NPFL123 Dialogue Systems

6. Dialogue Policy
(non-neural)

https://ufal.cz/npfl123

Ondřej Dušek, Patrícia Schmidtová, Vojtěch Hudeček & Jan Cuřín

20. 3. 2023
Dialogue Management

- Two main components:
  - State tracking (last lecture)
  - Action selection with a policy (today)
- 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)
DM/Action Selection Approaches

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

- **Frame-based** (VoiceXML)
  - slot-filling + providing information – basic agenda

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

- **Statistical**
  - typically using reinforcement learning

- Note that state tracking differs with different action selection
FSM Dialogue Management

Dialogues = graphs going through possible conversations
- nodes = system actions
- edges = possible user response semantics

Advantages:
- easy to design
- predictable

Disadvantages:
- very rigid – not real conversations (ignores anything that’s not a reply to last question)
- doesn’t scale to complex domains

Good for basic DTMF (tone-selection) phone systems

Thanks for calling Bank X. For account balance, press 1, for money transfers, press 2…
Frame-based Approach

• Making the interaction more flexible
• State = frame with slots
  • required slots need to be filled
  • this can be done in any order
  • more information in one utterance possible
• If all slots are filled, query the database
• Multiple frames (e.g. flights, hotels…)
  • needs frame tracking
• Standard implementation: VoiceXML
• Still not completely natural, won’t scale to more complex problems

Slot | Question
---|---
ORIGIN | What city are you leaving from?
DEST | Where are you going?
DEPT DATE | What day would you like to leave?
DEPT TIME | What time would you like to leave?
AIRLINE | What is your preferred airline?

(from Hao Fang’s slides)

```
<form>
  <field name="transporttype">
    <prompt>Please choose airline, hotel, or rental car. </prompt>
    <grammar type="application/x=nuance-gsl">
      [airline hotel "rental car"]
    </grammar>
  </field>
  <prompt>You have chosen <value expr="transporttype">. </prompt>
</form>
```

(from Pierre Lison’s slides)
We can use a probabilistic belief state
  - DA types, slots, values
  - With **if-then-else** rules in programming code
  - using thresholds over belief state for reasoning
  - Output: system DA
  - Very flexible, easy to code
    - allows relatively natural dialogues
  - Gets messy
  - Dialogue policy is still pre-set
    - which might not be the best thing to do

(Jurčiček et al., 2014)  
https://github.com/UFAL-DSG/alex/blob/master/alex/applications/PublicTransportInfoCS/hdc_policy.py
DM with supervised learning

- **Action selection ~ classification** → use supervised learning?
  - set of possible actions is known
  - belief state should provide all necessary features
- Yes, but…
  - You **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
DM as a Markov Decision Process

• MDP = probabilistic control process
  • modelling situations that are partly random, partly controlled
  • **agent** in an **environment**:
    • has internal **state** $s_t \in S$
    • takes **actions** $a_t \in A$
    • actions chosen according to **policy** $\pi: S \rightarrow A$
    • gets **rewards** $r_t \in \mathbb{R}$ & state changes from the environment
  • Markov property – state defines everything
    • no other temporal dependency

• let’s assume we know the state for now
  • let’s go with MDPs, see how they map to POMDPs later

(from Milica Gašić’s slides)

(Sutton & Barto, 2018)
Deterministic vs. stochastic policy

• **Deterministic** = simple mapping $\pi: S \rightarrow A$
  - always takes the same action $\pi(s)$ in state $s$
  - enumerable in a table
  - equivalent to a rule-based system
  - but can be learned instead of hand-coded!

• **Stochastic** = specifies a probability distribution $\pi(s, a)$
  - $\pi(s, a) \sim$ probability of choosing action $a$ in state $s$ – $p(a|s)$
  - decision = sampling from $\pi(s, a)$
Reinforcement learning

- 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

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

- **return**: accumulated long-term reward (from timestep \( t \) onwards)
- \( \gamma \in [0,1] = \text{discount factor} \)
  - 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 \( \pi \)
State-value Function

- Using return, we define the **value of a state** $s$ under policy $\pi$: $V^\pi(s)$
  - Expected return for starting in state $s$ and following policy $\pi$
- Return is recursive: $R_t = r_{t+1} + \gamma \cdot R_{t+1}$
- This gives us a recursive equation (**Bellman Equation**):

  $$V^\pi(s) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi, s_0 = s \right] = \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} p(s'|s, a)(r(s, a, s') + \gamma V^\pi(s'))$$

