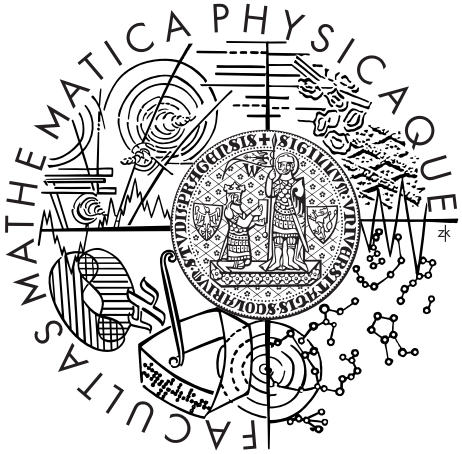
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

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:	program
Studijní obor:	obor

Praha 2013

Poděkování.

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

Katedra: Ústav formální a aplikované lingvistiky

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

Abstrakt:

Klíčová slova:

Title:

Author: David Marek

Department: Institute of Formal and Applied Linguistics

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

Abstract:

Keywords:

Obsah

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 vnikaly programy, které se snažily využívat přirozenou řeč pro interakci s uživatelem. Jedním z takových programů byl například Eliza, program, který předstíral, že jej zajímá, co mu uživatel říká. 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. S využitím statistickým metod a metod strojového učení bylo možné začít s porozumíváním mluveného slova. Přijetí bylo zpočátku chladné a veřejnost byla

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, který bude 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 systémy, které se účastnily soutěže Dialog State Tracking Challenge 2013.

4. Důležitou části algoritmů pro inferenci je určení pořadí, v jakém má inference probíhat. Práce bude obsahovat implementaci několika strategií pro inferenci, které budou umožňovat efektivní inferenci pro různé druhy bayesovských sítí (stromy, dynamické sítě, obecné grafy).

5. Nakonec se práce bude zabývat učením parametrů sítě a představí algoritmus Expectation Propagation, který umožňuje inferenci v bayesovských sítích se spojitými náhodnými proměnnými.

6. Při většině reálných použití dochází k aproximacím a úpravám modelu v závislosti na problému tak, aby bylo možné inferenci provádět v reál-

ném čase. Práce bude obsahovat implementaci frameworku, do kterého je možné jednoduše zasadit vlastní moduly pro aproximaci pravděpodobnostních rozdělení, které bude algoritmus Expectation Propagation používat.

7. Jako příklad bude ukázán systém pro učení parametrů pravděpodobnostního rozložení pro pozorování v generickém modelu pro reprezentaci dialogového stavu.

1. Teorie dialogových systémů

1.1 Dialogový systém

Dialogový systém je počítačový systém, který umožňuje uživatelům komunikovat s počítačem ve formě, která je přirozená a efektivní pro použití. Dialogové systémy stále mají spoustu problémů k překonání a pro praktické použití je třeba se uchýlit k několika 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. Vždy je potřeba při vývoji dialogového systému mít ontologii určující, jaké informace má systém poskytovat a o čem se může chtít uživatel bavit.

Další zjednodušení se týkají přímo dialogu. Předpokládá se, že dialog probíhá vždy 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. Příkazu od uživatele může vypadat např. takto: „Chci jet z Malostranského náměstí na Anděl“. Dialogový systém se nyní může rozhodnout, zda-li mu zadané informace stačí pro nalezení spojení. V tomto případě systém stále neví, kdy chce uživatel jet. Může předpokládat, že uživatel už na zastávce stojí a tak tedy uživateli nalezne nejbližší spojení.

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

1.2 Součásti dialogového systému

Dialogový systém se skládá z několika částí, které spolu komunikují. 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 dialogový manager (DM), který zvolí patřičnou odpověď. Výstupem dialogového manageru jsou informace, které se mají předat uživateli. 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. Po získání textové podoby je teprve 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. Pro tuto část dialogového systému existuje řada dostupných otevřených toolkitů, např. systém HTK [25], Kaldi [16] nebo SPHINX [22]. Existuje i celá řada komerčního software od firem jako IBM nebo Nuance.

Úspěšnost systému rozpoznání řeči je závislá na obtížnosti úlohy a na počtu trénovacích dat, pocházejících ze stejné domény. Pro obecnou doménu se problém stává mnohem těžší a je třeba velké množství dat. Word error rate (WER) je častá metrika pro počítání výkonu ASR. Pro spočítání WER je třeba nejprve provést zarovnání rozpoznaného a originálního textu. WER je pak počet slov, která jsou změněná, smazaná anebo přidaná, vydělený počtem slov v originálním textu. Systém Let's Go! [17] 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, z kterých by mohl pocházet. Reprezentace možných hypotéz může být seznam slovních sekvencí s jejich věrohodností. Věrohodnosti jsou skóre přiřazené hypotézám, které určují jakou důvěru má systém rozpoznávání řeči ve správnost dané slovní sekvence. V ideálním případě je věrohodnost ekvivalentní aposteriorní pravděpodobnosti sekvence slov, dáno vstupní zvuk. Ovšem ne všechny rozpoznávače pracují na pravděpodobnostním principu a pak není možné od nich požadovat skutečné pravděpodobnosti.

Přidat příklad seznamu hypotéz

Další možností jak reprezentovat výstup je použití konfůzní sítě [3]. Konfůzní 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 konfůzní sítě je, že umožňuje v komprimované podobě uložit mnohem více hypotéz.

Přidat příklad konfůzní sítě

1.2.2 Porozumění mluvené řeči (SLU)

Jakmile má systém seznam možných hypotéz toho, co uživatel řekl, musí se pokusit porozumět, co tím uživatel myslel. Dialogový systém nepotřebuje vědět, co přesně uživatel řekl, důležité je pouze zjistit, co se uživatel snaží sdělit. Pokud například uživatel řekne "Chtěl bych nalézt spojení z Malostranského náměstí na Anděl", anebo "Jak se dostanu na Anděl ze zastávky Malostranské náměstí?", tak

výsledek je stejný, uživatel požaduje informace o spojení mezi dvěma zastávkami, i když v jednom případě jde o větu oznamovací a v druhém případě o otázku.

Semántická reprezentace sdělení uživatele se nazývá dialogový akt (DA), skládá se z jedné nebo více položek dialogového aktu (DAI), které jsou spojené 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 z dialogového systému pro hledání restaurací:

```
hello()&inform(food="chinese").
```

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 *food* a hodnota je „chinese“, tedy uživatel nám říká, že hledá restauraci, kde servírují čínské jídlo.

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 buď „Ano“ nebo „Ne“, U implicitního, pokud se nám nedostane odpovědi předpokládáme, že protějšek souhlasí,
- *select* — žádáme protějšek, aby zvolil z nabízených možností.

Existuje široké množství technik, které lze použít pro porozumění mluvené řeči. Unifikace pomocí šablon anebo gramatiky jsou příklady ručně psaných metod. Metody založené na datech jsou například Hidden Vector State model [8], techniky strojového překladu [24], Combinatory Categorical Grammars [28] nebo Support Vector Machines [11].

1.2.3 Dialogový manager (DM)

Pokud už jsou pravděpodobné dialogové akty dekódovány, je třeba rozhodnout, co bude systém dělat. Komponenta tvořící tato rozhodnutí je dialogový manager. Odpověď systému je zakódována do formy dialogových aktů a nazývá se systémová akce.

Zvolená systémová akce je vybrána z množiny možných akci $a \in \mathcal{A}$ a závisí na vstupu, který systém obdržel z SLU. Tento vstup se nazývá pozorování $o \in \mathcal{O}$, protože obsahuje vše, co systém pozoroval o uživateli.

Zvolení správné akce potřebuje více znalostí než jen poslední pozorování. Celá historie dialogu a také kontext hrají důležitou roli. Dialogový manager bere na vše ohled pomocí udržovaní interní reprezentace celého pozorovaného dialogu. Tato reprezentace se nazývá dialogový stav, nebo také belief stav, značí se $b \in \mathcal{B}$. Aktuální dialogový stav závisí na přechodové funkci, která dialogový stav aktualizuje pro každé nové pozorování a systémovou akci. Přechodová funkce je tedy mapování $\mathcal{T} : \mathcal{B} \times \mathcal{A} \times \mathcal{O} \longrightarrow \mathcal{B}$. V této práci se budeme věnovat právě metodám aktualizace dialogového stavu.

Chování dialogového stavu definuje dialogová strategie π. Strategie určuje co má systém provést v závislosti na aktuálním dialogovém stavu. Obecně strategie vytvoří pravděpodobnostní rozložení přes možné akce. Pokud $\prod(\mathcal{A})$ značí množinu těchto distribucí, pak dialogová strategie bude zobrazení z dialogového stavu do této množiny, $\pi : \mathcal{B} \longrightarrow \prod(\mathcal{A})$.

