NPFL099 Statistical Dialogue Systems

6. Dialogue Policy

Ondřej Dušek & Vojtěch Hudeček

http://ufal.cz/npfl099

3. 11. 2020
Dialogue Management

- Two main components:
  - **State tracking** (last lecture)
  - **Action selection/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)
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
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**

- 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

---

**Diagram:**

- States (grey = observed, white = unobserved)
- Actions
- Observations
- Transitions

[Source](https://en.wikipedia.org/wiki/Voronoi_diagram)
Simulated Users

- Static datasets aren’t enough for RL
  - 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/supervised policy from data
  - combination (best!)

(from Milica Gašić's slides)
Summary Space

• 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

(from Milica Gašić's slides)
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

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

- accumulated long-term reward

- 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 \( \pi \)
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**):

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

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

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

• \( Q^\pi(s, a) \) – 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

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

• \( Q^\pi(s, a) \) also defines a greedy policy:

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

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

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

but \( Q \) function itself tends to be more complex than \( V \)
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
**RL Agents Taxonomy**

- **Quantity to optimize:**
  - value function – **critic**
    - either $Q$ or $V$, typically $Q$ in practice
  - policy – **actor**
  - both – **actor-critic**

- **Environment model:**
  - **model-based** (assume known $p(s'|s, a), r(s, a, s)$)
    - nice but typically not satisfied in practice
  - **model-free** (don’t assume anything, sample)
    - this is the usual real-world case
    - this is where using $Q$ instead of $V$ comes handy
Reinforcement Learning Approaches

• How to optimize:
  • **dynamic programming** – find the exact solution from Bellman equation
    • iterative algorithms, refining estimates
    • expensive, assumes known environment → not practical for real-world use
  • **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
    • can’t use that with batch learning (decision policy is changing constantly)
  • **off-policy** – decide according to a different policy
Deep Reinforcement Learning

• Exactly the same as “plain” RL
  • agent & environment, actions & rewards

• “deep” = part of the agent is handled by a NN
  • value function (typically $Q$)
  • policy

• function approximation approach
  • $Q$ values / policy are represented as a parameterized function $Q(s, a; \theta) / \pi(s; \theta)$
  • enumerating in a table would take up too much space, be too sparse
  • the parameters $\theta$ are optimized

• assuming huge state space
  • much fewer weights than possible states
  • update based on one state changes many states

• needs tricks to make it stable

(Sutton & Barto, 2018)
Q-Learning

- temporal difference – update $Q$ as you go
- off-policy – directly estimates best $Q^*$
  - 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 $s'$
    - $Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \left( r + \gamma \cdot \max_a Q(s', a') \right)$
    - $s \leftarrow s'$

$Q^*$ 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 $Q^*(s, a)$: we need to explore to converge

https://towardsdatascience.com/td-in-reinforcement-learning-the-easy-way-f92ecfa9f3ce

Animated example for SARSA & Q-Learning: https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html
Deep Q-Networks

• Q-learning, where $Q$ function is represented by a neural net
• “Usual” Q-learning doesn’t converge well with NNs:
  a) SGD is unstable
  b) correlated samples (data is sequential)
  c) TD updates aim at a moving target (using $Q$ in computing updates to $Q$)
  d) scale of rewards & $Q$ values unknown → numeric instability
• → DQN adds fixes:
  a) minibatches (updates by averaged $n$ samples, not just one)
  b) experience replay
  c) freezing target $Q$ function
  d) clipping rewards

(Mnih et al., 2013, 2015)
http://arxiv.org/abs/1312.5602
http://www.nature.com/articles/nature14236

cool!

common NN tricks
DQN tricks ~ making it more like supervised learning

- **Experience replay** – break correlated samples
  - run through some episodes (dialogues, games…)
  - store all tuples \((s, a, r', s')\) in a buffer
  - for training, don’t update based on most recent moves – use buffer
    - sample minibatches randomly from the buffer
  - overwrite buffer as you go, clear buffer once in a while
  - only possible for off-policy

- **Target Q function freezing**
  - fix the version of Q function used in update targets
    - have a copy of your Q network that doesn’t get updated every time
  - once in a while, copy your current estimate over

\[
\text{loss} := \mathbb{E}_{(s,a,r',s') \in \text{buf}} \left[ (r' + \gamma \max_a Q(s', a'; \overline{\theta}) - Q(s, a; \theta))^2 \right]
\]
DQN algorithm

- initialize $\theta$ randomly
- initialize replay memory $D$ (e.g. play for a while using current $Q(\theta)$)
- repeat over all episodes:
  - set initial state $s$
  - for all timesteps $t = 1 \ldots T$ in the episode:
    - select action $a_t$ from $\varepsilon$-greedy policy based on $Q(\theta)$
    - take $a_t$, observe reward $r_{t+1}$ and new state $s_{t+1}$
    - store $(s_t, a_t, r_{t+1}, s_{t+1})$ in $D$
  - sample a batch $B$ of random $(s, a, r', s')$’s from $D$
  - update $\theta$ using loss $\mathbb{E}_{(s,a,r',s') \in B} \left[ \left( r' + \gamma \max_{a'} Q (s', a'; \overline{\theta}) - Q(s, a; \theta) \right)^2 \right]$ “replay” a. k. a. training (1 update)
  - once every $\lambda$ steps (rarely):
    - $\overline{\theta} \leftarrow \theta$
      update the frozen target function

storing experience
(1 step of Q-learning exploration)
DQN for Atari

- 4-layers:
  - 2x CNN
  - 2x fully connected with ReLU activations

- Another trick:
  - output values for all actions at once
    - vector $Q(s)$ instead of $Q(s, a)$
    - $a$ is not fed as a parameter
  - faster computation

