9. Neural Dialogue Management & Natural Language Generation

Ondřej Dušek & Ondřej Plátek & Jan Cuřín

ufal.cz/npfl123

16. 4. 2019
Deep Reinforcement Learning

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

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

• NN = function approximation approach
  • such as REINFORCE / policy gradients
  • NN → complex non-linear functions

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

(Sutton & Barto, 2018)
Value Function Approximation

• Searching for approximate \( V(s) \) or \( Q(s, a) \)
  • exact values are too big to enumerate in a table
  • **parametric approximation** \( V(s; \theta) \) or \( Q(s, a; \theta) \)

• Regression: **Mean squared value error**
  • weighted over states’ importance
  • useful for gradient descent
  • \( \rightarrow \sim \text{ any supervised learning approach possible} \)
    • not all work well though

• MC = stochastic gradient descent

• TD is **semi-gradient** (not true gradient descent)
  • \( \leftarrow \) using current weights in target estimate
  • we still want TD over MC for speed
  • guaranteed convergence for linear approximations
  • unstable for NNs!

\[
\text{Mean squared value error} \quad \overline{VE}(\theta) := \sum_{s \in S} \mu(s)(V_\pi(s) - V(s, \theta))^2
\]

- states’ importance weight (probability distribution)
- target value (which we don’t have!)
  \( \rightarrow \) using \( R_t \) in MC
  \( \rightarrow \) using \( r_{t+1} + \gamma V(s', \theta) \)
Deep Q-Networks  
(Mnih et al., 2013, 2015)

• Q-learning with function approximation  
  • $Q$ function represented by a neural net

• Causes of poor convergence in basic Q-learning 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

• Fixes in DQN:
  a) minibatches (updates by averaged $n$ samples, not just one)
  b) experience replay  
  c) freezing target Q function  
  d) clipping rewards

(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

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

“generate your own ‘supervised’ training data”

“have a fixed target, like in supervised learning”
DQN algorithm

- initialize $\theta$ randomly
- initialize replay memory $D$ (e.g. play for a while using current $Q(\theta)$)
- repeat over all episodes:
  - for episode, set initial state $s$
    - select action $a$ from $\epsilon$-greedy policy based on $Q(\theta)$
    - take $a$, observe reward $r'$ and new state $s'$
    - store $(s, a, r', s')$ in $D$
    - $s \leftarrow s'$
  - once every $k$ steps:
    - sample a batch $B$ of random $(s, a, r', s')$’s from $D$
    - update $\theta$ using loss $\mathbb{E}_{(s,a,r',s') \in B} \left[ \left( r' + \gamma \max_{a'} Q(s', a'; \overline{\theta}) - Q(s, a; \theta) \right)^2 \right]$
  - once every $\lambda$ steps:
    - $\overline{\theta} \leftarrow \theta$

often $\rightarrow$ once every $k$ steps:
  - “replay” a. k. a. training

rarely $\rightarrow$ once every $\lambda$ steps:
  - storing experience
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)

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

• a simple DQN can drive a dialogue system’s action selection
  • DQN is function approximation – works fine for POMDPs
  • no summary space tricks needed here

rule-based simulator with agenda running on DA level

DQN – feed-forward, 1 hidden ReLU layer

error model controller (simulating ASR/NLU noise)

movie ticket booking: better than rule-based

replay memory initialized using a simple handcrafted policy

NPFL123 L9 2019
Policy Networks

• Learning policy directly – **policy network**
  • can work better than Q-learning
  • NN: input = state, output = prob. dist. over actions
  • actor-critic: network predicts both $\pi$ and $V/Q$

• Training can’t use/doesn’t need the DQN tricks
  • just REINFORCE with baseline / actor-critic
    • reward – baseline = **advantage**
    • these are on-policy → no experience replay
      • minibatches used anyway
    • extension: parallel training (A3C algorithm)
      • sample in multiple threads, gather gradients
      • better speed, more diverse experience

---

https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2
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

