NPFL123 Dialogue Systems
7. Neural Policies & Natural Language Generation

https://ufal.cz/npfl123

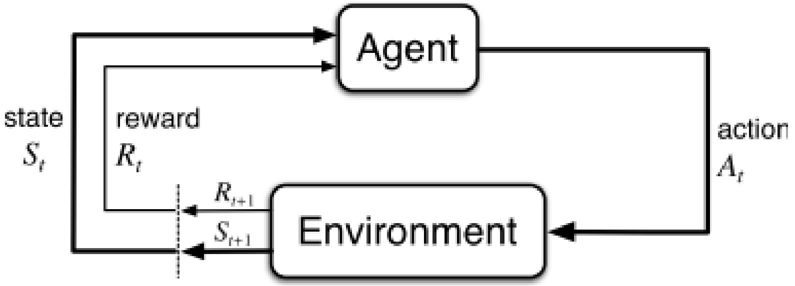
Ondřej Dušek, Mateusz Lango, Ondřej Plátek & Jan Cuřín

3. 4. 2024

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Deep Reinforcement Learning

- Exactly the same as "plain" RL
- **"deep" = part of the agent is handled by a NN**
  - value function (typically Q)
  - policy
- NN = parametric function approximation approach
  - NN → complex non-linear functions
  - **REINFORCE / policy gradients: $\pi(a|s,\boldsymbol{\theta})$ – works out of the box**
  - value functions: using $V(s;\boldsymbol{\theta})$ **or** $Q(s,a;\boldsymbol{\theta})$, regression
- assuming huge state space
  - much fewer weights than possible states
  - update based on one state changes many states
  - no more summary space ☺



(Sutton & Barto, 2018)

Deep Q-Networks

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

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

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

"generate your own 'supervised' training data"

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

- **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

"have a fixed target, like in supervised learning"

DQN algorithm

- initialize $\boldsymbol{\theta}$ randomly
- initialize replay memory D (e.g. play for a while using current $Q(\boldsymbol{\theta})$)
- repeat over all episodes:
  - set initial state s
  - for all timesteps $t = 1 \dots T$ in the episode:
    - select action a_t from ϵ-greedy policy based on $Q(\boldsymbol{\theta})$
    - 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

    storing experience
    (1 step of Q-learning exploration)

    - sample a batch B of random (s, a, r', s')'s from D
    - update $\boldsymbol{\theta}$ using loss $\mathbb{E}_{(s,a,r',s')\in B}\left[\left(r' + \gamma \max_{a'} Q\left(s', a'; \overline{\boldsymbol{\theta}}\right) - Q(s, a; \boldsymbol{\theta})\right)^2\right]$

    "replay"
    a. k. a. training
    (1 update)

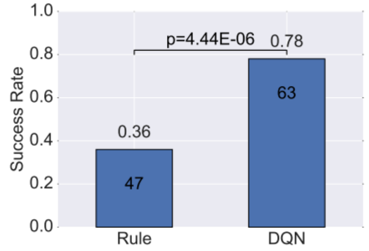
  - once every λ steps (rarely):
    - $\overline{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$

update the frozen target function

DQN for Dialogue Systems

(Li et al., 2017)
https://arxiv.org/abs/1703.01008
https://github.com/MiuLab/TC-Bot

