NPFL099 Statistical Dialogue Systems

7. Dialogue Policy (2) + Language Generation

http://ufal.cz/npfl099

Ondřej Dušek & Vojtěch Hudeček

10. 11. 2020
Recap from last time: Reinforcement Learning

• RL = find a **policy** that maximizes long-term reward
  • MDP representation: agent in an environment
  • taking **actions**, moving across **states**, getting **rewards**

• optimization approaches:
  • Monte Carlo – sample (a dialogue), then update
  • Temporal Difference – look ahead, refine estimates as you go
  • **actor** (optimize policy directly) **vs. critic** (indirectly via state/action values)

• Q-networks – optimizing indirectly (critic) via \( Q = \text{action-value function} \)
  • \( Q = \text{expected return of taking action } a \text{ in state } s \text{ under policy } \pi \)
  • greedy policy under \( Q \): “choose what’s best for next step according to \( Q \)”
  • if \( Q \) is optimal, its greedy policy is also optimal

• Deep \( Q \) Networks = just represent \( Q \) with a neural net
  • + a few tricks (experience replay, target freezing)
Policy Gradients

- Instead of value functions, train a **network to represent the policy**
  - allows better action sampling – according to actual stochastic policy
    - no need for $\epsilon$-greedy (which is partially random, suboptimal)
- To optimize, we need a **performance metric**: $J(\theta) = V^{\pi_\theta}(s_0)$
  - expected return in starting state when following $\pi_\theta$
  - we want to directly optimize this using gradient ascent

- **Policy Gradient Theorem**:
  - expresses $\nabla J(\theta)$ in terms of $\nabla \pi(a|s, \theta)$

\[
\nabla J(\theta) \propto \sum_s \mu(s) \sum_a Q^\pi(s, a) \nabla \pi(a|s, \theta) = E_\pi \left[ \sum_a Q^\pi(s, a) \nabla \pi(a|s, \theta) \right]
\]

$\mu(s)$ is state probability under $\pi$ – this is the same as expected value $E_\pi$

(Sutton & Barto, 2018; p. 324ff)
REINFORCE: Monte Carlo Policy Gradients

• direct 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 $t = 0,1, ..., T$: $\theta \leftarrow \theta + \alpha \gamma^t R_t \nabla \ln \pi(a_t | s_t, \theta)$

returns $R_t = \sum_{i=t}^{T-1} \gamma^{i-t} r_{i+1}$

variant – advantage instead of returns:
discounting a baseline $b(s)$ (predicted by any model)
$A_t = R_t - b(s_t)$ instead of $R_t$
gives better performance

$V(s)$ is actually a good $b(s)$

this will guarantee the right state distribution/frequency $\mu(s)$

this is stochastic $\nabla J(\theta)$:
• from policy gradient theorem
• using single action sample $a_t$
• expressing $Q^\pi$ as $R_t$ (under $E_{\pi}$)
• using $\nabla \ln x = \frac{\nabla x}{x}$

(Sutton & Barto, 2018; p. 327f)
Policy Gradients (Advantage) Actor-Critic

• REINFORCE + $V$ approximation + TD estimates – better convergence
  • differentiable policy $\pi(a|s, \theta)$
  • differentiable state-value function parameterization $\hat{V}(s, w)$
  • two learning rates $\alpha^\theta, \alpha^w$

• loop forever:
  • set initial state $s$ for the episode
  • for each step $t$ of the episode:
    • sample action $a$ from $\pi(\cdot|s, \theta)$, take $a$ and observe reward $r$ and new state $s'$
    • compute advantage $A \leftarrow r + \gamma \hat{V}(s', w) - \hat{V}(s, w)$
    • update $\theta \leftarrow \theta + \alpha^\theta \gamma^t A \nabla \ln \pi(a|s, \theta)$, $w \leftarrow w + \alpha^w \cdot A \nabla \hat{V}(s, w)$
    • $s \leftarrow s'$

actor (policy update) critic (value function update)

