NPFL099 Statistical Dialogue Systems
6. Dialogue Management (1)
mostly Dialogue State Tracking

http://ufal.cz/npfl099

Ondřej Dušek, Vojtěch Hudeček & Tomáš Nekvinda
8. 11. 2021
Dialogue Management & State

• Dialogue management consists of:
  • **State update** ← we need to track dialogue state over time
  • Action selection (discussed later)

• **Dialogue state** needed to remember what was said in the past
  • tracking the dialogue progress
  • summary of the whole dialogue history
  • basis for action selection decisions

*U:* I’m looking for a restaurant in the city centre.
*S:* OK, what kind of food do you like?
*U:* Chinese.
*S:* What part of town do you have in mind?
*×* *S:* Sure, the Golden Dragon is a good Chinese restaurant. It is located in the west part of town.
*×* *S:* Sure, the Golden Dragon is a good Chinese restaurant. It is located in the city centre.

*✓* *S:* Sure, the Golden Dragon is a good Chinese restaurant. It is located in the city centre.
Dialogue State Contents

• “All that is used when the system decides what to say next” (Henderson, 2015)

• **User goal/preferences ~ NLU output**
  • slots & values provided (search constraints)
  • information requested

• **Past system actions**
  • information provided
    • slots and values
    • list of venues offered
  • slots confirmed
  • slots requested

• **Other semantic context**
  • user/system utterance: bye, thank you, repeat, restart etc.

U: Give me the address of the first one you talked about.
U: Is there any other place in this area?

S: OK, Chinese food. […]
S: What time would you like to leave?
Problems with Dialogue State

• NLU is unreliable
  • takes unreliable ASR output
  • makes mistakes by itself – some utterances are ambiguous
  • output might conflict with ontology

• Possible solutions:
  • detect contradictions, ask for confirmation
  • ignore low-confidence NLU input
    • what’s “low”?
    • what if we ignore 10x the same thing?

• Better solution: make the state probabilistic – belief state

ASR: 0.5 I’m looking for an expensive hotel
     0.5 I’m looking for inexpensive hotels

NLU: 0.3 inform(type=restaurant, stars=5)

only hotels have stars!
Belief State

• Assume we don’t know the true current dialogue state $s_t$
  • states (what the user wants) influence observations $o_t$ (what the system hears)
  • based on observations $o_t$ & system actions $a_t$, we can estimate a probability distribution $b(s)$ over all possible states – belief state

• More robust than using dialogue state directly
  • accumulates probability mass over multiple turns
    • low confidence – if the user repeats it, we get it the 2nd time
    • accumulates probability over NLU n-best lists

• Plays well with probabilistic dialogue policies (POMDPs)
  • but not only them – rule-based, too
no probability accumulation (1-best, no state)

accumulating over NLU n-best list (still no state)

accumulating over NLU n-best + turns

this is what we need (=belief state)
Basic Discriminative Belief Tracker

- **Partition the state** by assuming conditional independence
  - simplify – assume each slot is independent:
    - state $s = [s^1, ... s^N]$, belief $b(s_t) = \prod_i b(s^i_t)$

- **Always trust the NLU**
  - this makes the model parameter-free
  - ...and basically rule-based
  - but very fast, with reasonable performance

The belief state update rule is deterministic

\[
\begin{align*}
    b(s^i_t) &= \sum_{s^i_{t-1}, o^i_t} p(s^i_t | a^i_{t-1}, s^i_{t-1}, o^i_t) b(s^i_{t-1}) \\
    p(s^i_t | a^i_{t-1}, s^i_{t-1}, o^i_t) &= \begin{cases} 
        p(o^i_t) & \text{if } s^i_t = o^i_t \land o^i_t \neq ⏯ \\
        p(o^i_t) & \text{if } s^i_t = s^i_{t-1} \land o^i_t = ⏯ \\
        0 & \text{otherwise}
    \end{cases} \\
    b(s^i_t) &= \begin{cases} 
        p(s^i_t = ⏯)p(o^i_t = ⏯) & \text{if } s^i_t = ⏯ \\
        p(o^i_t = s^i_t) + p(o^i_t = ⏯)p(s^i_t = s^i_{t-1}) & \text{otherwise}
    \end{cases}
\end{align*}
\]
Basic Feed-forward Neural Tracker