• general NLG objective:
  - **given input & communication goal**
  - **create accurate + natural, well-formed, human-like text**

• additional NLG desired properties:
  - variation
  - simplicity
  - adaptability
NLG Use Cases

• **dialogue systems**
  • very different for task/non-task-oriented/QA systems

• **standalone**
  • data-to-text
  • short text generation for web & apps
    • weather, sports reports
    • personalized letters

• **machine translation**
  • now mostly integrated end-to-end
  • formerly not the case

• **summarization**
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

deciding what to say

deciding how to say it
NLG Implementations

• Few systems implement the whole pipeline
  • All stages: mostly domain-specific data-to-text, standalone
    • e.g. weather reports
  • Dialogue systems: just sentence planning + realization
  • Systems focused on content + sentence planning with trivial realization
    • frequent in DS: focus on sentence planning, trivial or off-the-shelf realizer
  • Surface realization only
    • requires very detailed input
    • some systems: just ordering words

• Pipeline vs. end-to-end approaches
  • planning + realization in one go – popular for neural approaches
  • pipeline: simpler components, might be reusable (especially realizers)
  • end-to-end: no error accumulation, no intermediate data structures
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
  - RNNs 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

(Alex public transport information rules)
https://github.com/UFAL-DSG/alex

(Facebook, 2015)

(Facebook, 2019)

inflection rules
Trainable Sentence Planning: Overgenerate & Rerank

- Assuming you have a flexible handcrafted planner
  - underspecified grammar
  - rules with multiple options...
- Generate multiple outputs
- Select the best one
  - train just the selection – learning to rank
  - any supervised approach possible
    a) “top” = 1, “not top” = 0
    b) loss incurred by relative scores
     \[ \text{loss} = \max(0, \text{“not top”} - \text{“top”}) \]

SpoT trainable planner
(RankBoost ranking)
Trainable Sentence Planning: Parameter Optimization

- Assuming you have a flexible handcrafted planner
  - + one that has **configurable parameters**, for e.g.:
    - sentence aggregation
    - fillers
    - lexical choices

- Train the best parameters for your task
  - generate under different settings
  - annotate the outputs with linguistic features
  - learn classifiers: linguistic features → generator settings
    - any supervised learning
    - can predict the settings jointly/independently

(Mairesse & Walker, 2008; 2011)
https://www.aclweb.org/anthology/P08-1020
https://www.aclweb.org/anthology/J11-3002
Grammar-based realizers

- Various grammar formalisms
  - production / unification rules in the grammar
- typically general-domain, reusable
- KPML – multilingual
  - systemic functional grammar
- FUF/SURGE – English
  - functional unification grammar

KPML sentence plan for *A dog is in the park.*

(10 / spatial-locating
  :speechact (a0 / assertion :polarity positive :speaking-time t0)
  :reference-time-id t0
  :event-time (t0 / time)
  :theme d0
  :domain (d0 / object :lex dog
    :identifiability-q notidentifiable)
  :range (p0 / three-d-location :lex park
    :identifiability-q identifiable))

(Bateman, 1997)

FUF/SURGE input and output

Input Specification ($I_1$):

Output Sentence ($S_1$): “She hands the draft to the editor”

(Elhadad & Robin, 1996)
https://academiccommons.columbia.edu/doi/10.7916/D83T9RG1/download
Grammar-based Realizers: OpenCCG

- **OpenCCG** – English
  - combinatory categorial grammar
  - reuse/reverse of CCG parser
  - (reverse) lexical lookup
  - combination according to grammar – dynamic programming
  - statistical enhancements

OpenCCG input for flight information

```plaintext
be [tense=pres info=th id=n1]
<Arg> flight [num=sg det=the info=th id=f2]
<HasProp> cheapest [kon=+ id=n2]
<Prop> has-rel [id=n3]
<of> f2
<Airline> Ryanair [kon=+ id=n4]
```

(Moore et al., 2004)