TD: update after each step

same as REINFORCE, except:
• we use $\hat{V}(s, w)$ as baseline
• $r$ is used instead of $R_t$ (TD instead of MC)

(Su et al., 2017)
http://arxiv.org/abs/1707.00130
ACER: Actor-Critic with Experience Replay

- off-policy actor-critic – using **experience replay** buffer
  - same approach as Q learning
  - since ER buffer has past experience with out-of-date policies (using “old” $\tilde{\theta}$), it’s considered off-policy (behaviour policy $\pi_{\tilde{\theta}} \neq$ target policy $\pi_{\theta}$)
    - sampling behaviour from $\pi_{\tilde{\theta}}$ is biased w. r. t. $\pi_{\theta}$
    - correcting the bias – **importance sampling**: multiply by importance weight $\rho_t = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\tilde{\theta}}(a_t|s_t)}$
  - all updates are summed over batches & importance-sampled
    - new objective/performance metric: $\hat{J}_{t} = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\tilde{\theta}}(a_t|s_t)} \hat{A}_t$
      - using advantage instead of returns
      - batch average over timesteps $t$
      - importance sampled

• ACER may be unstable/slow to learn
  • prone to excessively large updates
    - need to set learning rates low
      • high learning rate = unstable, high variance
      • low learning rate = too slow
  • → regularize: **limit KL-divergence change**
    between updated policy $\theta$ & average policy $\bar{\theta}$
    • $\bar{\theta}$ is a moving average of past policies: $\bar{\theta} \leftarrow \alpha \bar{\theta} + (1 - \alpha) \theta$
    • modified policy gradient $g$ is defined as:
      $$
      \min_g \frac{1}{2} \left\| \nabla \theta - g \right\|^2_2 \text{ so that } \nabla KL[\pi_{\bar{\theta}}(s_t) || \pi_{\theta}(s_t)]^T g \leq \xi
      $$
      • minimizing sum of squared differences (L2)
      • i.e. the closest you can get to the gradient, but don’t increase KL between the average and new policy too much
      • quadratic programming, has closed-form solution

Proximal Policy Optimization

- Changing the objective to be more like trust-region
  - without the need to adjust gradients & do the optimization
- Basically clipping the ACER objective
  - define $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\bar{\theta}}(a_t|s_t)}$ – ratio to old params
  - starting from $\hat{E}_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\bar{\theta}}(a_t|s_t)} \hat{A}_t \right] = \hat{E}_t [r_t(\theta)\hat{A}_t]$ (see ACER)
  - using $\hat{E}_t \left[ \min(r_t(\theta)\hat{A}_t, \text{clip}[r_t(\theta)]^{1+\epsilon} \hat{A}_t) \right]$ where
    - original
    - clipped to stay close to 1
- minimum – lower bound on the unclipped objective


NPFL099 L7 2020
Rewards in RL

• Reward function is critical for successful learning
• Handcrafting is not ideal
  • domain knowledge typically needed to detect dialogue success
  • need simulated or paid users,
    can’t learn from users without knowing their task
  • paid users often fail to follow pre-set goals
• Having users provide feedback is costly & inconsistent
  • real users don’t have much incentive to be cooperative
• Learning/optimizing the rewards is desirable
Supervised dialogue quality estimation

- turn features $\rightarrow$ RNN/CNN $\rightarrow$ success/fail or return (multi-class/regression)
  - user & system DA (one-hot)
  - belief state (per-slot prob. distributions)
  - turn number

- trained from data collected by training a DM with a user simulator
  - using handcrafted rewards
  - success/failure & return known
  - acc. >93% on 18k dialogues, ~85-90% on 1k dialogues
    - binary RNN best (not too huge differences)

- used as reward estimator $\geq$ handcrafted
  - similar performance & doesn’t need known goals
  - can learn from real users
  - still ultimately based on handcrafted rewards