• a simple feed-forward (fully connected) network
  • input – features (w.r.t. slot-value $v$ & time $t$)
    • NLU score of $v$
    • n-best rank of $v$
    • user & system intent (inform/request)
    • … – other domain-independent, low-level NLU features
  • 3 tanh layers
  • output – softmax
    (= probability distribution over values)
  • static – does not model dialogue as a sequence
    • uses a sliding window:
      current time $t$ + few steps back + $\sum$ previous

(Henderson et al., 2013)
https://aclweb.org/anthology/W13-4073
Basic RNN Tracker

- plain sigmoid RNN with a memory vector
  - not quite LSTM/GRU, but close
  - memory updated separately, used in belief update
  - turn-level LSTM would work similarly
- does not need NLU
  - turn features = lexicalized + delexicalized n-grams from ASR n-best list, weighted by confidence
- delexicalization is very harsh: `<slot> <value>`
  - you don’t even know which slot it is
  - this apparently somewhat helps the system generalize across domains
- **dynamic** – explicitly models dialogue as sequence
  - using the network recurrence

(Mrkšić et al., 2015)
http://arxiv.org/abs/1506.07190
Incremental Recurrent Tracker

• Simple: LSTM over words + classification on hidden states
  • runs over the whole dialogue history (user utterances + system actions)
  • classification can occur after each word, right as it comes in from ASR

• Dynamic/sequential

• Doesn’t use any NLU
  • infrequent values are delexicalized (otherwise it can’t learn them)

• Slightly worse performance – possible causes:
  • only uses ASR 1-best
  • very long recurrences (no hierarchy)

(Žilka & Jurčiček, 2015)
https://dl.acm.org/citation.cfm?id=2955040
http://arxiv.org/abs/1507.03471
• No delexicalization needed
• Current turn + rule-based updates (=static tracker)
• Pretrained word vectors (kept fixed)
  • GloVe enhanced with paraphrases
• Text = n-gram sums/CNNs, summed
  • same parameters + handling for all inputs
    • contextual: requested/confirmed slot (+value)
    • current user utterance
    • candidate slot-value pair (run once for each)
• Simple combinations
  • dot product, feed-forward
  • binary decision: is the candidate correct?

( Mrkšić et al., 2017) https://www.aclweb.org/anthology/P17-1163
Candidate Ranking

- Previous systems consider all values for each slot
  - this is a problem for open-ended slots (e.g. restaurant name)
  - enumerating over all takes ages, some are previously unseen
- Alternative: always consider just $K$ candidates
  - use last $K$ candidates from system actions and NLU output
    - NB: only way history is incorporated here (~static)
  - select from them using a per-slot softmax

![Diagram](https://arxiv.org/abs/1712.10224)

- pictures assume $K = 2$
- 2 sigmoid layers
  - representation of $i$-th candidate: utterance/slot/candidate features (next slide)
  - additional values to consider (even if not mentioned in NLU)
  - padding (not enough values mentioned)

(Rastogi et al., 2017)
Candidate Ranking – representation

• Using BiGRU over lexicalized & delexicalized utterance

• Features:
  • **utterance** – last GRU state + NLU indicators for non-slot DAs (user & prev. system)
  • **slot** – NLU indicators for DAs with this slot (user & prev. system) \( \text{inform(slot=*)}, \text{request(slot)} \)
    + last turn scores for \( \text{null} \) & \( \text{dontcare} \)
  • **candidate** – GRU states over matched value words
    + NLU indicators for DAs with this slot & value (user & prev. system) \( \text{inform(slot=value)} \)

(Rastogi et al., 2017)
https://arxiv.org/abs/1712.10224
Candidate Ranking Extensions

• What if multiple values are true?
  • previous approach picks one (softmax)
  • use set of binary classifiers (log loss) instead

• Making it dynamic
  • embedding previous states, system actions, text of the whole dialogue

• Hybrid classify/rank
  • ranking is faster & more flexible vs. classification can be more accurate for some slots
    • generally ranking better with many values, classification with fewer values
  • check for performance on development data & decide which model to use

(Goel et al., 2018)
http://arxiv.org/abs/1811.12891

(Goel et al., 2019)
http://arxiv.org/abs/1907.00883
BERT & Span Selection  a.k.a. Span Tagging  
(≈question answering/reading comprehension)

• BERT over previous system & current user utterance
• from 1st token’s representation, get a **decision:** **none/dontcare/span**
  • per-slot (BERT is shared, but the final decision is slot-specific)
• span = need to find a concrete value as a span somewhere in the text
  • **predict start & end token** of the span using 2 softmaxes over tokens
• rule-based update (static):
  • if **none** is predicted, keep previous value