- $V^\pi(s)$ defines a **greedy policy**:

  $$\pi(s, a) := \begin{cases} 
  \frac{1}{\# \text{ of } a's} & \text{for } a = \arg \max_a \sum_{s' \in \mathcal{S}} p(s'|s, a)(r(s, a, s') + \gamma V^\pi(s')) \\
  0 & \text{otherwise}
  \end{cases}$$

  actions that look best for the next step
Action-value (Q-)Function

- \( Q^\pi(s, a) \) – exp. return of taking action \( a \) in state \( s \), under policy \( \pi \)
  - Same principle as value \( V^\pi(s) \), just **considers the current action, too**
  - Has its own version of the Bellman equation

\[
Q^\pi(s, a) = \mathbb{E}\left[ \sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s, a_0 = a \right] = \sum_{s' \in S} p(s'|s, a) \left( r(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} Q^\pi(s', a') \pi(s', a') \right)
\]

- \( Q^\pi(s, a) \) also defines a greedy policy:
  - \( \pi(s, a) := \begin{cases} \frac{1}{\# \text{ of } a' \text{'s}} & \text{for } a = \arg \max_a Q^\pi(s, a) \\ 0 & \text{otherwise} \end{cases} \)

  again, “actions that look best for the next step”

  simpler: no need to enumerate \( s' \), no need to know \( p(s'|s, a) \) and \( r(s, a, s') \)

  but \( Q \) tables are bigger than \( V \) tables
Optimal Policy in terms of $V$ and $Q$

- **optimal policy** $\pi^*$ – one that maximizes expected return $\mathbb{E}[R_t | \pi]$
  - $V^\pi(s)$ expresses $\mathbb{E}[R_t | \pi]$ → use it to define $\pi^*$

- $\pi^*$ is a policy such that $V^{\pi^*}(s) \geq V^{\pi'}(s)$ $\forall \pi'$, $\forall s \in S$
  - $\pi^*$ always exists in an MDP (need not be unique)
  - $\pi^*$ has the **optimal state-value function** $V^*(s) := \max_{\pi} V^\pi(s)$
  - $\pi^*$ also has the **optimal action-value function** $Q^*(s, a) := \max_{\pi} Q^\pi(s, a)$

- greedy policies with $V^*(s)$ and $Q^*(s, a)$ are optimal
  - we can search for either $\pi^*$, $V^*(s)$ or $Q^*(s, a)$ and get the same result
  - each has their advantages and disadvantages
• Quantity to optimize:
  • value function – **critic**
  • policy – **actor**
  • (both – actor-critic – omitted)

• Environment model:
  • **model-based** (assume known \( p(s'\mid s, a), r(s, a, s') \))
    • makes for mathematically nice solutions
    • but you can only know the full model in limited settings
  • **model-free** (don’t assume anything, sample)
    • this is the one for “real-world” use
    • using \( Q \) instead of \( V \) comes handy here (“hiding” \( p(s'\mid s, a) \))
RL Approaches

• How to optimize:
  • **dynamic programming** – find the exact solution from Bellman equation
    • iterative algorithms, refining estimates
    • expensive, assumes known environment (=must be model-based)
  • **Monte Carlo** learning – learn from experience
    • sample, then update based on experience
  • **Temporal difference** learning – like MC but look ahead (bootstrap)
    • sample, refine estimates as you go

• Sampling & updates:
  • **on-policy** – improve the policy while you’re using it for decisions
  • **off-policy** – decide according to a different policy
1) Choose a threshold $\tau$, Initialize $V_0(s)$ arbitrarily

2) While $V_i(s) - V_{i-1}(s) \geq \tau$ for any $s$: 
   
   for all $s$: $V_{i+1}(s) \leftarrow \max_a \sum_{s' \in S} p(s'|s,a) (r(s,a,s') + \gamma V_i(s'))$

   $i \leftarrow i + 1$

• At convergence, we’re less than $\tau$ away from optimal state values
  
  • resulting greedy policy is typically already optimal in practice

• Can be done with $Q_i(s,a)$ instead of $V_i(s)$

• Assumes known $p(s'|s,a)$ and $r(s,a,s')$
  
  • can be estimated from data if not known – but it’s expensive
Value iteration example (Gridworld)

- Robot in a maze: can stay or move ←, ↑, →, ↓ (all equally likely)
  - reward +1 for staying at “G”
  - reward -1 for hitting a wall
  - discount factor $\gamma = 0.9$

(Heidrich-Meisner et al., 2007)
https://christian-igel.github.io/paper/RLiaN.pdf
https://youtu.be/9YN1R6Lh9Jo
(note that rewards here come from states, not movements)
Monte Carlo Methods