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

1.2.4 Generování přirozené řeči (NLG a TTS)

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

Nejjednodušším přístup ke generování přirozeného jazyka z dialogových aktů je použití šablon. Například pro dialogový akt `inform(type="x")` bude vytvořena šablona „Restaurace servíruje x jídlo“, kde „x“ bude nahrazeno například za „čínské“, „indické“, atd. Šablony se při generování osvědčily, protože počet možných dialogových je většinou zvládnutelný.

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 [5] nebo FLite [6].

Alternativní metodou syntézy je použití skrytých Markovských modelů pro generování zvuku, příkladem je HTS systém [27].

1.3 Dialogový stav

Nejistota je základním problémem, s kterým se dialogový systém musí vypořádat. Systémy pro rozpoznávání i porozumění řeči často chybují a tuto možnost musí brát dialogový manager 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 a je možné je odvodit z pozorování. Pravděpodobnostní rozložení těchto náhodných proměnných dává dobře definovanou reprezentaci nejistoty, navíc je možné je reprezentovat pomocí Bayesovské sítě.

Jedním z možných modelů dialogového stavu je generativní model. Definujeme množinu stavů prostředí, $s \in \mathcal{S}$. Předpokládáme, že pozorování závisí podmíněně pouze na stavu prostředí a definujeme pravděpodobnostní rozdělení pro pozorování, $p(o \mid s)$. Dále předpokládáme, že cíle uživatele se nemění v čase a stav prostředí je tedy 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 stav prostředí závisí pouze na minulé hodnotě se nazývá Markovská vlastnost.

Pokud vezmeme předchozí předpoklady, tak lze využít bayesovský přístup pro počítání s nejistotou. Stav v čase t označíme s_t. Podle Bayesova vzorce můžeme spočítat pravděpodobnost stavu v čase $t+1$ po přijetí nového pozorování $o_{t+1} = o'$.

$$p(s_{t+1} = s') \propto \sum_{s \in \mathcal{S}} p(s_t = s) p(s_{t+1} = s' \mid s_t = s, a_t = a) p(o_{t+1} = o' \mid s_{t+1} = s') \tag{1.1}$$ {eq:belief

Nyní můžeme definovat dialogový stav v čase t, b_t, jako pravděpodobnost přes stavy dáno všechna pozorování až do času t. Množina všech možných dialogových stavů je pravděpodobnost přes všechny možné stavy prostředí $\mathcal{B} = \prod(\mathcal{S})$.

Můžeme přepsat rovnici (1.1) s pomocí dialogových stavů.

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

Rovnice (1.2) nám dává předpis pro přechodovou funkci $\mathcal{T}$. V praxi ovšem bude množina možných hodnot pro slot 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 slot je jedním možným cílem uživatele. Tedy velikost stavového prostoru roste exponenciálně.

1.3.1 Aktualizace dialogového stavu

Efektivní metodou pro aktualizaci dialogového stavu je použití dynamických bayesovských sítí [20]. Bayesovské sítě umoňují efektivní výpočet 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 [21].

Alternativním zjednodušením je rozdělit stav prostředí do skupin. Tento přístup se nazývá Hidden Information State (HIS) [26]. 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 sítích. 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 rozložení.

2. Bayesovské sítě a inference

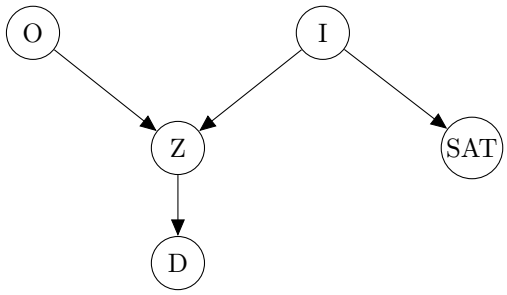
V této kapitole představíme Bayesovské sítě, grafický model pro efektivní reprezentaci pravděpodobnostních rozdělení a nezávislostí mezi náhodnými proměnnými. Bayesovská síť zároveň vytváří koncept pro inferenci, tedy zodpovídání dotazů nad proměnnými v síti. Ukážeme si několik přístupů k inferenci, nejprve naivní výpočet vycházející 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. Nakonec se dostaneme k metodě Expectation Propagation, která je zobecněním LBP a je tedy možné ji použít pro libovolné rozdělení. Díky ní budeme schopní vytvořit generativní model pro aktualizaci dialogového stavu, který bude pracovat stejně jako LBP, ale bude schopen adaptace.

2.1 Bayesovské sítě

Bayesovské sítě jsou pravděpodobnostní grafický model, který využívá podmíněných nezávislostí pro úspornou reprezentaci sdružené pravděpodobnosti. Bayesovská sít je orientovaný acyklický graf, jeho 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 její rodiče, $P(X \mid parents(X))$. Pokud proměnná nemá žádná rodiče, pak její podmíněná pravděpodobnostní distribuce je ekvivalentní marginální pravděpodobnostní distribuci.

Příklad Student [9]: 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 ale nemusí stačit pro správné zhodnocení inteligence. Požadují tak tedy i doporučení (D) od jednoho z učitelů. Učitel studentovi napíše doporučující dopis na základě známky (Z), kterou student získal v jeho předmětu. Předměty se ovšem liší v obtížnosti (O) a tak je studentova známka v předmětu závislá nejen na jeho inteligenci, ale také na obtížnosti předmětu. Grafický model reprezentující tento problém je vyobrazen na obrázku 2.1.



{fig:stude

Obrázek 2.1: Bayesovská síť pro příklad se studentem.

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 studentova inteligence. Konečně doporučení, které student obdrží, je podmíněně nezávislé na všech ostatních proměnných, dáno studentova známka.

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

$$P(O, I, Z, S, D) = P(D \mid Z)P(Z \mid O, I)P(SAT \mid I)P(O)P(I) \tag{2.1}$$

Předpokládejme, že obtížnost předmětu, inteligence studenta, doporučující dopis a výsledek SAT jsou binární proměnné. Známka z předmětu pak je ternární proměnná. Pokud bychom zapsali sdruženou pravděpodobnost ve formě tabulky, tak se dostaneme k 48 položkám. Díky rozdělení do podmíněných pravděpodobnostních rozložení, které nám bayesovská síť poskytuje, se dostáváme k $2 + 2 + 12 + 4 + 6 = 26$ položkám. Tedy i v tomto jednoduchém modelu dochází k 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, pokud vezmeme příklad se studentem a budeme chtít znát marginální pravděpodobnost známek, musíme vysčítat všechny ostatní proměnné.

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

Další a asi nejčastější dotaz nastává, pokud některé náhodné proměnné pozorujeme. Pak chceme znát pravděpodobnost jiných proměnných dáno naše po-

zorování, $P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$, kde $\boldsymbol{X}$ jsou dotazované proměnné, $\boldsymbol{E}$ jsou pozorované proměnné a $\boldsymbol{e}$ jsou pozorované hodnoty. Z definice podmíněné pravděpodobnosti dostáváme

$$P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e}) = \frac{P(\boldsymbol{X}, \boldsymbol{E} = \boldsymbol{e})}{P(\boldsymbol{E} = \boldsymbol{e})} \tag{2.3}$$

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

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

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

$$P(\boldsymbol{E} = \boldsymbol{e}) = \sum_{\boldsymbol{x}} P(\boldsymbol{x}, \boldsymbol{e}) \tag{2.5}$$

a tedy můžeme použít už vypočítané hodnoty.

2.2.1 Exaktní inference

V předchozí části jsme viděli, že pomocí definice podmíněné pravděpodobnosti a marginalizace sdružené pravděpodobnosti lze 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 $A \to B \to C \to D$. Sdružená pravděpodobnost $P(A, B, C, D)$ je součinem jednotlivých podmíněných pravděpodobnostních distribucí

$$P(A, B, C, D) = P(D \mid C)P(C \mid B)P(B \mid A)P(A) \tag{2.6}$$

Pokud nyní budeme chtít spočítat marginální distribuci D, tak musíme marginalizovat všechny ostatní proměnné

$$P(D) = \sum_{A,B,C} P(D \mid C)P(C \mid B)P(B \mid A)P(A) \tag{2.7}$$

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

$$P(D) = \sum_C P(D \mid C) \sum_B P(C \mid B) \sum_A P(B \mid A) P(A) \tag{2.8}$$

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

$$\psi_2(B, C) = P(C \mid B)\tau_1(B) \tag{2.9}$$

$$\tau_2(C) = \sum_B \psi_2(B, C) \tag{2.10}$$

A nakonec spočítáme finální marginální pravděpodobnost

$$\psi_3(D, C) = P(D \mid C)\tau_2(C) \tag{2.11}$$

$$P(D) = \sum_C \psi_3(D, C) \tag{2.12}$$

Definice 1. *Nechť $\mathcal{X}$ je množina náhodných proměnných. Potom definujeme faktor ϕ jako zobrazení z $Val(\mathcal{X})$ do $\mathbb{R}$. Faktor je nezáporný, pokud všechny jeho obrazy jsou nezáporné. Množina proměnných $\mathcal{X}$ je doménou faktoru a značíme ji 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.