---

(Chao & Lane, 2019)  
http://arxiv.org/abs/1907.03040
Span Selection with Modelled Update

- Also uses BERT, but not necessarily
  - works slightly worse with random-initialized word embeddings
- sequence of 3 decisions
  - do we carry over last turn’s prediction? (Yes/No) (~static tracking, but not so rigid)
  - if no: what kind of answer are we looking for? (yes/no/dontcare/span of text)
  - if span: predict span’s start and end

(Gao et al., 2019)  
https://www.aclweb.org/anthology/W19-5932/
Span Selection & Better Copying

• “triple-copy” – gets the value from 3 sources:
  • user utterance (same as previous span tagging models)
  • system informs (last value the system mentioned)
  • another slot (coreference), e.g. a taxi ride to a hotel (hotel name = destination)

• rule-based update (static)

Boolean slots are handled separately (classification)

Coreference – distribution over slots to copy from

Same decision as previously, just different options:
none/dontcare/span/inform/refer
Multi-domain Span Selection

- encode domain & slot names w. static pretrained word-embeddings (GloVe)
  - adding **new unseen domains & slots** is easy (no retraining)
- otherwise similar as previous, BERT-based:
  - decide if domain changed (BERT: yes/no/chitchat)
  - if yes, detect new domain(s) (BERT + GloVe: 1/0 for domain candidate)
  - for each domain, find values (BERT + GloVe span selection)

(Dey & Desarkar, 2021)
https://aclanthology.org/2021.sigdial-1.23
Generator-based Tracker

- Similar to span selection: encodes whole dialogue history (static)
- Pointer-generator seq2seq decoder produces values
  - specific start token for each slot -- copies from input & generates new tokens
- Slot gate: “use generated”/\textit{dontcare}/\textit{none}
  - same as the decisions done in span tagging, just applied after getting the value

\begin{itemize}
  \item takes concatenated dialogue history
  \item $G_i$: Slot Gate
  \item $c_{j=0}$: Context Vector
  \item $p_{history_j}$: Utterances
  \item $p_{generate}$: Domain
  \item $p_{attend}$: Utterances
  \item $p_{slot}$: Slots
  \item $h_{seq}$: State Generator
\end{itemize}

pointer-generator net (see NLU lecture): can\textbf{ generate} tokens from vocabulary or\textbf{ copy} tokens from attention

\begin{itemize}
  \item specific start token for each slot (& domain)
\end{itemize}

(Wu et al., 2019)
https://www.aclweb.org/anthology/P19-1078
Action Selection / Policy

• Dialogue management:
  • **State tracking** (↑)
  • **Action selection/Policy** (↓)

• action selection – **deciding what to do next**
  • based on the current belief state – under uncertainty
  • following a **policy** (strategy) towards an end **goal** (e.g. book a flight)
  • controlling the coherence & flow of the dialogue
  • actions: linguistic & non-linguistic

• DM/policy should:
  • manage uncertainty from belief state
  • recognize & follow dialogue structure
  • plan actions ahead towards the goal

*Did you say Indian or Italian?*

follow convention, don’t be repetitive

e.g. ask for all information you require

(from Milica Gašić’s slides)
Action Selection Approaches

• Finite-state machines
  • simplest possible
  • dialogue state is machine state

• Frame-based (VoiceXML)
  • slot-filling + providing information – basic agenda
  • rule-based in essence

• Rule-based
  • any kind of rules (e.g. Python code)

• Statistical
  • typically using reinforcement learning
Why Reinforcement Learning

- **Action selection ~ classification** → use supervised learning?
  - set of possible actions is known
  - belief state should provide all necessary features

- Yes, but…
  - You’d need sufficiently large **human-human data** – hard to get
    - human-machine would just mimic the original system
  - Dialogue is ambiguous & complex
    - there’s **no single correct next action** – multiple options may be equally good
    - but datasets will only have one next action
    - **some paths will be unexplored** in data, but you may encounter them
  - DSs won’t behave the same as people
    - ASR errors, limited NLU, limited environment model/actions
    - **DSs should behave differently** – make the best of what they have
  - supervised classification **doesn’t plan ahead**!
    - RL optimizes for the whole dialogue, not just the immediate action
RL World Model: Markov Decision Process