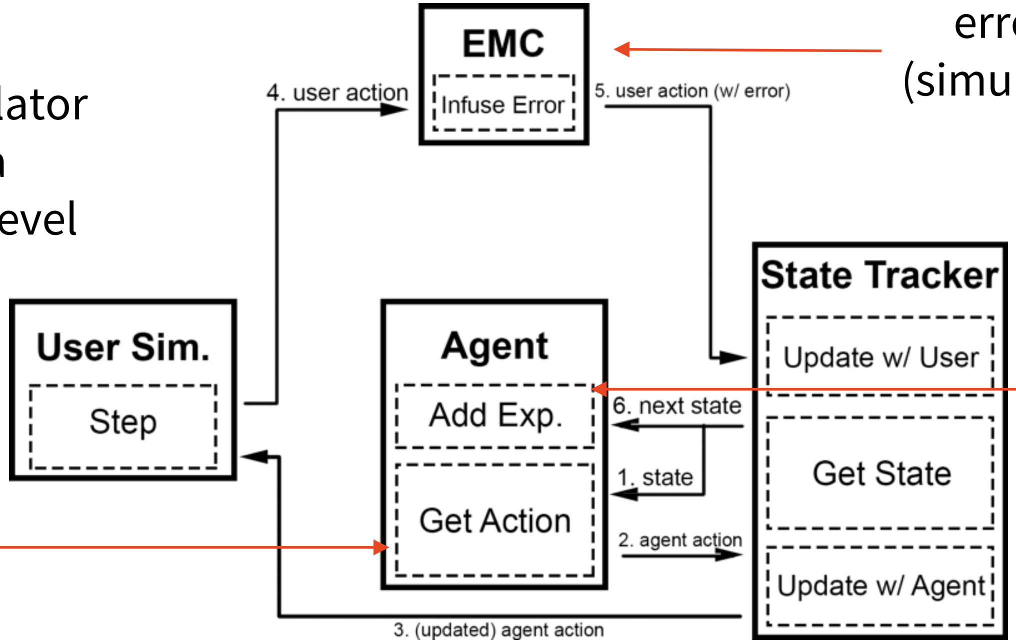
- a simple DQN can drive a dialogue system's action selection
  - DQN is function approximation – works fine for POMDPs
  - no summary space tricks needed here



movie ticket booking: better than rule-based

rule-based simulator with agenda running on DA level

error model controller (simulating ASR/NLU noise)



DQN – feed-forward, 1 hidden ReLU layer

replay memory initialized using a simple handcrafted policy

https://towardsdatascience.com/training-a-goal-oriented-chatbot-with-deep-reinforcement-learning-part-i-introduction-and-dce3af21d383

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 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
  - creative generation (stories)
- **machine translation**
  - now mostly integrated end-to-end
  - formerly not the case
- **summarization**

NLG Subtasks (textbook pipeline)

deciding what to say

deciding how to say it

Inputs

- ↓ **Content/text/document planning**
  - content selection according to communication goal
  - basic structuring & ordering

typically handled by dialogue manager in dialogue systems

Content plan

- ↓ **Sentence planning/microplanning**
  - aggregation (facts → sentences)
  - lexical choice
  - referring expressions

organizing content into sentences & merging simple sentences

e.g. *restaurant* vs. *it*

Sentence plan

- ↓ **Surface realization**
  - linearization according to grammar
  - word order, morphology

Text

this is needed for NLG in dialogue systems

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

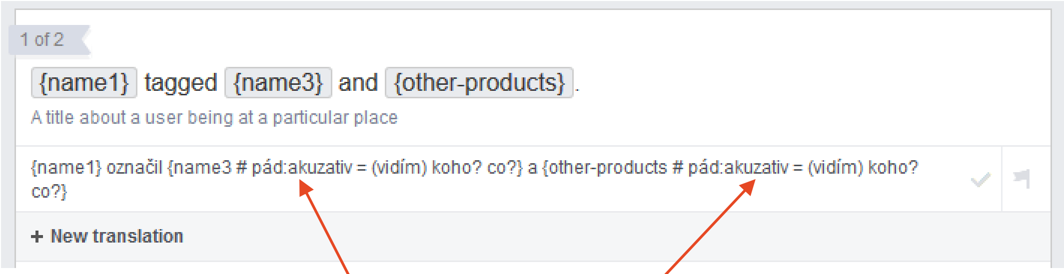
NLG Basic Approaches

- **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
  - neural nets 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)



(Facebook, 2019)

inflection rules

```
'iconfirm(to_stop={to_stop})&iconfirm(from_stop={from_stop})':
    "Alright, from {from_stop} to {to_stop},",

'iconfirm(to_stop={to_stop})&iconfirm(arrival_time_rel="{arrival_time_rel}")':
    "Alright, to {to_stop} in {arrival_time_rel},",

'iconfirm(arrival_time="{arrival_time}")':
    "You want to be there at {arrival_time},",

'iconfirm(arrival_time_rel="{arrival_time_rel}")':
    "You want to get there in {arrival_time_rel},",
```

