DiscreteVariableNode`

Předek: `alex.infer.node.VariableNode`

Reprezentace diskrétních proměnných.

`belief`

Marginální pravděpodobnostní rozdělení pro proměnnou. Typu **`Factor`**.

`__init__(name, values, logarithmic=True)`

Vytvoření nového vrcholu pro diskrétní proměnné.

Pro vytvoření je třeba jméno proměnné a hodnoty, kterých může nabývat.

Také je možné zadat, jestli používat při výpočtech trik s aritmetikou nad zlogaritmovanými čísly. Je třeba, aby všechny vrcholy v grafickém modelu byly nastaveny stejně.

Parametry

- **`name`** – název proměnné.
- **`values`** – seznam možných hodnot.

- **logarithmic** – zda-li používat trik s logaritmy.

add_neighbor(*node*)

Přidání souseda.

Parametr *node* – sousední vrchol.

connect(*node*, *options*)**

Propojení dvou vrcholů.

Propojení dvou vrcholů, stačí zavolat pouze jednou pro dvojici vrcholů.

Interně zavolá metodu **add_neighbor()** a předá jí parametry.

Parametr *node* – sousední vrchol.

init_messages()

Inicializace všech zpráv mezi vrcholem a jeho sousedy.

message_from(*node*, *message*)

Přijmutí zprávy ze sousedního vrcholu.

Zpráva je uložena a použita při dalších výpočtech.

Parametry

- **node** – sousední vrchol.
- **message** – zpráva ze sousedního vrcholu.

message_to(*node*)

Poslání zprávy do sousedního vrcholu.

Parametr *node* – sousední vrchol, kterému má být zpráva zaslána.

normalize()

Normalizace pravděpodobnostního rozdělení.

Při počítání nejsou zprávy normalizované, proto ani výsledek nemusí být normalizovaný a je tedy jej třeba normalizovat před jakýmkoliv dalším použitím.

update()

Aktualizace marginální pravděpodobnostní distribuce.

Příchozí zprávy jsou pouze uloženy a je potřeba aktualizovat interní reprezentaci jejich součinu před počítáním odchozích zpráv.

observed(*assignment_dict*)

Nastavení pozorovaných hodnot.

Proměnná je nastavená na pozorované rozdělení. Pravděpodobnost nepozorovaných hodnot je nulová.

Parametr `assignment_dict` – slovník se zobrazením z hodnot do jejich pravděpodobností.

`most_probable(n)`

Získání nejpravděpodobnějších hodnot.

Parametr `n` – počet nejpravděpodobnějších hodnot, které mají být vráceny.

Vrací seznam dvojic ve tvaru (hodnota, její pravděpodobnost) seřazený podle pravděpodobnosti.

`class alex.infer.node.DiscreteFactorNode`

Předek: `alex.infer.node.FactorNode`

Vrchol pro diskrétní faktor, reprezentující diskrétní pravděpodobnostní rozdělení.

`__init__(name, factor)`

Vytvoření nového vrcholu pro diskrétní faktor.

Pro vytvoření je třeba jméno faktoru a samotný faktor.

Také je možné zadat, jestli používat při výpočtech trik s aritmetikou nad zlogaritmovanými čísly. Je třeba, aby všechny vrcholy v grafickém modelu byly nastaveny stejně.

Parametry

- **`name`** – název proměnné.
- **`factor (Factor)`** – faktor reprezentující diskrétní pravděpodobnostní rozdělení.

`add_neighbor(node)`

Přidání souseda.

Parametr `node` – sousední vrchol.

`connect(node)`

Propojení dvou vrcholů.

Propojení dvou vrcholů, stačí zavolat pouze jednou pro dvojici vrcholů. Interně zavolá metodu `add_neighbor()` obou vrcholů a předá jí parametry.

Parametr `node` – sousední vrchol.

`init_messages()`

Inicializace všech zpráv mezi vrcholem a jeho sousedy.

`message_from(node, message)`

Přijmutí zprávy ze sousedního vrcholu.

Zpráva je uložena a použita při dalších výpočtech.

Parametry

- **node** – sousední vrchol.
- **message** – zpráva ze sousedního vrcholu.

message_to(*node*)

Poslání zprávy do sousedního vrcholu.