Definice 2. *Nechť $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ jsou tři disjunktní množiny náhodných proměnných. Nechť $\phi_1(\boldsymbol{X}, \boldsymbol{Y})$ a $\phi_2(\boldsymbol{Y}, \boldsymbol{Z})$ jsou faktory. Definujeme součin faktorů $\phi_1 \times \phi_2$ jako faktor $\psi : Val(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) \to \mathbb{R}$ následovně:*

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

Násobíme prvky, které mají stejné ohodnocení společných proměnných $\boldsymbol{Y}$. Stejný princip použijeme pro všechny matematické operace.

Definice 3. *Nechť $\boldsymbol{X}$ je množina náhodných proměnných a $Y \notin \boldsymbol{X}$ náhodná proměnná. Nechť $\phi(\boldsymbol{X}, Y)$ je faktor. Definujeme marginalizaci Y v ϕ, značenou $\sum_Y \phi$, jako faktor ψ s doménou $\boldsymbol{X}$ takový, že*

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

Této operaci také říkáme vysčítání Y ve ϕ.

Faktory se při počítání chovají jako čísla, všechny operace 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 $X \notin Dom(\phi_1)$, 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.

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

Opět se pokusíme spočítat marginální pravděpodobnost proměnné D.

$$P(D) = \sum_C \sum_B \sum_A \phi_A \phi_B \phi_C \tag{2.14}$$

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

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, který je 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 je třeba nejprve vynásobit všechny faktory, které ji obsahují a následně ji vysčítat. Tak získáme faktor, který už tuto proměnnou neobsahuje, tedy jsme ji eliminovali. Eliminace proměnných je popsána v algoritmu 1.

Algoritmus 1 Eliminace proměnných {alg:ve}

function SUM-PRODUCT-VE(Φ, $\boldsymbol{X}$, $\prec$)
  Φ množina všech faktorů.
  $\boldsymbol{X}$ množina náhodných proměnných, které mají být eliminovaný.
  $\prec$ pořadí proměnných, v jakém mají být eliminovány.

  Nechť $X_1, \ldots, X_k$ je seřazení proměnných z $\boldsymbol{X}$, t.ž. $X_i \prec X_j \Leftrightarrow i < j$.
  for $i = 1 \ldots k$ **do**
    $\Phi \leftarrow$ Sum-Product-Eliminate-Var(Φ, Z_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.

  $\Phi' \leftarrow \{\phi \in \Phi : X \in Dom(\phi)\}$
  $\Phi'' \leftarrow \Phi - \Phi'$
  $\psi \leftarrow \prod_{\phi \in \Phi'} \phi$
  $\tau \leftarrow \sum_X \psi$
  return $\Psi'' \bigcup \{\tau\}$
end function

Věta 1. *Nechť $\boldsymbol{X}$ je množina náhodných proměnných, nechť Φ je množina faktorů, t.ž. pro každé $\phi \in \Phi$, $Dom(\phi) \subseteq \boldsymbol{X}$. Nechť $\boldsymbol{Y} \subset \boldsymbol{X}$ je množina dotazovaných náhodných proměnných a nechť $\boldsymbol{Z} = \boldsymbol{X} - \boldsymbol{Y}$. Pak pro každé seřazení $\prec$ nad $\boldsymbol{Z}$, Sum-Product-VE(Φ, $\boldsymbol{Z}$, $\prec$) vrátí faktor $\phi^*(\boldsymbol{Y})$ takový, že*

$$\phi^*(\boldsymbol{Y}) = \sum_Z \prod_{\phi \in \Phi} \phi$$

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, při kterých je vždy eliminována jedna proměnná.

Při jednom eliminačním kroku je vybrána proměnná X_i, všechny faktory, které ji obsahují jsou pronásobeny 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, kde

m je počet faktorů, které vznikly vysčítáním proměnné. Každý z těchto faktorů je zahrnut do součinu pouze jednou, pokud 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 $\mathcal{O}(nN_{max})$.

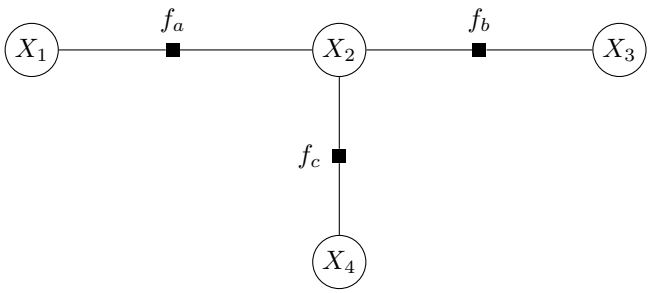
Pokud k je maximum z velikostí domén proměnných, pak velikost faktoru obsahujícího n proměnných může být až k^n. Složitost eliminace proměnných je tedy dominována velikostí faktorů, které vznikají při výpočtu a je exponenciální. Navíc bylo dokázáno, že výběr nejlepšího pořadí proměnných pro eliminaci 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 faktorizace sdružené pravděpodobnosti.

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

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, který ji obsahuje.



Obrázek 2.2: Příklad faktor grafu se třemi náhodnými proměnnými X_1, X_2, X_3 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

$$P(\boldsymbol{X}) = \prod_{s \in ne(X)} F_s(X, \boldsymbol{X}_s), \tag{2.16}$$ {eq:fs}

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 $\boldsymbol{X}_s$ 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

$$P(X) = \prod_{s \in ne(X_i)} \sum_{\boldsymbol{X}_s} F_s(X, \boldsymbol{X}_s) \tag{2.17}$$

$$= \prod_{s \in ne(X)} \mu_{f_s \to X}(X) \tag{2.18}$$ {eq:margx}

Zavedli jsme funkce

$$\mu_{f_s \to X} \equiv \sum_{\boldsymbol{X}_s} F_s(X, \boldsymbol{X}_s), \tag{2.19}$$ {eq:defmsg}

které můžeme nazývat zprávami z faktoru f_s do proměnné X.

Každé $F_s(X, \boldsymbol{X}_s)$ je popsáno faktor grafem 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 $X_1, \ldots, X_M$. Součin faktorů v podstromech určených těmito proměnnými označíme $G_i(X_i, \boldsymbol{X}_{si})$. Faktor $F_s(X, \boldsymbol{X}_s)$ tedy můžeme přepsat jako

$$F_s(X, \boldsymbol{X}_s) = f_s(X, X_1, \ldots, X_M) G_1(X_1, \boldsymbol{X}_{s1}), \ldots, G_M(X_M, \boldsymbol{X}_{sM}) \tag{2.20}$$

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

$$\mu_{f_s \to X}(X) = \sum_{X_1} \cdots \sum_{X_M} f_s(X, X_1, \ldots, X_M) \prod_{m \in ne(f_s) \setminus X} \sum_{\boldsymbol{X}_{sm}} G_m(X_m, \boldsymbol{X}_{sm}) \tag{2.21}$$

$$= \sum_{X_1} \cdots \sum_{X_M} f_s(X, X_1, \ldots, X_M) \prod_{m \in ne(f_s) \setminus X} \mu_{X_m \to f_s}(X_m) \tag{2.22}$$ {eq:mfsx}

Zavedli jsme další funkce

$$\mu_{X_m \to f_s}(X_m) \equiv \sum_{\boldsymbol{X}_{sm}} G_m(X_m, \boldsymbol{X}_{sm}), \tag{2.23}$$ {eq:defmsg}

které budeme nazývat zprávami z proměnné X_m do faktoru f_s.