- $V(s)$ or $Q(s, a)$ estimated iteratively, on-policy
  - explores states with more value more often
- Loop over episodes (dialogues)
  - record $(s_t, a_t, r_t)$ for $t = 0, ..., T$ in the episode
  - for all $s, a$ in the episode:
    - $R(s, a) \leftarrow$ list of all returns for taking action $a$ in state $s$ (sum of rewards till end of episode)
    - $Q(s, a) \leftarrow$ mean($R(s, a)$)
- To converge, we need to explore – using $\epsilon$-greedy policy:
  \[
  a = \begin{cases} 
  \arg \max_a Q(s, a) & \text{with probability } 1 - \epsilon \\
  \text{random action} & \text{with probability } \epsilon
  \end{cases}
  \]
  - $\epsilon$ can be large initially, then gradually lowered
- $R_t = \sum_{i=t}^{T-1} \gamma^{i-t} r_{i+1}$
- Off-policy extensions exist (omitted)

NPFL123 L6 2023
• estimate $Q(s, a)$ iteratively, on-policy, with immediate updates
  • **TD**: don’t wait till the end of episode
• choose learning rate $\alpha$, initialize $Q$ arbitrarily
• for each episode:
  • choose initial $s$, initial $a$ according to $\epsilon$-greedy policy based on $Q$
  • for each step:
    • take action $a$, observe reward $r$ and state $s'$
    • choose action $a'$ from $s'$ acc. to $\epsilon$-greedy policy based on $Q$
    • $Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma Q(s', a'))$
    • $s \leftarrow s'$, $a \leftarrow a'$
• typically converges faster than MC (but not always)
Q-Learning (off-policy TD)

- off-policy – directly estimate $Q^*(s, a)$
  - regardless of policy used for sampling
- choose learning rate $\alpha$, initialize $Q$ arbitrarily
- for each episode:
  - choose initial $s$
  - for each step:
    - choose $a$ from $s$ according to $\epsilon$-greedy policy based on $Q$
    - take action $a$, observe reward $r$ and state $s'$
    - $Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \left( r + \gamma \cdot \max_{a'} Q(s', a') \right)$
    - $s \leftarrow s'$

update uses best $a'$, regardless of current policy: $a'$ is not necessarily taken in the actual episode

any policy that chooses all actions & states enough times will converge to $Q^*(s, a)$

Animated example for SARSA & Q-Learning: https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html
https://towardsdatascience.com/td-in-reinforcement-learning-the-easy-way-f92ecfa9f3ce
REINFORCE: Policy gradients

- we assume a differentiable parametric policy $\pi(a|s, \theta)$
- MC search for policy parameters by stochastic gradient ascent
  - looking to maximize performance $J(\theta) = V^{\pi\theta}(s_0)$
- choose learning rate $\alpha$, initialize $\theta$ arbitrarily
- loop forever:
  - generate an episode $s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T$, following $\pi(\cdot | \cdot, \theta)$
  - for each $t = 0, 1, ..., T$: $\theta \leftarrow \theta + \alpha \gamma^t R_t \nabla \ln \pi(a_t|s_t, \theta)$

variant: discounting a baseline $b(s)$ (predicted by any model)
- $R_t - b(s_t)$ instead of $R_t$ gives better performance
- a good $b(s)$ is actually $V(s)$

Returns
- $R_t = \sum_{i=t}^{T-1} \gamma^{i-t} r_{i+1}$
- this is stochastic $\nabla J(\theta)$
  - from policy gradient theorem
POMDP Case

• POMDPs – belief states instead of dialogue states
  • probability distribution over states
  • can be viewed as **MDPs with continuous-space states**
• All MDP algorithms work…
  • if we **quantize/discretize** the states
  • use grid points & nearest neighbour approaches
  • this might introduce errors / make computation complex
• REINFORCE/policy gradients work out of the box
  • function approximation approach, allows continuous states

---

(from Milica Gašić's slides)

[Image of Voronoi diagram]
• 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 (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
Simulated Users

• We can’t really learn just from static datasets
  • on-policy algorithms don’t work
  • 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 policy from data
Summary

• Action selection – deciding what to do next

• Approaches
  • Finite-state machines (system-initiative)
  • Frames (VoiceXML)
  • Rule-based
  • Machine learning (RL better than supervised)

• RL – in a POMDP scenario (can be approximated by MDP)
  • optimizing value function or policy
  • learning on-policy or off-policy
  • learning with or without a model
  • using summary space
  • training with a user simulator
Thanks

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

Get these slides here:
http://ufal.cz/npfl123

References/Inspiration/Further:
- Milica Gašić’s slides (Cambridge University): http://mi.eng.cam.ac.uk/~mg436/teaching.html
- Oliver Lemon’s slides (Heriot-Watt University): https://sites.google.com/site/olemon/conversational-agents
- Pierre Lison’s slides (University of Oslo): https://www.uio.no/studier/emner/matnat/ifi/INF5820/h14/timeplan/
- David Silver’s course on RL (UCL): http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Labs in 10 mins