Parametr node – sousední vrchol, kterému má být zpráva zaslána.

normalize(*parents*)

Normalizace pravděpodobnostního rozdělení.

Při počítání nejsou zprávy normalizované, proto ani výsledek nemusí být normalizovaný a je tedy jej třeba normalizovat před jakýmkoliv dalším použitím. Pro normalizaci je možné zadat, které proměnné jsou rodiče, pak musí být suma ohodnocení všech přiřazení se stejnými rodiči rovna jedné.

Parametr parents – seznam proměnných, které jsou rodiči vrcholu.

update()

Aktualizace interní reprezentace.

Příchozí zprávy jsou pouze uloženy a je potřeba aktualizovat interní reprezentaci jejich součinu před počítáním odchozích zpráv.

Vrcholy pro Dirichletovské parametry

class alex.infer.node.DirichletParameterNode

Předek: **alex.infer.node.VariableNode**

Reprezentace parametrů Dirichletovského rozdělení.

alpha

Parametry dirichletovského rozdělení. Typu **Factor**.

__init__(*name*, *alpha*[, *aliases*])

Vytvoření nového vrcholu pro parametry Dirichletovského rozdělení.

Pro vytvoření je třeba jméno proměnné a parametry. Aliasy pro proměnné slouží k přejmenování proměnných ve zprávách, které přicházejí do vrcholu.

Parametry

- **name** – název proměnné.

- **alpha** (**Factor**) – parametry Dirichletovského rozdělení.
- **aliases** – zobrazení ze jmen proměnných, které se mohou objevit ve zprávě, do proměnných v parametru.

add_neighbor(*node*)

Přidání souseda.

Parametr node (**DirichletFactorNode**) – sousední vrchol. ■

connect(*node*, ***options*)

Propojení dvou vrcholů.

Propojení dvou vrcholů, stačí zavolat pouze jednou pro dvojici vrcholů.

Interně zavolá metodu **add_neighbor()** a předá jí parametry.

Parametr node (**DirichletFactorNode**) – sousední vrchol. ■

init_messages()

Inicializace všech zpráv mezi vrcholem a jeho sousedy.

message_from(*node*, *message*)

Přijmutí zprávy ze sousedního vrcholu.

Zpráva je uložena a použita při dalších výpočtech.

Parametry

- **node** – sousední vrchol.
- **message** – zpráva ze sousedního vrcholu.

message_to(*node*)

Poslání zprávy do sousedního vrcholu.

Parametr node – sousední vrchol, kterému má být zpráva zaslána.

class alex.infer.node.DirichletFactorNode

Předek: **alex.infer.node.FactorNode**

Diskrétní faktor, parametry jeho rozdělení ovšem nejsou pevně dány, ale pochází z vrcholu typu **DirichletParameterNode**.

__init__(*name*[, *aliases*])

Vytvoření nového vrcholu pro faktor.

Pro vytvoření je třeba jméno faktoru.

Alias pro proměnné slouží k přejmenování proměnných ve zprávách, které přicházejí do vrcholu.

Parametry

- **name** – název faktoru.
- **aliases** – zobrazení ze jmen proměnných, které se mohou objevit ve zprávě do proměnných ve faktoru.

add_neighbor(*node*)

Přidání souseda.

Parametr *node* – sousední vrchol.

connect(*node*)

Propojení dvou vrcholů.

Propojení dvou vrcholů, stačí zavolat pouze jednou pro dvojici vrcholů. Interně zavolá metodu **add_neighbor()** obou vrcholů a předá jí parametry.

Parametr *node* – sousední vrchol.

init_messages()

Inicializace všech zpráv mezi vrcholem a jeho sousedy.

message_from(*node*, *message*)

Přijmutí zprávy ze sousedního vrcholu.

Zpráva je uložena a použita při dalších výpočtech.

Parametry

- **node** – sousední vrchol.
- **message** – zpráva ze sousedního vrcholu.

message_to(*node*)

Poslání zprávy do sousedního vrcholu.

Parametr *node* – sousední vrchol, kterému má být zpráva zaslána.

update()

Aktualizace interní reprezentace.