Z rovnice 2.18 vidíme, že marginální pravděpodobnost proměnné vypočítá-

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

Stejně jako u $F_s(X, \boldsymbol{X}_s)$ nám tady $G_m(X_m, \boldsymbol{X}_{sm})$ 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 $G_m(X_m, \boldsymbol{X}_{sm})$, a pak

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

Přepsáno ve formě zpráv

$$\mu_{X_m \to f_s}(X_m) = \prod_{l \in ne(X_m) \setminus f_s} \sum_{\boldsymbol{X}_{ml}} F_l(X_m, \boldsymbol{X}_{ml}) \tag{2.25}$$

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

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

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

$$\mu_{f \to X}(X) = f(X) \tag{2.27}$$

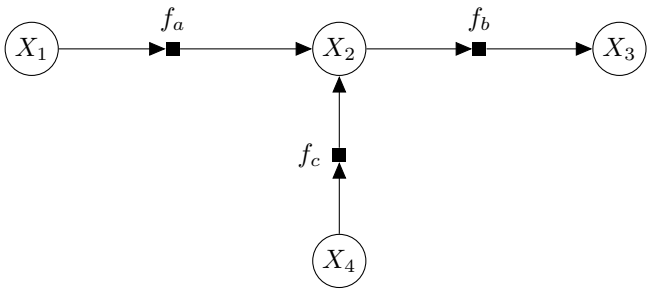
V případě náhodné proměnné je odchozí zpráva

$$\mu_{X \to f}(X) = 1 \tag{2.28}$$

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

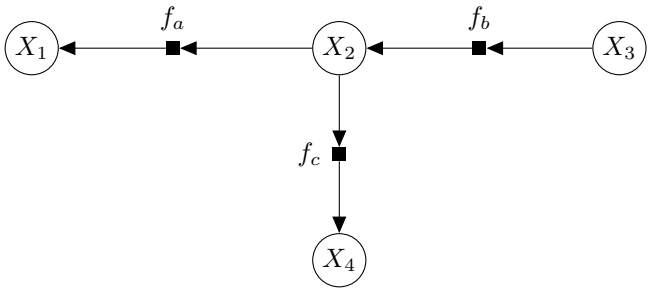
2.2.3 Belief Propagation

Algoritmus exaktní inference na stromech s pomocí posílání zpráv se nazývá belief propagation [15]. 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, tak lze spočítat jeho marginální pravděpodobnost. Strom vždy můžeme zakořenit v nějaké 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 linárním 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. Hlavní důvod proč nelze použít Belief Propagation na obecných grafech je, že můžeme narazit na cyklus v grafu, což znamená, že 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 není třeba znát příchozí zprávy ze všech sousedních vrcholů, kromě toho, kterému je zpráva určena. Chybějící zprávy jsou nastaveny na jedničku. Loopy Belief Propagation je iterativní algoritmus. V grafu nemusí existovat vrchol, který může posílat zprávu k nějakému ze svých sousedů. Je tedy třeba vybrat vrchol podle nějaké strategie a z něj poslat zprávy do všech sousedních vrcholů. V další iteraci je pak vybrán zase jiný vrchol, který bude posílat zprávy. Iterace končí ve chvíli, kdy se už nemění marginální pravděpodobnosti proměnných.

Vzhledem k tomu, že posílané zprávy už neodpovídají skutečným faktorům, je důležitá otázka, zda-li Loopy Belief Propagation vůbec nalezne správné marginální pravděpodobnosti. Existují podmínky za kterých algoritmus bude konvergovat [19]. V obecném případě ovšem 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 [13]. 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 nedokonvergují, anebo můžeme nastavit pevný počet iterací. Pak vybíráme vrcholy a z každého pošleme zprávu do všech okolních. Způsobem výběru vrcholů se budeme zabývat později. I kdybychom vybírali vrcholy v náhodném pořadí, tak se můžeme dobrat k výsledku, pouze to bude nejspíš trvat déle.

Poslání zprávy se liší podle toho, zda 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 zprávy vynásobíme faktorem a marginalizujeme všechny proměnné, kromě té, které zprávu posíláme.

Ve faktor grafu se tedy zprávy šíří pouze přes pravděpodobnostní rozdělení jedné proměnné. To může právě vést k oscilacím anebo konvergenci ke špatným hodnotám. Příkladem může být 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 {alg:lbp}

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**
          $\mu_{v\to n}(n) \leftarrow$ MESSAGE-TO-VAR(v, n)
        else
          $\mu_{v\to n}(v) \leftarrow$ MESSAGE-TO-FACTOR(v, n)
        end if
      end for
    end for
  until konvergence
  for každá proměnná X **do**
    $P(X) = \prod_{f\in ne(X)} \mu_{f\to X}(X)$
  end for
end function

function INIT-FACTOR-GRAPH(F)
  F – faktor graf

  for každou proměnnou $X \in F$ **do**
    for sousední faktor f **do**
      $\mu_{X\to f}(X) \leftarrow 1$
      $\mu_{f\to X}(X) \leftarrow 1$
    end for
  end for
end function

function MESSAGE-TO-VAR(f, v)
  f – zdrojový faktor
  v – cílová proměnná

  $\boldsymbol{X} \leftarrow \mathrm{Dom}(f)$
  $\mu_{f\to v}(v) \leftarrow \sum_{\boldsymbol{X}\setminus v} f(\boldsymbol{X}) \prod_{u\in ne(f)\setminus v} \mu_{u\to f}(u)$
end function

function MESSAGE-TO-FACTOR(v, f)
  v – zdrojová proměnná
  f – cílový faktor

  $\mu_{v\to f}(v) \leftarrow \prod_{g\in ne(v)\setminus f} \mu_{g\to v}(v)$
end function

si pamatovat součin všech jeho zpráv

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

Pak můžeme při odesílání zprávy z vrcholu v do vrcholu u pouze $b(v)$ vydělit zprávou $\mu_{u \to v}$, pro faktory marginalizovat, a zpráva je připravena k odeslání. $b(v)$ můžeme aktualizovat po výběru vrcholu v pro posílání zpráv, anebo po každém přijetí zprávy.

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

2.3 Propagace s aproximovanými zprávami

V předchozí sekci byla faktorizována sdružená pravděpodobnost a tak došlo k zjednodušení inference. Mezi jednotlivými vrcholy byly stále posílány posílány exaktní zprávy. V této sekci se zaměříme na inferenci v modelech, kde není možné spočítat zprávy exaktně a je třeba je aproximovat. Příkladem jsou například modely se spojitými náhodnými proměnnými.

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

$$P(\boldsymbol{X}, \boldsymbol{E}) = \prod_i f_i(\boldsymbol{X}) \tag{2.30}$$

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

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

a pravděpodobnost pozorování je dána

$$P(\boldsymbol{E}) = \int \prod_i f_i(\boldsymbol{X}) \mathrm{d}\boldsymbol{X} \tag{2.32}$$

Pro aproximativní inferenci ukážeme algoritmus Expectation propagation (EP) [12]. Vytvoříme aproximaci aposteriorní distribuce, která je také dána součinem faktorů

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

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) [10]. KL divergence, také známá jako relativní entropie, mezi dvěma pravděpodobnostními rozděleními $p(x)$ a $q(x)$ je

$$KL(p\|q) = \int p(x)\frac{p(x)}{q(x)}\mathrm{d}x. \tag{2.34}$$

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 [2] přes $\boldsymbol{X}$, dáno parametry $\boldsymbol{\eta}$ je definována jako množina distribucí ve tvaru

$$P(\boldsymbol{X} \mid \boldsymbol{\eta}) = h(\boldsymbol{X})g(\boldsymbol{\eta})\exp\left(\boldsymbol{\eta}^T u(\boldsymbol{X})\right), \tag{2.35}$$ {eq:gj}

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

$$g(\boldsymbol{\eta})\int h(\boldsymbol{X})\exp\left(\boldsymbol{\eta}^T u(\boldsymbol{X})\right)\mathrm{d}\boldsymbol{X} = 1 \tag{2.36}$$ {eq:g1}

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

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

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

$$-\nabla\log g(\boldsymbol{\eta}) = \mathbb{E}_{p(\boldsymbol{X})}[u(\boldsymbol{X})]. \tag{2.38}$$ {eq:kl}

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

podle $\boldsymbol{\eta}$. Získáme

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

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

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

Ve výsledku získáváme

$$-\nabla \log g(\boldsymbol{\eta}) = \mathbb{E}[u(\boldsymbol{X})] \tag{2.41}$$ {eq:ge}

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

$$\mathbb{E}_{q(\boldsymbol{X})}[u(\boldsymbol{X})] = \mathbb{E}_{p(\boldsymbol{X})}[u(\boldsymbol{X})], \tag{2.42}$$ {eq:ee}

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

2.3.2 Expectation Propagation

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

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

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.

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

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

Další možností je aproximovat jednotlivé faktory, díky tomu bychom v jednom kroku nalezli 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 [4] sice aproximuje jednotlivé faktory, ale vždy v kontextu všech ostatních. 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 sekcí. Předpokládejme, že chceme aktualizovat faktor $\tilde{f}_j(\boldsymbol{X})$. Nejprve odstraníme faktor z produktu $\prod_{i \neq j} \tilde{f}_i(\boldsymbol{X})$. Následně nalezneme novou hodnotu pro faktor $\tilde{f}_j$ tak, aby pravděpodobnostní rozložení

$$q^{new}(\boldsymbol{X}) \propto \tilde{f}_j(\boldsymbol{X}) \prod_{i \neq j} \tilde{f}_i(\boldsymbol{X}) \tag{2.45}$$ {eq:qnew}

bylo co nejblíže

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

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

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

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

Mohli bychom počítat $q^{\setminus j}(\boldsymbol{X})$ jako součin všech faktorů kromě j, ale v praxi je dělení rychlejší. Této nenormalizované distribuci se říká cavity distribuce. Její kombinací se skutečným faktorem $f_j(\boldsymbol{X})$ dostaneme distribuci

$$\frac{1}{Z_j} f_j(\boldsymbol{X}) q^{\setminus j}(\boldsymbol{X}), \tag{2.48}$$ {eq:aprox}

kde Z_j je normalizační konstanta, dána

$$Z_j = \int f_j(\boldsymbol{X}) q^{\setminus j}(\boldsymbol{X}) \mathrm{d}\boldsymbol{X}. \tag{2.49}$$ {eq:zj}