(Su et al., 2015)
http://arxiv.org/abs/1508.03386
Interaction Quality

- turns annotated by experts (Likert 1-5)
- trained model (SVM/RNN)
  - very low-level features
  - mostly ASR-related
  - multi-class classification
- result is domain-independent
  - trained on a very small corpus (~200 dialogues)
  - same model applicable to different datasets
- can be used in a RL reward signal
  - works better than task success

## Turn-level Quality Estimation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASRRecognitionStatus</td>
<td>ASR status: success, no match, no input</td>
</tr>
<tr>
<td>ASRConfidence</td>
<td>confidence of top ASR results</td>
</tr>
<tr>
<td>RePrompt?</td>
<td>is the system question the same as in the previous turn?</td>
</tr>
<tr>
<td>ActivityType</td>
<td>general type of system action: statement, question</td>
</tr>
<tr>
<td>Confirmation?</td>
<td>is system action confirm?</td>
</tr>
<tr>
<td>MeanASRConfidence</td>
<td>mean ASR confidence if ASR is success</td>
</tr>
<tr>
<td>#Exchanges</td>
<td>number of exchanges (turns)</td>
</tr>
<tr>
<td>#ASRSuccess</td>
<td>count of ASR status is success</td>
</tr>
<tr>
<td>%ASRSuccess</td>
<td>rate of ASR status is success</td>
</tr>
<tr>
<td>#ASRRejections</td>
<td>count of ASR status is reject</td>
</tr>
<tr>
<td>%ASRRejections</td>
<td>rate of ASR status is reject</td>
</tr>
<tr>
<td>{Mean}ASRConfidence</td>
<td>mean ASR confidence if ASR is success</td>
</tr>
<tr>
<td>#{}ASRSuccess</td>
<td>count of ASR is success</td>
</tr>
<tr>
<td>#{}ASRRejections</td>
<td>count of ASR is reject</td>
</tr>
<tr>
<td>#{}RePrompts</td>
<td>count of times RePrompt? is true</td>
</tr>
<tr>
<td>#{}SystemQuestions</td>
<td>count of ActivityType is question</td>
</tr>
</tbody>
</table>

“reject” = ASR output doesn’t match in-domain LM

(Schmitt & Ultes, 2015; Ultes et al., 2017; Ultes, 2019)

https://doi.org/10.1016/j.specom.2015.06.003
https://doi.org/10.21437/Interspeech.2017-1032
https://aclweb.org/anthology/W19-5902/
Reward as discriminator

• no predefined rewards, learn from data
  • known success, but learned reward for it
  • success = match user slot values & provide all requested information

• discriminator: LSTM + max-pooling
  • classify 1/0 successful (from dataset) vs. simulated over whole dialogue

• dialogue manager
  • LSTM tracker & feed-forward policy in a single model

• supervised pretraining + GAN-style training
  • supervised reward learning = “inverse RL”
  • DM: REINFORCE with rewards from discriminator
  • discriminator: sample with current DM & train to classify successful vs. simulated

(Liu & Lane, 2018) http://arxiv.org/abs/1805.11762
Reward as discriminator

• comparing rewards
  - **oracle** = 1/0 successful/failed
  - **designed** = +1 for each correct slot, +1 for each informed request (with correct slots)
  - **pretrained** = without the GAN training
  - **adversarial** = full setup with GAN training
    • adversarial better than handcrafted

• can also learn from partial user feedback
  • counters disadvantage for dialogues different from previous policy

• use discriminator if feedback is not available
  • further slight improvement

---

Turn-level adversarial rewards

- discriminator: policy vs. human-human
  - irrespective of success → can be done on turn level
- policy $\pi$ & reward estimator $f$ are feed-forward
  - ReLU, 1 hidden layer
- still the same process:
  - pretrain both $\pi$ & $f$ using supervised learning
  - sample dialogs using $\pi$
  - update $f$ to distinguish sampled vs. human-human
  - update $\pi$ using rewards provided by $f$
- using proximal policy optimization to update $\pi$
- using 2 different user simulators
  - provides more diversity