Příchozí zprávy jsou pouze uloženy a je potřeba aktualizovat interní reprezentaci jejich součinu před počítáním odchozích zpráv.

B.1.3 Modul **lbp**

exception alex.infer.lbp.LBPErrror

Výjimka pro chyby při inferenci.

class alex.infer.lbp.LBP

Inference v grafickém modelu pomocí Loopy Belief Propagation.

__init__(*strategy='sequential'*)

Vytvoření nové instance LBP algoritmu.

Je možné nastavit strategii výběru vrcholů:

- *sequential* – vrcholy jsou vybírány v takovém pořadí, v jakém byly vloženy,
- *tree* – grafický model je ve formě stromu,
- *layers* – inference na Dynamické Bayesovské síti.

Parametr strategy – strategie, která má být použita pro výběr vrcholů.

add_nodes(*nodes*)

Přidání vrcholů k inferenci.

Parametr nodes – seznam vrcholů.

add_layer(*layer*)

Přidání vrstvy na konec grafického modelu.

Parametr layer – seznam vrcholů tvořících jednu vrstvu.

add_layers(*layers*)

Přidání vrstev na konec grafického modelu.

Parametr layers – seznam seznamů s vrcholy.

clear_nodes()

Smazání všech vrcholů.

clear_layers()

Smazání všech vrstev.

init_messages()

Inicializace zpráv pro všechny vrcholy.

run(*n_iterations=1, from_layer/*)

Spustit inferenci v grafickém modelu.

Je možné nastavit počet iterací, po kterém se má inference zastavit. U stromu není třeba zadávat počet iterací, bude provedena pouze jedna.

U vrstev je možné zadat od které vrstvy inferenci provádět. Popřípadě budou zprávy propagovány od začátku.

Parametry

- **n_iterations** – počet iterací.
- **from_layer** – vrstva, od které se mají zprávy propagovat.

Literatura

- [1] Arnborg, S.; Corneil, D. G.; Proskurowski, A.: Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, ročník 8, č. 2, 1987: s. 277–284.
- [2] Austin, J. L.: *How to do things with words*, ročník 1955. Oxford university press, 1975.
- [3] Bernardo, J. M.; Smith, A. F.: *Bayesian theory*, ročník 405. Wiley. com, 2009.
- [4] Bertoldi, N.; Federico, M.: A new decoder for spoken language translation based on confusion networks. In *Automatic Speech Recognition and Understanding, 2005 IEEE Workshop on*, IEEE, 2005, s. 86–91.
- [5] Bishop, C. M.; Nasrabadi, N. M.: *Pattern recognition and machine learning*, ročník 1. springer New York, 2006.
- [6] Black, A.; Taylor, P.; Caley, R.; aj.: The Festival Speech Synthesis System, Version 1.4. 2. *Unpublished document available via <http://www.cstr.ed.ac.uk/projects/festival.html>*, 2001.
- [7] Black, A. W.; Lenzo, K. A.: Flite: a small fast run-time synthesis engine. In *4th ISCA Tutorial and Research Workshop (ITRW) on Speech Synthesis*, 2001.
- [8] Brier, G. W.: Verification of forecasts expressed in terms of probability. *Monthly weather review*, ročník 78, č. 1, 1950: s. 1–3.
- [9] Crockford, D.: The application/json media type for javascript object notation (json). 2006.
- [10] He, Y.; Young, S.: Semantic processing using the hidden vector state model. *Computer speech & language*, ročník 19, č. 1, 2005: s. 85–106.
- [11] Juang, B. H.; Rabiner, L. R.: Hidden Markov models for speech recognition. *Technometrics*, ročník 33, č. 3, 1991: s. 251–272.
- [12] Koller, D.; Friedman, N.: *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
- [13] Kullback, S.: *Information theory and statistics*. Courier Dover Publications, 1997.
- [14] Mairesse, F.; Gasic, M.; Jurcicek, F.; aj.: Spoken language understanding from unaligned data using discriminative classification models. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, IEEE, 2009, s. 4749–4752.
- [15] Minka, T.; Winn, J.; Guiver, J.; aj.: Infer.NET 2.5. 2012, microsoft Research Cambridge. <http://research.microsoft.com/infernet>.