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

$$KL\left(\frac{f_j(\boldsymbol{X}) q^{\setminus j}(\boldsymbol{X})}{Z_j} \middle\| q^{new}(\boldsymbol{X}) \right). \tag{2.50}$$

To uděláme jednoduše použitím poznatku z (2.42), který říká, že nám stačí najít parametry q^{new} tak, aby jeho 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

$$\tilde{f}_j(\boldsymbol{X}) = K\frac{q^{new}(\boldsymbol{X})}{q^{\setminus j}(\boldsymbol{X})} \tag{2.51}$$ {eq:newfj}

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

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

použili jsme navíc toho, že $q^{new}(\boldsymbol{X})$ je normalizovaná distribuce. Hodnota K pak může být nalezena srovnáním

$$\int \tilde{f}_j(\boldsymbol{X}) q^{\setminus j}(\boldsymbol{X}) \mathrm{d}\boldsymbol{X} = \int f_j(\boldsymbol{X}) q^{\setminus j}(\boldsymbol{X}) \mathrm{d}\boldsymbol{X} \tag{2.53}$$

Z čehož zjistíme, že $K = Z_j$ a tedy může být nalezeno přímo z (2.49).

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 $q(\boldsymbol{X})$ v exponenciální rodině, pokud iterace konverguje, pak nalezené řešení je stacionární bod specifické potenciální energie [12].

Algoritmus 3 Expectation propagation {alg:ep}

Inicializujeme všechny aproximativní faktory $\tilde{f}_i(\boldsymbol{X})$ na neinformativní.
Inicializujeme aposteriorní aproximace nastavením:

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

repeat

Vyber faktor $\tilde{f}_j(\boldsymbol{X})$, který bude aktualizován.
Odeber $\tilde{f}_j(\boldsymbol{X})$ z aposteriorní distribuce vydělením:

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

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

$$Z_j = \int q^{\setminus j}(\boldsymbol{X}) f_j(\boldsymbol{X}) \mathrm{d}\boldsymbol{X}$$

Nastav novou hodnotu faktoru:

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

until konvergence
Spočítej aproximaci pozorování:

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

3. Učení parametrů

{ch:ep}

3.1 Grafický model

Máme vybraný faktor f, tento faktor je spojený s několika proměnnými $\boldsymbol{x} = (x_0, x_1, \ldots, x_{N_x})$ a množinami parametrů $\boldsymbol{\Theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{N_\theta})$. Tento faktor reprezentuje podmíněnou pravděpodobnost:

$$f(\boldsymbol{x}, \Theta) = p(x_0 | x_1, \ldots, x_{N_x}; \Theta)$$

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

$$f(\boldsymbol{x}, \Theta) = p(x_0 | x_1, \ldots, x_{N_x}; \Theta) = \theta_{\rho(x'), x_0}$$

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 množinu parametrů propojeny. Pro daný faktor je cavity distribuce $q^{\setminus}(x_i)$, popř. $q^{\setminus}(\boldsymbol{\theta}_i)$ rovna součinu zpráv ze všech ostatních faktorů do x_i, popř $\boldsymbol{\theta}_i$. Aproximovaná marginální pravděpodobnost proměnné je pak součinem cavity distribuce a zprávy z faktoru:

$$q(x_i) = q^{\setminus}(x_i) m_{f \to x_i}(x_i)$$

$$q(\boldsymbol{\theta}_i) = q^{\setminus}(\boldsymbol{\theta}_i) m_{f \to \theta}(\boldsymbol{\theta}_i)$$

Cavity distribuce je právě zpráva z proměnné, popř. množiny parametrů do faktoru.

$$m_{x_i \to f} = q^{\setminus}(x_i)$$

$$m_{\boldsymbol{\theta}_i \to f} = q^{\setminus}(\boldsymbol{\theta}_i)$$

3.2.1 Marginální pravděpodobnost proměnných

Pokud chceme aktualizovat hodnotu naší aproximace marginální pravděpodobnosti, tak je třeba minimalizovat její vzdálenost od skutečné marginální pravděpo-

dobnosti:

$$p^*(\tilde{x}_j) = \sum_{\boldsymbol{x}:x_j=\tilde{x}_j} \int_{\boldsymbol{\Theta}} \prod_i q^{\setminus}(x_i) \prod_l q^{\setminus}(\boldsymbol{\theta}_l) f(\boldsymbol{x};\boldsymbol{\Theta}) \tag{3.1}$$ {eq:1}

$$= \sum_{\boldsymbol{x}:x_j=\tilde{x}_j} \prod_i q^{\setminus}(x_i) \int_{\boldsymbol{\theta}_{\rho(\boldsymbol{x'})}} q^{\setminus}(\boldsymbol{\theta}_{\rho(\boldsymbol{x'})}) \theta_{\rho(\boldsymbol{x'}),x_0} \tag{3.2}$$ {eq:2}

$$= \sum_{\boldsymbol{x}:x_j=\tilde{x}_j} \prod_i q^{\setminus}(x_i) \mathbb{E}_{q^{\setminus}}(\theta_{\rho(\boldsymbol{x'}),x_0}) \tag{3.3}$$ {eq:3}

$$= \sum_{\boldsymbol{x}:x_j=\tilde{x}_j} \prod_i m_{x_i\to f}(x_i) \mathbb{E}_{q^{\setminus}}(\theta_{\rho(\boldsymbol{x'}),x_0}) \tag{3.4}$$

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, je jejich jejich cavity 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

$$p^*(\tilde{x}) = \sum_{\boldsymbol{x}:x_j=\tilde{x}_j} \prod_i m_{x_i\to f}(x_i) \mathbb{E}_{q^{\setminus}}(\theta_{\rho(\boldsymbol{x'}),x_0}) \tag{3.5}$$

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

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

$$m_{f\to x_j}(x_j) = \sum_{\boldsymbol{x}:x_j=\tilde{x}_j} \prod_{i\neq j} m_{x_i\to f}(x_i) \mathbb{E}_{q^{\setminus}}(\theta_{\rho(\boldsymbol{x'}),x_0}) \tag{3.6}$$ {eq:msgfro

3.2.2 Marginální pravděpodobnost parametrů

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

$$p^*(\tilde{\boldsymbol{\theta}}_j) = \sum_{\boldsymbol{x}} \int_{\boldsymbol{\Theta}:\boldsymbol{\theta}_j=\tilde{\boldsymbol{\theta}}_j} \prod_i q^{\setminus}(x_i) \prod_l q^{\setminus}(\boldsymbol{\theta}_l) f(\boldsymbol{x};\boldsymbol{\Theta}) \tag{3.7}$$

$$= \sum_{l\neq j} \sum_{\boldsymbol{x}:\rho(\boldsymbol{x}')=l} \prod_i q^{\setminus}(x_i) \int_{\boldsymbol{\Theta}:\boldsymbol{\theta}_j=\tilde{\boldsymbol{\theta}}_j} \prod_k q^{\setminus}(\boldsymbol{\theta}_k)\theta_{l,x_0} + \tag{3.8}$$

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

$$= \left[\sum_{l\neq j} \sum_{\boldsymbol{x}:\rho(\boldsymbol{x}')=l} \prod_i q^{\setminus}(x_i)\mathbb{E}_{q^{\setminus}(\boldsymbol{\theta}_l)}(\theta_{l,x_0})\right] q^{\setminus}(\tilde{\boldsymbol{\theta}}_j) + \tag{3.9}$$

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

$$= w_0 q^{\setminus}(\tilde{\boldsymbol{\theta}}_j) + \sum_k w_k \tilde{\theta}_{j,k} q^{\setminus}(\tilde{\boldsymbol{\theta}}_j), \tag{3.10}$$

$$= w_0 m_{\tilde{\boldsymbol{\theta}}_j\to f}(\tilde{\boldsymbol{\theta}}_j) + \sum_k w_k \tilde{\theta}_{j,k} m_{\tilde{\boldsymbol{\theta}}_j\to f}(\tilde{\boldsymbol{\theta}}_j), \tag{3.11}$$

kde

$$w_0 = \sum_{l\neq j} \sum_{\boldsymbol{x}:\rho(\boldsymbol{x}')=l} \prod_i m_{x_i\to f}(x_i)\mathbb{E}_{q^{\setminus}(\boldsymbol{\theta}_l)}(\theta_{l,x_0}) \tag{3.12}$$

$$w_k = \sum_{\boldsymbol{x}:\rho(\boldsymbol{x}')=j,x_0=k} \prod_i m_{x_i\to f}(x_i) \tag{3.13}$$