NPFL123 L9 2019 19

OpenCCG input

\[
\begin{align*}
\phi_e (\text{man} \land (\text{GENREL}) (e \land \text{see} \land (\text{TENSE}) \text{past} \\
\land (\text{ACT}) (b \land \text{Bob}) \land (\text{PAT}) x) \\
0 : \phi_e \text{man}, 1 : \phi_e (\text{GENREL}) e, 2 : \phi_e \text{see} \\
3 : \phi_e (\text{TENSE}) \text{past}, 4 : \phi_e (\text{ACT}) b \\
5 : \phi_e (\text{PAT}) x, 6 : \phi_e \text{Bob}
\end{align*}
\]

OpenCCG lexical lookup

\[
\{2, 3, 4, 5\} \{e, b, x\}
\]

saw \(\vdash\) \(\phi_e (\text{np}) / \text{np}_e : \phi_e \text{see} \land \phi_e (\text{TENSE}) \text{past} \land \phi_e (\text{ACT}) b \land \phi_e (\text{PAT}) x\)

\[
\{2, 4, 5\} \{e, b, x\}
\]

see \(\vdash\) \(\phi_e (\text{np}) / \text{np}_e : \phi_e \text{see} \land \phi_e (\text{ACT}) b \land \phi_e (\text{PAT}) x\)

\[
\{1\} \{e, x\}
\]

that \(\vdash\) \((\phi_e (\text{np}) / \phi_e (\text{np}_e)) : \phi_e (\text{GENREL}) e\)

\[
\{1\} \{e, x\}
\]

that \(\vdash\) \((\phi_e (\text{np}) / \phi_e (\text{np}_e)) : \phi_e (\text{GENREL}) e\)

(White & Baldridge, 2003)
https://www.aclweb.org/anthology/W03-2316

OpenCCG parsing (combinatory rules)

Bob \(\vdash\) \(\phi_e (\text{np}) : \phi_e \text{Bob}\)

to see \(\vdash\) \(\phi_e (\text{np}) / \text{np}_e : \phi_e \text{see} \land \phi_e (\text{ACT}) b \land \phi_e (\text{PAT}) x\)

Bob saw \(\vdash\) \(\phi_e (\text{np}) : \phi_e \text{see} \land \phi_e (\text{TENSE}) \text{past} \\
\land \phi_e (\text{ACT}) b \land \phi_e (\text{PAT}) x \land \phi_e \text{Bob}\)

Bob to see \(\vdash\) \(\phi_e (\text{np}) : \phi_e \text{see} \land \phi_e (\text{ACT}) b \land \phi_e (\text{PAT}) x \land \phi_e \text{Bob}\)

man that Bob saw \(\vdash\) \(\phi_e (\text{np}) : \phi_e \text{man} \land \phi_e (\text{GENREL}) e \\
\land \phi_e \text{see} \land \phi_e (\text{TENSE}) \text{past} \\
\land \phi_e (\text{ACT}) b \land \phi_e (\text{PAT}) x \land \phi_e \text{Bob}\)
Procedural realizer: SimpleNLG

- A simple Java API
  - “do-it-yourself” style – only cares about the grammar
  - input needs to be specified precisely
  - building up ~syntactic structure
  - final linearization
- built for English
  - large coverage lexicon included
  - ports to multiple languages available

SimpleNLG generation procedure

```
Lexicon lexicon = new XMLLexicon("my-lexicon.xml");
NLGFactory nlgFactory = new NLGFactory(lexicon);
Realiser realiser = new Realiser(lexicon);

SPhraseSpec p = nlgFactory.createClause();
p.setSubject("Mary");
p.setVerb("chase");
p.setObject("the monkey");
p.setFeature(Feature.TENSE, Tense.PAST);

String output = realiser.realiseSentence(p);
System.out.println(output);
```

>>> Mary chased the monkey.