2 simulators:
- agenda/rules
- seq2seq

domains

(previous slide)

(this model)
Alternating supervised & RL

• we can do better than just supervised pretraining

• alternate regularly
  • start with supervised more frequently
    • alleviate sparse rewards, but don’t completely avoid exploring
  • later do more RL
    • but don’t forget what you learned by supervised learning

• options:
  • schedule supervised every $N$ updates
  • same + increase $N$ gradually
  • use supervised after RL does poorly (worse than baseline)
    • baseline = moving average over history + $\lambda \cdot$ std. error of the average
    • agent is less likely to be worse than baseline in later stages of learning

(Xiong et al., 2018)
Natural Language Generation

• conversion of system action semantics → text (in our case)
• NLG output is well-defined, but input is not:
  • DAs
  • any other semantic formalism
  • database tables
  • raw data streams
  • user model
  • dialogue history
  can be any kind of knowledge representation
  e.g. “user wants short answers”
  e.g. for referring expressions, avoiding repetition
• general NLG objective:
  given input & communication goal,
  create accurate + natural, well-formed, human-like text
• additional NLG desired properties:
  • variation
  • simplicity
  • adaptability
NLG Subtasks (textbook pipeline)

• Inputs
  • ↓ Content/text/document planning
    • content selection according to communication goal
    • basic structuring & ordering
  • Content plan
  • ↓ Sentence planning/microplanning
    • aggregation (facts → sentences)
    • lexical choice
    • referring expressions
      e.g. restaurant vs. it
  • Sentence plan
  • ↓ Surface realization
    • linearization according to grammar
    • word order, morphology
  • Text

typically handled by dialogue manager in dialogue systems
organizing content into sentences & merging simple sentences
this is needed for NLG in dialogue systems
NLG Basic Approaches

- **canned text**
  - most trivial – completely hand-written prompts, no variation
  - doesn’t scale (good for DTMF phone systems)

- **templates**
  - “fill in blanks” approach
  - simple, but much more expressive – covers most common domains nicely
  - can scale if done right, still laborious
  - most production dialogue systems

- **grammars & rules**
  - grammars: mostly older research systems, realization
  - rules: mostly content & sentence planning

- **machine learning**
  - modern research systems
  - pre-neural attempts often combined with rules/grammar
  - NNs made it work much better
Template-based NLG

- Most common in dialogue systems
  - especially commercial systems
- Simple, straightforward, reliable
  - custom-tailored for the domain
  - complete control of the generated content
- Lacks generality and variation
  - difficult to maintain, expensive to scale up
- Can be enhanced with rules
  - e.g. articles, inflection of the filled-in phrases
  - template coverage/selection rules, e.g.:
    - select most concrete template
    - cover input with as few templates as possible
    - random variation

(Facebook, 2015)

inflection rules

(Alex public transport information rules)
https://github.com/UFAL-DSG/alex
Neural End-to-End NLG: RNNLG

- Unlike previous, doesn’t need alignments
  - no need to know which word/phrase corresponds to which slot
- Using RNNs, generating word-by-word
  - neural language models conditioned on DA
  - generating delexicalized texts
- input DA represented as binary vector
- Enhanced LSTM cells (SC-LSTM)
  - special part of the cell (gate) to control slot mentions

(Wen et al, 2015; 2016)
http://aclweb.org/anthology/D15-1199
http://arxiv.org/abs/1603.01232
Seq2seq NLG (TGen)

• Standard seq2seq with attention
  • encoder – triples <DA type, slot, value>
  • decodes words (possibly delexicalized)

• Beam search & reranking
  • DA classification of outputs
  • checking against input DA

(Dušek & Jurčiček, 2016)
https://aclweb.org/anthology/P16-2008
Delexicalization vs. Copy/Pointer net