Opět vycházíme z výpočtu marginální pravděpodobnosti ze sdružené pravděpodobnosti. V rovnici (3.8) jsme rozdělili sumu přes $\boldsymbol{x}$ na ty, pro které se ve faktoru použije množina parametrů $\tilde{\boldsymbol{\theta}}_j$ a na ty ostatní. Také jsme z integrálu vytknuli součin cavity distribucí pro proměnné. V dalším kroku (3.9) 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 cavity distribucí pro množiny parametrů. Ve výsledku získáme $q^{\setminus}(\tilde{\boldsymbol{\theta}}_j)\int_{\boldsymbol{\theta}_l} q^{\setminus}(\boldsymbol{\theta}_l)\theta_{l,x_0}$ 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

Stále tu ovšem zůstává problém, že spočítat aproximující distribuci $q(\boldsymbol{\theta}_j)$ může být příliš složité, protože skutečná marginální distribuce je směs několika distribucí a ta nemusí být v obecném případě vyjádřitelná. Je tedy třeba model dále aproximovat. Pro zjednodušení výpočtu jsou zprávy z faktoru do množiny parametrů, $m_{f\to\boldsymbol{\theta}_i}(\boldsymbol{\theta}_i)$, ve tvaru Dirichletovského rozdělení s parametry $\boldsymbol{\alpha}_{f\to\boldsymbol{\theta}_i}$:

$$m_{f\to\boldsymbol{\theta}_i}(\boldsymbol{\theta}_i) = Dir(\boldsymbol{\theta}_i;\boldsymbol{\alpha}_{f\to\boldsymbol{\theta}_i}) = \frac{\Gamma(\sum_j \boldsymbol{\alpha}_{f\to\boldsymbol{\theta}_i,j})}{\prod_j \Gamma(\boldsymbol{\alpha}_{f\to\boldsymbol{\theta}_i,j})} \prod_j \theta_{i,j}^{\boldsymbol{\alpha}_{f\to\boldsymbol{\theta}_i,j}-1} \tag{3.14}$$

kde Γ je Gamma funkce (zobecnění faktoriálu):

$$\Gamma(z) = \int_0^\infty t^{z-1}\exp(-t)\mathrm{d}t \tag{3.15}$$

Dirichletovské rozdělení bylo zvoleno, protože má důležité vlastnosti pro součin, které budou využity dále pro výpočet cavity distribuce a celkové aproximace. Pokud označíme aproximované faktory indexem β a každý bude mít vlastní parametry $\boldsymbol{\alpha}_{f_\beta\to\boldsymbol{\theta}_i}$, tak výsledná aproximace bude tvaru:

$$\begin{aligned} q(\boldsymbol{\theta}_i) &\propto \prod_\beta m_{f_\beta\to\boldsymbol{\theta}_i}(\boldsymbol{\theta}_i) && (3.16)\\ &\propto \prod_\beta \prod_j \theta_{i,j}^{\boldsymbol{\alpha}_{f_\beta\to\boldsymbol{\theta}_i,j}-1} && (3.17)\\ &\propto Dir(\boldsymbol{\theta}_i;\sum_\beta \boldsymbol{\alpha}_{f_\beta\to\boldsymbol{\theta}_i} - (|\beta|-1)\mathbf{1}) && (3.18)\\ &= Dir(\boldsymbol{\theta}_i;\boldsymbol{\alpha}_i) && (3.19) \end{aligned}$$ {eq:ep:apr

kde $\boldsymbol{\alpha}_i = \sum_\beta \boldsymbol{\alpha}_{f_\beta\to\boldsymbol{\theta}_i} - (|\beta|-1)\mathbf{1})$.

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

$$\begin{aligned} q^{\setminus\tilde{\beta}}(\boldsymbol{\theta_i}) &\propto \prod_{\beta\neq\tilde{\beta}} m_{f_\beta\to\boldsymbol{\theta}_i}(\boldsymbol{\theta}_i) && (3.20)\\ &\propto Dir(\boldsymbol{\theta}_i;\boldsymbol{\alpha}_i - \boldsymbol{\alpha}_{f_\beta\to\boldsymbol{\theta}_i} + \mathbf{1}) && (3.21) \end{aligned}$$

Naším cílem je nalézt parametry $\boldsymbol{\alpha}^*$ aproximované marginální pravděpodobnosti (3.19), které minimalizují vzdálenost od skutečné marginální pravděpodobnosti (3.10). Pro měření vzdálenosti mezi dvěma pravděpodobnostními rozloženími se používá Kullback-Leiblerova divergence:

$$KL(p\|q) = \int_{-\infty}^{\infty} p(x)\log\left(\frac{p(x)}{q(x)}\right)\mathrm{d}x \tag{3.22}$$

Pro nalezení minima použijeme algoritmus Expectation Propagation a budeme tedy minimalizovat $KL(p^*\|q)$.

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

$$w_j\theta_j Dir(\boldsymbol{\theta};\boldsymbol{\alpha}) \propto w_j\theta_j \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \prod_i \theta_i^{\alpha_i - 1} \tag{3.23}$$

$$\propto w_j \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \theta_j^{\alpha_j} \prod_{i \neq j} \theta_i^{\alpha_i - 1} \tag{3.24}$$

$$\propto w_j \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \frac{\Gamma(\alpha_j + 1)\prod_{i\neq j}\Gamma(\alpha_i)}{\Gamma(1 + \sum_i \alpha_i)} Dir(\boldsymbol{\theta};\boldsymbol{\alpha} + \boldsymbol{\delta}_j) \tag{3.25}$$

$$\propto w_j \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \frac{\alpha_j\Gamma(\alpha_j)\prod_{i\neq j}\Gamma(\alpha_i)}{(\sum_i \alpha_i)\Gamma(\sum_i \alpha_i)} Dir(\boldsymbol{\theta};\boldsymbol{\alpha} + \boldsymbol{\delta}_j) \tag{3.26}$$

$$\propto w_j \frac{\alpha_j}{\sum_i \alpha_i} Dir(\boldsymbol{\theta};\boldsymbol{\alpha} + \boldsymbol{\delta}_j) \tag{3.27}$$

$$\tag{3.28}$$

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

$$p^*(\boldsymbol{\theta}) = w_0^* Dir(\boldsymbol{\theta};\boldsymbol{\alpha}) + \sum_j w_j^* Dir(\boldsymbol{\theta};\boldsymbol{\alpha} + \boldsymbol{\delta}_j) \tag{3.29}$$

kde

$$w_0^* \propto w_0 \tag{3.30}$$

$$w_j^* \propto w_j \frac{\alpha_j}{\sum_i \alpha_i} \tag{3.31}$$

$$\sum_{i=0}^{k} w_i^* = 1 \tag{3.32}$$

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

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

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

$$= w_0^* \int \boldsymbol{\theta} Dir(\boldsymbol{\theta}; \boldsymbol{\alpha}) \ d\boldsymbol{\theta} + \sum_j w_j^* \int \boldsymbol{\theta} Dir(\boldsymbol{\theta}; \boldsymbol{\alpha} + \boldsymbol{\delta}_j) \ \mathrm{d}\boldsymbol{\theta} \tag{3.35}$$

$$= w_0^* \mathbb{E}_{Dir(\boldsymbol{\alpha})}[\boldsymbol{\theta}] + \sum_j w_j^* \mathbb{E}_{Dir(\boldsymbol{\alpha}+\boldsymbol{\delta}_j)}[\boldsymbol{\theta}] \tag{3.36}$$

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. Střední hodnota proměnné X_i podle Dirichletovského rozdělení je

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

$$Var[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2 \tag{3.37}$$

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

$$\mathbb{E}_{p^*}[\boldsymbol{\theta}^2] = \int \boldsymbol{\theta}^2 p^*(\boldsymbol{\theta}) \ d\boldsymbol{\theta} \tag{3.38}$$

$$= w_0^* \int \boldsymbol{\theta}^2 Dir(\boldsymbol{\theta}; \boldsymbol{\alpha}) \ d\boldsymbol{\theta} + \sum_j w_j^* \int \boldsymbol{\theta}^2 Dir(\boldsymbol{\theta}; \boldsymbol{\alpha} + \boldsymbol{\delta}_j) \ d\boldsymbol{\theta} \tag{3.39}$$

$$= w_0^* \mathbb{E}_{Dir(\boldsymbol{\alpha})}[\boldsymbol{\theta}^2] + \sum_j w_j^* \mathbb{E}_{Dir(\boldsymbol{\alpha}+\boldsymbol{\delta}_j)}[\boldsymbol{\theta}^2] \tag{3.40}$$

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

$$\mathbb{E}_{Dir(\boldsymbol{\alpha})}[x_i^2] = \int x_i^2 Dir(\boldsymbol{x}; \boldsymbol{\alpha}) d\boldsymbol{x} \tag{3.41}$$

$$= \int x_i^2 \frac{\Gamma(\alpha_0)}{\prod_{j=1}^N \Gamma(\alpha_j)} \prod_{j=1}^N x_j^{\alpha_j - 1} d\boldsymbol{x} \tag{3.42}$$