- **MDP = probabilistic control process**
  - modelling situations that are partly random, partly controlled
- **agent in an environment:**
  - has internal **state** $s_t \in S$ (~ dialogue state)
  - takes **actions** $a_t \in A$ (~ system dialogue acts)
  - actions chosen according to **policy** $\pi: S \rightarrow A$
  - gets **rewards** $r_t \in \mathbb{R}$ & state changes from the environment
- **rewards are typically handcrafted**
  - very high positive for a successful dialogue (e.g. +40)
  - high negative for unsuccessful dialogue (-10)
  - small negative for every turn (-1, promote short dialogues)
- **Markov property** – state defines everything
  - no other temporal dependency
- **policy may be deterministic or stochastic**
  - stochastic: prob. dist. of actions, sampling

(from Milica Gašić’s slides)

(Sutton & Barto, 2018)
Partially-observable MDPs

- **POMDPs** – **belief** states instead of dialogue states
  - true states ("what the user wants") are not observable
  - observations ("what the system hears") depend on states
  - belief – probability distribution over states
  - can be viewed as **MDPs with continuous-space states**
    - just represent 1 slot as set of binary floats 😊

- All MDP algorithms work…
  - if we **quantize/discretize** the states
  - use grid points & nearest neighbour approaches
  - this might introduce errors / make computation complex

- Deep RL typically works out of the box
  - function approximation approach, allows continuous states

---

(from Milica Gašić’s slides)

https://en.wikipedia.org/wiki/Voronoi_diagram
Simulated Users

- Static datasets aren’t enough for RL
  - data might not reflect our newly learned behaviour
- RL needs a lot of data, more than real people would handle
  - 1k-100k’s dialogues used for training, depending on method

- solution: **user simulation**
  - basically another DS/DM
  - (typically) working on DA level
  - errors injected to simulate ASR/NLU

- approaches:
  - rule-based (frames/agenda)
  - n-grams
  - MLE/supervised policy from data
  - combination (best!)
for a typical DS, the belief state is too large to make RL tractable

solution: map state into a reduced space, optimize there, map back

reduced space = **summary space**
- handcrafted state features
  - e.g. top slots, # found, slots confirmed…

reduced action set = **summary actions**
- e.g. just DA types \(\text{inform, confirm, reject}\)
  - remove actions that are not applicable
  - with handcrafted mapping to real actions

state is still tracked in original space
- we still need the complete information for accurate updates
Reinforcement learning: Definition

- RL = finding a **policy that maximizes long-term reward**
  - unlike supervised learning, we don’t know if an action is good
  - immediate reward might be low while long-term reward high

\[ R_t = \sum_{t=0}^{\infty} \gamma^t r_{t+1} \]

alternative – **episodes**: only count to \( T \) when we encounter a terminal state (e.g. 1 episode = 1 dialogue)

\( \gamma \in [0,1] = \text{discount factor} \)
(Immediate vs. future reward trade-off)

\( \gamma < 1: R_t \) is finite (if \( r_t \) is finite)
\( \gamma = 0: \) greedy approach (ignore future rewards)

- state transition is stochastic \( \rightarrow \) maximize **expected return**

\[ \mathbb{E}[R_t | \pi, s_0] \]

expected \( R_t \) if we start from state \( s_0 \) and follow policy \( \pi \)
Summary

- **State tracking**: track user goal over multiple turns (probabilistic – belief state)
  - good NLU + rules – works well (and used frequently)
  - static (sliding-window/rule-based update) vs. dynamic (explicit modelling)
  - with vs. without NLU
- **classification** vs. candidate ranking vs. span selection vs. generation
  - classifiers are more accurate than rankers but slower, limited to seen values
  - span selection is the SotA approach, works nicely but probably slow
  - many architectures (FC/CNN/RNN), newest mostly BERT-based
- **Action selection**: deciding what to do next (following a policy)
  - FSM, frames, rule-based, supervised, reinforcement learning
  - RL – agent in an environment, taking actions, getting rewards
    - MDP formalism (+POMDP can be converted to it)
    - summary states might be needed
    - trained often with user simulators
Thanks

Contact us:
https://ufaldsg.slack.com/
{odusek,hudecek,nekvinda}@ufal.mff.cuni.cz
Skype/Meet/Zoom/Troja (by agreement)

Get these slides here:
http://ufal.cz/npfl099

References/Inspiration/Further:
• Milica Gašić’s slides (Cambridge University): http://mi.eng.cam.ac.uk/~mg436/teaching.html

Labs in 10 minutes
DailyDialog training

Next Mon 15:40
rest of Dialogue Policy