- Learns many games at human level
  - with the same network structure
  - no game-specific features

input: Atari 2600 screen, downsized to 84x84 (grayscale)
4 last frames

values for all actions (joystick moves)

(Mnih et al., 2015)

(input: Atari 2600 screen, downsized to 84x84 (grayscale))

(from David Silver's slides)

https://youtu.be/V1eYniJ0Rnk?t=18
DQN for Dialogue Systems

- DQN can drive dialogue action selection/policy
- **Warm start** needed to make the training actually work:
  - *Pretrain* the network using supervised learning
  - **Replay buffer spiking** – initialize using simple rule-based policy
    - so there are at least a few successful dialogues
    - the RL agent has something to catch on

---

BBQ – Bayes-by-Backprop Q-Networks

• better exploration than $\epsilon$-greedy – explore uncertain regions

• **Bayes-by-Backprop** – probability distribution over network weights
  • start from prior $p(\theta)$, learn posterior $p(\theta|D)$ for training data $D$
  • posterior approximated by Gaussians $q(\theta|w)$, each $\theta_i \sim \mathcal{N}(\mu_i, \sigma_i)$
    • now learning $w_i = \{(\mu_i, \rho_i)\}$ where $\sigma_i = \log(1 + \exp \rho_i)$, to keep $\sigma_i$ positive
    • VAE-style: minimizing KL divergence between $q$ and $p$, reparameterization trick

• using BB to represent DQN + posterior (Thompson) sampling
  • actions sampled acc. to posterior probability that they’re optimal in current state
  • just sample $\theta_t$ from $q$, then choose $a_t = \arg \max_a Q(s_t, a; \theta_t)$

• no need to sample from the frozen target network, just use $\mu$
  • it’s faster, actually more stable
MLP with 2 hidden layers, ReLU, width=256
movie booking task
one-hot dialogue state representation (268 dim)
39 actions (basic `hello()`, `deny()`, `thanks()` etc. + inform/request for each slot)

(enhanced rewards with surprisal)

- $\epsilon$-greedy
- no sampling from frozen network
- sampling from frozen network

(Lipton et al., 2018)
Recurrent Q-Networks

- Joint dialogue tracking & action selection
  - actions are either system DAs or updates to state (DB hypothesis)
  - forced to alternate action types by masking
  - rewards from DB for narrowing down selection
- Models the Q-network as a LSTM
  - or rather LSTM underlying multiple MLPs
    - LSTM maintains internal state representation
  - 1 MLP for system DAs
  - 1 MLP per slot (action=select value X)

(Zhao & Eskenazi, 2016)
http://arxiv.org/abs/1606.02560
Deep Dyna-Q: learning from humans & simulator

• humans are costly, simulators are inaccurate
  • ⇒ learn from both, improve simulator as you go
    • direct RL = learn from users
    • world model learning = improve internal simulator
      • supervised, based on previous dialogues with users
    • planning = learn from simulator
  • DQN, feed-forward policy
  • simulator: feed-forward multi-task net
    • draw a goal uniformly at the start
    • predict actions, rewards, termination
    • use $K$ simulated (“planning”) dialogues per 1 real
  • discriminative DDQ: only use a simulated dialogue if it looks real (according to a discriminator)

(Peng et al., 2018) https://www.aclweb.org/anthology/P18-1203
(Su et al., 2018) https://www.aclweb.org/anthology/D18-1416
Hierarchical RL

• good for multiple subtasks
  • e.g. book a flight to London and a hotel for the same day, close to the airport
• top-level policy: select subtask $g_i$
• low-level policy: actions $a_j, g_i$ to complete subtask $g_i$
  • given initiation/termination conditions
    • keeps on track until terminal state is reached
  • shared by all subtasks (subtask=parameter)
• internal critic (=prob. that subtask is solved)
• global state tracker
  • integrates information from subtasks

(Peng et al., 2017)
http://aclweb.org/anthology/D17-1237
• spatial (slot-based) split instead of temporal
  - doesn’t need defined subtasks & sub-rewards
• belief state representation – features
  - master $\phi_m$, slot-independent $\phi_i$, per-slot $\phi_{sk}$
  - handcrafted (could be neural nets)
  - supports sharing parameters across domains
• two-step action selection:
  1) master action: “slot-dependent or not”?  
     - master policy
  2) primitive action  
     a) slot-independent policy  
     b) slot-specific policies (with shared parameters, distinguished only by belief state)
       - chooses max. $Q$ for all slot-action pairs – involves choosing the slot
• everything is trained using the same global reward signal

(Casanueva et al., 2018)
http://arxiv.org/abs/1803.03232
Summary

- 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)
  - dynamic programming, Monte Carlo, Temporal Difference
  - optimizing value function $V/Q$ (critic), policy (actor), or both (actor-critic)
  - learning on-policy or off-policy (act by the policy you learn/not)
- summary states might be needed
- user simulators: good to use & mix with humans
- DQN – representing & optimizing $Q$ function with a network
  - minibatches, target function freezing, experience replay
- multiple tasks: hierarchical / feudal RL
Thanks

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

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

References/Inspiration/Further:

• Nie et al. (2019): Neural approaches to conversational AI: https://arxiv.org/abs/1809.08267
• Milica Gašić’s slides (Cambridge University): http://mi.eng.cam.ac.uk/~mg436/teaching.html
• Young et al. (2013): POMDP-Based Statistical Spoken Dialog Systems: A Review:

No labs today (project questions?)
Topic deadline: Nov 10
Fixes for datasets required

Next Tue 9:50am: Direct Policy Optimization
Language Generation