Nyní jsme v podobné situaci jako v (3.23). Budeme postupovat stejně, vyjádříme nové Dirichletovské rozdělení.

$$\mathbb{E}_{Dir(\boldsymbol{\alpha})}[x_i^2] = \int \frac{\Gamma(\alpha_0+2)\alpha_i(\alpha_i+1)}{\alpha_0(\alpha_0+1)\Gamma(\alpha_i+2)\prod_{j\neq i}\Gamma(\alpha_j)} x_i^{\alpha_i+1}\prod_{j\neq i} x_j^{\alpha_j-1}\mathrm{d}\boldsymbol{x} \tag{3.43}$$

$$= \frac{\alpha_i(\alpha_i+1)}{\alpha_0(\alpha_0+1)}\int \frac{\Gamma(\beta_0)}{\prod_i\Gamma(\beta_i)}\prod_i x_i^{\beta_i-1}\mathrm{d}\boldsymbol{x} \tag{3.44}$$

$$= \frac{\alpha_i(\alpha_i+1)}{\alpha_0(\alpha_0+1)}\int Dir(\boldsymbol{x};\boldsymbol{\beta})\mathrm{d}\boldsymbol{x} \tag{3.45}$$

$$= \frac{\alpha_i(\alpha_i+1)}{\alpha_0(\alpha_0+1)} \tag{3.46}$$

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

$$\Gamma(\alpha_0) = \frac{\Gamma(\alpha_0+2)}{\alpha_0(\alpha_0+1)} \tag{3.47}$$

$$\Gamma(\alpha_i) = \frac{\Gamma(\alpha_i+2)}{\alpha_i(\alpha_i+1)} \tag{3.48}$$

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

$$\beta_i = \alpha_i + 2 \tag{3.49}$$

$$\beta_{j\neq i} = \alpha_j \tag{3.50}$$

$$\beta_0 = \sum_i \beta_i \tag{3.51}$$

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

$$\frac{\mathbb{E}[X_1]-\mathbb{E}[X_1^2]}{\mathbb{E}[X_1^2]-\mathbb{E}[X_1]^2} = \frac{\frac{\alpha_1}{\alpha_0}-\frac{\alpha_1(\alpha_1+1)}{\alpha_0(\alpha_0+1)}}{\frac{\alpha_1(\alpha_1+1)}{\alpha_0(\alpha_0+1)}-\frac{\alpha_1^2}{\alpha_0^2}} \tag{3.52}$$ {eq:5}

$$= \frac{\frac{\alpha_1(\alpha_0+1)-\alpha_1(\alpha_1+1)}{\alpha_0(\alpha_0+1)}}{\frac{\alpha_0\alpha_1(\alpha_1+1)-\alpha_1^2(\alpha_0+1)}{\alpha_0^2(\alpha_0+1)}} \tag{3.53}$$

$$= \frac{\alpha_0\alpha_1(\alpha_0-\alpha_1)}{\alpha_1(\alpha_0\alpha_1+\alpha_0-\alpha_0\alpha_1-\alpha_1)} \tag{3.54}$$

$$= \alpha_0 \tag{3.55}$$

$$\alpha_i = \mathbb{E}[X_i]\alpha_0 \tag{3.56}$$ {eq:6}

Z rovnice (3.52) 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 (3.56).

3.4 Algoritmus

Algoritmus 4 Expectation Propagation pro učení parametrů {alg:ep}

Parametry zpráv z faktoru β do množiny parametrů $\boldsymbol{\theta}_i$ označíme $\boldsymbol{\alpha}_{f_\beta \to \boldsymbol{\theta}_i}$.
Parametry zpráv z množiny parametrů $\boldsymbol{\theta}_i$ do faktoru β označíme $\boldsymbol{\alpha}_{\boldsymbol{\theta}_i \to f_\beta}$.
Parametry marginální distribuce množiny parametrů $\boldsymbol{\theta}_i$ označíme $\boldsymbol{\alpha}_i$.
init
  Nastav zprávy mezi faktory a proměnnými na 1.
  Nastav parametry $\boldsymbol{\alpha}_{f_\beta \to \boldsymbol{\theta}_i}$ na 1.
  Nastav parametry $\boldsymbol{\alpha}_i$ na apriorní hodnotu.
end init
repeat
  Vyber faktor $f_{\tilde{\beta}}$, který se bude aktualizovat.
  Spočítej všechny zprávy z parametrů:
  for každý parametr $\boldsymbol{\theta}_i$ spojený s faktorem $f_{\tilde{\beta}}$ **do**
    Parametry zprávy z $\boldsymbol{\theta}_i$ do $f_{\tilde{\beta}}$: $\boldsymbol{\alpha}_{\boldsymbol{\theta}_i \to f_\beta} = \boldsymbol{\alpha}_i - \boldsymbol{\alpha}_{f_\beta \to \boldsymbol{\theta}_i} + 1$.
  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 (3.6):
      $\hat{f}(x_j) = \sum_{\boldsymbol{x}: x_j = \tilde{x}_j} \mathbb{E}_{q \setminus}(\theta_{\rho(\boldsymbol{x}'), x_0}) \prod_{i \neq j} m_{x_i \to f_{\tilde{\beta}}}(x_i)$
  end for
  Aktualizuj marginální pravděpodobnost parametrů:
  for každý parametr $\boldsymbol{\theta}_i$ spojený s faktorem $f_{\tilde{\beta}}$ **do**
    Spočítej parametry $\boldsymbol{\alpha}_i^*$ pro Dirichletovské rozdělení, které nejlépe aproximuje cílovou marginální distribuci (3.29). Metoda popsána v předchozí sekci.
    Parametry zprávy z $f_{\tilde{\beta}}$ do $\boldsymbol{\theta}_i$:
      $\boldsymbol{\alpha}_{f_\beta \to \boldsymbol{\theta}_i} = \boldsymbol{\alpha}_i^* - \boldsymbol{\alpha}_{\boldsymbol{\theta}_i \to f_\beta} + 1$
    Aktualizuj parametry marginální distribuce $q(\boldsymbol{\theta}_i)$
      $\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_i^* = \boldsymbol{\alpha}_{f_\beta \to \boldsymbol{\theta}_i} + \boldsymbol{\alpha}_{\boldsymbol{\theta}_i \to f_\beta}$
  end for
  for každou proměnnou X_i, spojenou s faktorem $f_{\tilde{\beta}}$ **do**
    Aktualizuj zprávy z proměnných do faktoru:
      $m_{x_i \to f_{\tilde{\beta}}}(x_i) = \prod_{\beta \neq \tilde{\beta}} m_{f_\beta \to x_i}(x_i)$
  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 stojí na předchozí. Nejnižší vrstva implementuje efektivní počítání s faktory. Nad ní stojí vrstva reprezentující jednotlivé vrcholy ve faktor grafu. Tato vrstva také obsahuje funkcionalitu pro počítání zpráv. Nejvyšší vrstva se zaměřuje na samotnou propagaci zpráv.

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 rozhraní je samo o sobě faktor. Faktory jsou reprezentovány třídou `Factor`, která podporuje všechny základní matematické operace a také řadu specifických operací jako je marginalizace, normalizace, nastavení faktoru na pozorovanou hodnotu, vybrání nejpravděpodobnějšího přiřazení, atd. Dokumentace třídy je dostupná v příloze **??**.

Odkaz do dokumentace {sec:repfa

4.1.1 Reprezentace faktorů

Každý diskrétní faktor má seznam diskrétních 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 [14]. Pole pak využíváji 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 $\{0, \ldots, n-1\}$, kde n je kardinalita proměnné. Každá hodnota v poli je pak hodnotou faktoru pro nějaké přiřazení hodnot proměnným, např. $(0, 1, 0)$. Jednotlivé hodnoty jsou v poli seřazeny lexikograficky. Pro každou proměnnou si pamatujeme její kardinalitu a také tzv. krok. Krok určuje kolik pro každou proměnnou o kolik hodnot v tabulce se musíme

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. {tab:strid

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

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 rozložení, 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.

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 sdíli 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 faktoru jsou pak výsledkem aplikace operace na prvky ze vstupních faktorů, které

sdílí ohodnocení společných proměnných. Příklad s násobením:

f_1		
X	Y	$Hodnota$
0	0	0.2
0	1	0.3
1	0	0.1
1	1	0.4

$\times$

f_2		
Y	Z	$Hodnota$
0	0	0.2
0	1	0.2
1	0	0.2
1	1	0.4

$=$

f_r			
X	Y	Z	$Hodnota$
0	0	0	0.04
0	0	1	0.04
0	1	0	0.06
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

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 hodnotou $(1, 1)$ 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 ohodnocení $(0, 1, 1)$ na $(1, 0, 0)$, všechny proměnné binární. Při každé úpravě proměnné také aktualizujeme indexy ve

vstupních faktorech.