(Gatt & Reiter, 2009)
https://www.aclweb.org/anthology/W09-0613
Grammar/Procedural Realizer: RealPro

• Also procedural, but built on a grammar formalism
  • Meaning-Text Theory
• Pipeline, working through different levels of meaning description
  • deep syntax / semantics
  • surface syntax
  • morphology

RealPro input (textual/graphical representation)

for Do these boys see Mary?

```
SEE [ question:+ ]
( I boy [ number:pl ]
  ( ATTR THIS1 )
  II Mary [ class:proper_noun ] )
```

for This boy sees Mary

```
see

boy

ATTR

THIS1

Mary
```
Grammar/Procedural Realizer: TectoMT/Treex

• Similar to RealPro
  • based on Functional Generative Description (a.k.a. tectogrammatics)
  • deep syntax → surface syntax → morphology and linearization
  • English, Czech, Dutch, Spanish, Basque

• Simple Perl program:
  • copy deep syntax
  • fix morphology agreement
  • add prepositions, conjunctions & articles
  • add auxiliary verbs
  • inflect words
  • add punctuation & capitalization

(Popel & Žabokrtský 2010; Dušek et al., 2015)
https://www.aclweb.org/anthology/W15-3009
Trainable Realizers

• **Overgenerate & Rerank**
  • same approach as for sentence planning
  • assuming a flexible handcrafted realizer (e.g., OpenCCG)
  • underspecified input → more outputs possible
  • generate more & use statistical reranker, based on:
    • n-gram language models
    • Tree language models
    • expected text-to-speech output quality
    • personality traits & alignment/entrainment
  • more variance, but at computational cost

• **Grammar/Procedural-based**
  • same as RealPro or TectoMT, but predict each step using a classifier

This means the grammar may be smaller.
Non-neural End-to-End NLG: Language Models

• hierarchy of n-gram models
  • slot level (which slot follows which) & word level (words in the phrase for current slot)
  • limited history, no long-range dependencies
  • beam & reranking (sanity checks)

• hierarchy of maximum entropy models
  • unlimited history
  • conditioned also on higher-level decisions

• factored language models
  • conditioned on various features
  • global search for best sequence

Markov Model

MEMM style (Angeli et al., 2010) https://www.aclweb.org/anthology/D10-1049
(Liang et al., 2009) https://www.aclweb.org/anthology/P09-1011

CRF style (not completely) (Oh & Rudnicky, 2002) https://doi.org/10.1016/S0885-2308(02)00012-8

BAGEL (Mairesse et al., 2010; Mairesse & Young, 2014)
Non-neural End-to-End NLG: NLG as Parsing

- Probabilistic CFG
  - base handcrafted generator
  - rules chosen based on corpus probability

- PCFG with generic rules
  - domain independent (~DA → slots → values)
  - approx. search for best derivation – bottom-up n-best

- Synchronous CFGs – aligned MR & text CFGs
  - “translation” with hierarchical phrase models
  - parsing MR & synchronously generating text
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

Loch Fyne is a kid-friendly restaurant serving cheap Japanese food.

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

NPFL123 L9 2019
Seq2seq NLG (TGen)

- 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
Problems with neural NLG

- Checking the semantics
  - neural models tend to forget / make up irrelevant stuff
  - reranking currently best, but 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
  - needs specific tricks to improve this
- Still more hassle than writing up templates 😞

(Dušek et al., 2019)
http://arxiv.org/abs/1901.07931

open sets, verbatim on the output (e.g., restaurant/area names)
Summary

• Deep Reinforcement Learning
  • same as plain RL – agent + states, actions, rewards – just $Q$ or $\pi$ is a NN
  • function approximation for $Q$ – mean squared value error
  • Deep Q Networks – Q learning where $Q$ is a NN + tricks
    – experience replay, target function freezing
  • Policy networks – policy gradients where $\pi$ is a NN

• Natural Language Generation
  • steps: content planning, sentence planning, surface realization
    – not all systems implement everything (content planning is DM’s job in DS)
    – pipeline vs. end-to-end
  • approaches: templates, grammars, statistical
  • templates work great
  • state-of-the-art = seq2seq with reranking
Thanks

Contact me:
odusek@ufal.mff.cuni.cz
room 424 (but email me first)

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

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/
• Deep RL for NLP tutorial