- [16] Minka, T. P.: Expectation propagation for approximate Bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 2001, s. 362–369.
- [17] Murphy, K. P.; Weiss, Y.; Jordan, M. I.: Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 1999, s. 467–475.
- [18] Oliphant, T. E.: *Guide to NumPy*. Provo, UT, Březen 2006.
URL <http://www.tramy.us/>
- [19] Patil, A.; Huard, D.; Fonnesbeck, C. J.: PyMC: Bayesian stochastic modeling in Python. *Journal of statistical software*, ročník 35, č. 4, 2010: str. 1.
- [20] Pearl, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Pub, 1988.
- [21] Povey, D.; Ghoshal, A.; Boulianne, G.; aj.: The Kaldi Speech Recognition Toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, IEEE Signal Processing Society, Prosinec 2011, iEEE Catalog No.: CFP11SRW-USB.
- [22] Raux, A.; Bohus, D.; Langner, B.; aj.: Doing research on a deployed spoken dialogue system: One year of Let’s Go! experience. In *Proc. Interspeech*, 2006, s. 65–68.
- [23] Raux, A.; Langner, B.; Bohus, D.; aj.: Let’s go public! taking a spoken dialog system to the real world. In *in Proc. of Interspeech 2005*, Citeseer, 2005.
- [24] SpeechTech s.r.o.: SpeechTech TTS - syntéza řeči. 2013.
URL <http://www.speechtech.cz/cs/produkty/synteza-reci.html>
- [25] Tatikonda, S. C.; Jordan, M. I.: Loopy belief propagation and Gibbs measures. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 2002, s. 493–500.
- [26] Thomson, B.; Schatzmann, J.; Young, S.: Bayesian update of dialogue state for robust dialogue systems. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, IEEE, 2008, s. 4937–4940.
- [27] Thomson, B.; Young, S.: Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems. *Computer Speech & Language*, ročník 24, č. 4, 2010: s. 562–588.
- [28] Traum, D. R.: Speech acts for dialogue agents. In *Foundations of rational agency*, Springer, 1999, s. 169–201.
- [29] Walker, W.; Lamere, P.; Kwok, P.; aj.: Sphinx-4: A flexible open source framework for speech recognition. 2004.

- [30] Weizenbaum, J.: ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, ročník 9, č. 1, 1966: s. 36–45.
- [31] Williams, J.; Raux, A.; Ramachandran, D.; aj.: The Dialog State Tracking Challenge. In *14th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, Metz, France, 2013.
- [32] Williams, J. D.; Raux, A.; Ramachandran, D.; aj.: Dialog state tracking challenge handbook.
- [33] Williams, J. D.; Young, S.: Partially observable Markov decision processes for spoken dialog systems. *Computer Speech & Language*, ročník 21, č. 2, 2007: s. 393–422.
- [34] Wong, Y. W.; Mooney, R.: Learning synchronous grammars for semantic parsing with lambda calculus. In *ANNUAL MEETING-ASSOCIATION FOR COMPUTATIONAL LINGUISTICS*, ročník 45, 2007, str. 960.
- [35] Young, S.; Evermann, G.; Gales, M.; aj.: The HTK book. *Cambridge University Engineering Department*, ročník 3, 2002.
- [36] Young, S.; Gašić, M.; Keizer, S.; aj.: The Hidden Information State Model: a practical framework for POMDP-based spoken dialogue management. *Computer Speech & Language*, ročník 24, č. 2, 2010: s. 150–174.
- [37] Zen, H.; Nose, T.; Yamagishi, J.; aj.: The HMM-based speech synthesis system (HTS) version 2.0. In *Proc. of Sixth ISCA Workshop on Speech Synthesis*, 2007, s. 294–299.
- [38] Zettlemoyer, L. S.; Collins, M.: Online learning of relaxed CCG grammars for parsing to logical form. In *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL-2007)*, Citeseer, 2007.
- [39] Žilka, L.; Marek, D.; Korvas, M.; aj.: Comparison of Bayesian Discriminative and Generative Models for Dialogue State Tracking. In *14th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, Metz, France, 2013.

Přílohy