Pokud se přesunujeme na další hodnotu proměnné, tak stačí k indexu pro faktor přičíst krok upravené proměnné. Pokud je třeba se vrátit 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. Pokud proměnná není obsažena v doméně faktoru, pak její krok je roven 0.

Algoritmus 5 Aplikace operace na faktory s různými doménami {alg:apop}

```
function APPLY-OP(f1, f2, op)
    f1, f2 – vstupní faktory
    op – operace

    fr ← nový faktor pro výsledek operace na f1 a f2 s prázdným polem

    index[f1] ← 0
    index[f2] ← 0
    přiřazení[v] ← 0 pro každou proměnnou v ∈ proměnné[fr]

    for i ∈ {0, ..., LENGTH(pole[fr]) − 1} do
        pole[fr][i] = op(pole[f1][index[f1]], pole[f2][index[f2]])
        for v ∈ REVERSED(proměnné[fr]) do
            přiřazení[v] += 1
            if přiřazení[v] = kardinalita[v] then
                přiřazení[v] ← 0
                index[f1] −= (kardinalita[v] − 1) · krok[f1][v]
                index[f2] −= (kardinalita[v] − 1) · krok[f2][v]
            else
                index[f1] += krok[f1][v]
                index[f2] += krok[f2][v]
                break
            end if
        end for
    end for
    return fr
end function
```

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.

Algoritmus 6 Marginalizace faktoru {alg:marg}

```
function MARGINALIZE(f, vars)
    f – vstupní faktor, pole incializované na 0
    vars – seznam proměnných, které mají zůstat

    f_r ← nový faktor, obsahující pouze proměnné z vars
    přiřazení[v] ← 0 pro každou proměnnou v ∈ vars
    index ← 0                                   ▷ Index do nového faktoru

    for i ∈ {0, ..., LENGTH(pole[f]) − 1} do
        pole[f_r][index] += pole[f][i]

        for v ∈ vars do
            if (i + 1) mod krok[f][v] = 0 then
                přiřazení[v] += 1
                index += krok[f_r][v]
            end if

            if přiřazení[v] = kardinalita[v] then
                přiřazení[v] ← 0
                index −= kardinalita[v] · 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 s proměnnými 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 `node.py` a jejich dokumentace je v příloze **??**.

Odkaz na dokumentaci

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 v daném 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 dohodnout 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 vrhcoly jsou inicializovány vždy při zavolání metody `connect` pro jejich propojení. Pokud ovšem budeme provádět více výpočtů nad jedním grafickým modelem, tak 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 příchozích 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 it-

eraci jen jednou. Pro odeslání zpráv do všech vrcholů slouží jako zkratka metoda `send_messages`.

Implementace vrcholů pro diskrétní proměnné je v třídě `DiscreteVariableNode`. 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. 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 `node.py` jsou implementované vrcholy pro práci s dirichletovskými parametry. Jmenují se `DirichletParameterNode` a `DirichletFactorNode`. V rozhraní se neliší od standardních implementací vrcholů.

`DirichletParameterNode` musí být vždy spojen pouze s `DirichletFactorNode`, který už ale může být dále propojen se standardními 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 faktor vrcholu 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.

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 `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, odlišující 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, patřičná 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

{sec:noch}

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 přídávat 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 dialogového manageru 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, opět můžeme použít libovolnou heuristiku.

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, následně se přesuneme k použití v jednoduchých modelech. Nakonec bude představen příklad systému použitého v Dialog State Tracking Challenge (DSTC) 2013 [23].

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. 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,
    })
```

{lst:crfac

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.

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).

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.

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).

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

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
----------------------------------------
```

{lst:facpr

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
--------------------------------------------------
```

{lst:facmu

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
-------------------------------------------------
```

{lst:facmu

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
-------------------------------------------------
```

{lst:facma

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
-------------------------------------------------
```

{lst:facno

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)]
```

{lst:facmo

Vrcholy

V této sekci ukážeme vytváření jednoduchého grafického modelu a následně poslání 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 hodnoty 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)
```

{lst:nodex

Příklad 9 Nastavení pozorovaných hodnot

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

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

{lst:facob

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()
```

{lst:facms

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
-------------------------------------------------
```

{lst:facre

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()
```

{lst:lbpex

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 `DirichletParameterNode` nyní neurčuje pravděpodobnosti, ale parametry α dirichletovských rozdělení. Pro každou kombinaci hodnot rodičů existuje jedno dirichletovské rozdělení popisující apriorní rozdělení nad pravděpodobností 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 předpokládát, 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.

4.6 Dialog State Tracking Challenge

Knihovna pro inferenci byla použita v Dialog State Tracking Challenge (DSTC) 2013 [23]. 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 [18]. Zároveň poskytli i baseline dialogový systém. Cílem účastníků soutěže bylo vytvořit systém odhadu 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,
- *date* – datum odjezdu,
- *from.description* – popis místa odjezdu,
- *to.description* – popis místa příjezdu,
- *from.monument* – významný monument v místě odjezdu,

Příklad 13 Vytvoření grafického modelu s dirichletovskými parametry {lst:crtep

```
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 {lst:lbpep

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

obs_2.observed({
    ('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 {lst:epres

```
>>> 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
--------------------------------------------------
```

Obrátka	Aktér	Přepis
1	Systém dact	*Welcome to the Lets Go! bus information system for Pittsburgh. Where would you like to leave from?* hello()&request(from.desc)
	Uživatel dact	*The South Side.* inform(from.desc=„the south side“)
2	Systém dact	*Leaving from the south side neighborhood. Did I get that right?* confirm(from.desc=„the south side“)
	Uživatel dact	*Correct.* affirm()
3	Systém dact	*Where do you wanna go?* request(to.desc)
	Uživatel dact	*Oakland.* inform(to.desc=„Oakland“)
4	Systém dact	*When do you wanna travel?* request(time)
	Uživatel dact	*Next.* inform(time=next)
5	Systém dact	*The next bus is the 54C, leaving from south eighteenth 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, restart, or goodbye.* ...

Tabulka 4.2: Příklad dialogu mezi uživatelem a dialogovým systémem Let's Go. *dact* znamená dialogový akt.

{tab:examp

- *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 4.2. Uživatel může informovat systém o hodnotě libovolného slotu. Systém se 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

systému (live data). Dva dialogové systémy produkovaly seznam n nejlepších hypotéz. Jeden systém produkoval pouze první nejlepší hypotézu, záznamy hovorů 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 Generativní model

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.

Stav důvěry můžeme vyjádřit ze sdružené pravděpodobnosti generativního modelu:

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

Pro zrychlení výpočtu bylo využito stažených parametrů [21] s manuálně nastavenými hodnotami:

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

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:

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

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 [7], čím menší, tím lepší.

Výsledky ukazují (viz tabulka 4.3), že generativní model překonává baseline na všech datových sadách, kromě s batch dat. Manuální inspekcí 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 se svou úspěšností zařadil mezi většinu systémů (viz tabulka 4.4). 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 [29].

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 4.3: Přesnost systému pro odhad stavu na live a batch testovacích datech, kde BS je baseline systém a GS je generativní systém. All značí průměrné skóre přes všechny testovací sady.

{t:all:dat

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 4.4: 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.

{t:DSTC:ra

Závěr

Práce splnila všechny vytyčené cíle. Výsledkem je implementace metod pro odhad stavu a parametrů v dialogových systémech. Byly představeny nejpoužívanější metody pro inferenci v grafických modelech, navíc byl odvozen algoritmus pro inferenci parametrů dirichletovského rozdělení srovnáním momentů.

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] Bernardo, J. M.; Smith, A. F.: *Bayesian theory*, ročník 405. Wiley. com, 2009.

[3] 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.

[4] Bishop, C. M.; Nasrabadi, N. M.: *Pattern recognition and machine learning*, ročník 1. springer New York, 2006.

[5] 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.

[6] 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.

[7] Brier, G. W.: Verification of forecasts expressed in terms of probability. *Monthly weather review*, ročník 78, č. 1, 1950: s. 1–3.

[8] He, Y.; Young, S.: Semantic processing using the hidden vector state model. *Computer speech & language*, ročník 19, č. 1, 2005: s. 85–106.

[9] Koller, D.; Friedman, N.: *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[10] Kullback, S.: *Information theory and statistics*. Courier Dover Publications, 1997.

[11] 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.

[12] 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.

[13] 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.

[14] Oliphant, T. E.: *Guide to NumPy*. Provo, UT, Březen 2006.
URL `http://www.tramy.us/`

[15] Pearl, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausble Inference*. Morgan Kaufmann Pub, 1988.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] Walker, W.; Lamere, P.; Kwok, P.; aj.: Sphinx-4: A flexible open source framework for speech recognition. 2004.

[23] 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.

[24] 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.

[25] Young, S.; Evermann, G.; Gales, M.; aj.: The HTK book. *Cambridge University Engineering Department*, ročník 3, 2002.

[26] 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.

[27] 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.

[28] 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.

[29] 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