• Most models still use it
  • preprocess/postprocess step – names to <placeholders>
  • generator works with template-like stuff

• Alternative – **copy mechanisms** (see NLU)
  • generate or point & copy from input
  • does away with the pre/postprocessing

• Czech & other languages with rich morphology
  • basic delexicalization or copy don’t work
    • nouns need to be inflected
      (unlike English, where they only have 1 form)
  • basically another step needed: **inflection model**
    • one option: RNN LM

**NPFL099 L7 2020**

Baráčnická rychta je na <area>

inform(name=Baráčnická rychta, area=Malá Strana)

<table>
<thead>
<tr>
<th>Case</th>
<th>Example</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>nominative</td>
<td>Malá Strana</td>
<td>0.10</td>
</tr>
<tr>
<td>genitive</td>
<td>Malé Strany</td>
<td>0.07</td>
</tr>
<tr>
<td>dative, locative</td>
<td>Malé Straně</td>
<td>0.60</td>
</tr>
<tr>
<td>accusative</td>
<td>Malou Stranu</td>
<td>0.10</td>
</tr>
<tr>
<td>instrumental</td>
<td>Malou Stranou</td>
<td>0.03</td>
</tr>
</tbody>
</table>

(Shi et al., 2018)  [http://arxiv.org/abs/1812.02303](http://arxiv.org/abs/1812.02303)
Ensembling

• “two heads are better than one” – use more models & aggregate
  • common practice in neural models elsewhere in NLP
• base version: same model, different random initializations
• getting diverse predictions: use different models
  • different architectures – e.g. CNN vs. LSTM encoder
  • different data – diverse ensembling
    • cluster training data & train different models on different portions
  • clustering & training can be done jointly:
    • assign into groups randomly/train $k$ models for 1 iteration
    • check prob. of each training instance under each model
    • reassign to model that predicts it with highest probability

(Juraska et al., 2018)  http://arxiv.org/abs/1805.06553
(Gehrmann et al., 2018)  https://www.aclweb.org/anthology/W18-6505
Ensembling

• combine predictions from multiple models:
  • just use the model that’s best on development data
    • won’t give diverse outputs, but may give better quality
  • compose n-best list from predictions of all models
    • n-best lists are more diverse
    • assuming reranking (e.g. checking against input DA)
  • vote on the next word at each step / average predicted word distributions
    • & force-decode chosen word with all models
    • this is rather slow
    • might not even work:
      • each model may expect different sentence structures, combination can be incoherent
Problems with neural NLG

• Checking the semantics
  • neural models tend to forget / hallucinate (make up irrelevant stuff)
  • reranking works currently best to mitigate this, but it’s not perfect
• Delexicalization needed (at least some slots)
  • otherwise the data would be too sparse
  • alternative: copy mechanisms
• Diversity & complexity of outputs
  • still can’t match humans by far
  • needs specific tricks to improve this
    • vanilla seq2seq models tend to produce repetitive outputs
• Still more hassle than writing up templates 😞
Summary

• Policy optimization
  • optimizing directly (Policy Gradient Theorem)
  • REINFORCE = Monte Carlo policy gradients
  • advantage = return – baseline
  • policy gradients actor-critic = REINFORCE + TD + state value estimates
  • ACER (actor-critic with experience replay) + extensions

• RL rewards: critical for good performance & can be (partially) learned

• NLG: system DA → text
  • templates work pretty well
  • seq2seq & similar = best data-driven
    • problems: hallucination, not enough diversity
    • fixes: reranking, delexicalization/copy nets, ensembling
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:

- David Silver’s course on RL (UCL): http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- Milan Straka’s course on RL (Charles University): http://ufal.mff.cuni.cz/courses/npfl122/

No labs today (project questions?)
Topic deadline – today!

No class next week (holiday)

24 November: rest of NLG + hints on your experiments