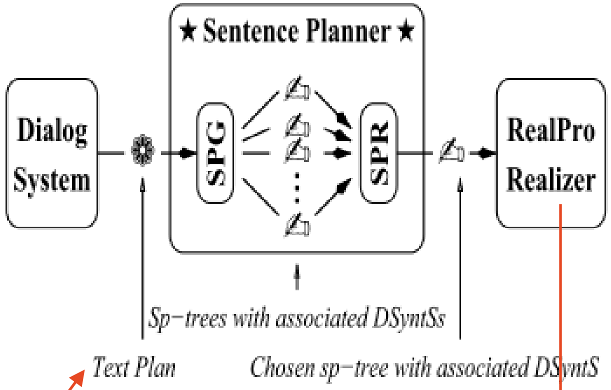
(Alex public transport information rules)
https://github.com/UFAL-DSG/alex

Grammar/Rules for Sentence Planning

- Handcrafted grammar/rules
  - input: base semantics (e.g. dialogue acts)
  - output: detailed sentence representation (=realizer inputs, see →)
- Statistical enhancements:
  generate more options & choose the best
  - generate multiple outputs
    - underspecified grammar
    - rules with multiple options…
  - choose the best one
    - train just the selection – learning to rank
    - any supervised approach possible
      e.g. “best” = 1, “not best” = 0

NB: this is slow!

SpoT trainable planner
(RankBoost ranking)



input DA

```
implicit-confirm(orig-city:NEWARK)
implicit-confirm(dest-city:DALLAS)
implicit-confirm(month:9)
implicit-confirm(day-number:1)
request(depart-time)
```

Human | RankBoost

Alt	Realization	H	RB
0	What time would you like to travel on September the 1st to Dallas from Newark?	5	.85
5	Leaving on September the 1st. What time would you like to travel from Newark to Dallas?	4.5	.82
8	Leaving in September. Leaving on the 1st. What time would you, traveling from Newark to Dallas, like to leave?	2	.39

(Walker et al., 2001)
https://www.aclweb.org/anthology/N01-1003

Grammar-based realizers

- Various grammar formalisms
  - production / unification rules in the grammar
  - lexicons to go with it
  - expect very detailed input (*sentence plans*)
- typically general-domain, reusable
  - **KPML** – multilingual
    - systemic functional grammar
  - **FUF/SURGE** – English
    - functional unification grammar
  - **OpenCCG** – English
    - combinatory categorial grammar

KPML input for *A dog is in the park.*

```
(l0 / 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))
```

FUF/SURGE input for *She hands the draft to the editor*

$$\begin{bmatrix} cat & clause \\ process & \begin{bmatrix} type & composite \\ relation & possessive \\ lex & \text{``}hand\text{''} \end{bmatrix} \\ partic & \begin{bmatrix} agent & \begin{bmatrix} cat & pers\_pro \\ gender & feminine \end{bmatrix} \\ affected & \boxed{1} \begin{bmatrix} cat & np \\ lex & \text{``}editor\text{''} \end{bmatrix} \\ possessor & \boxed{1} \\ possessed & \begin{bmatrix} cat & np \\ lex & \text{``}draft\text{''} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

OpenCCG input for *The cheapest flight is on Ryanair*

```
be [tense=pres info=rh 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]
```


(Bateman, 1997) http://www.academia.edu/download/3459017/bateman97-jnle.pdf
(Elhadad & Robin, 1996) https://academiccommons.columbia.edu/doi/10.7916/D83T9RG1/download
(White & Baldridge, 2003) https://www.aclweb.org/anthology/W03-2316
(Moore et al., 2004) http://www.aaai.org/Papers/FLAIRS/2004/Flairs04-155.pdf

Procedural realizers

SimpleNLG

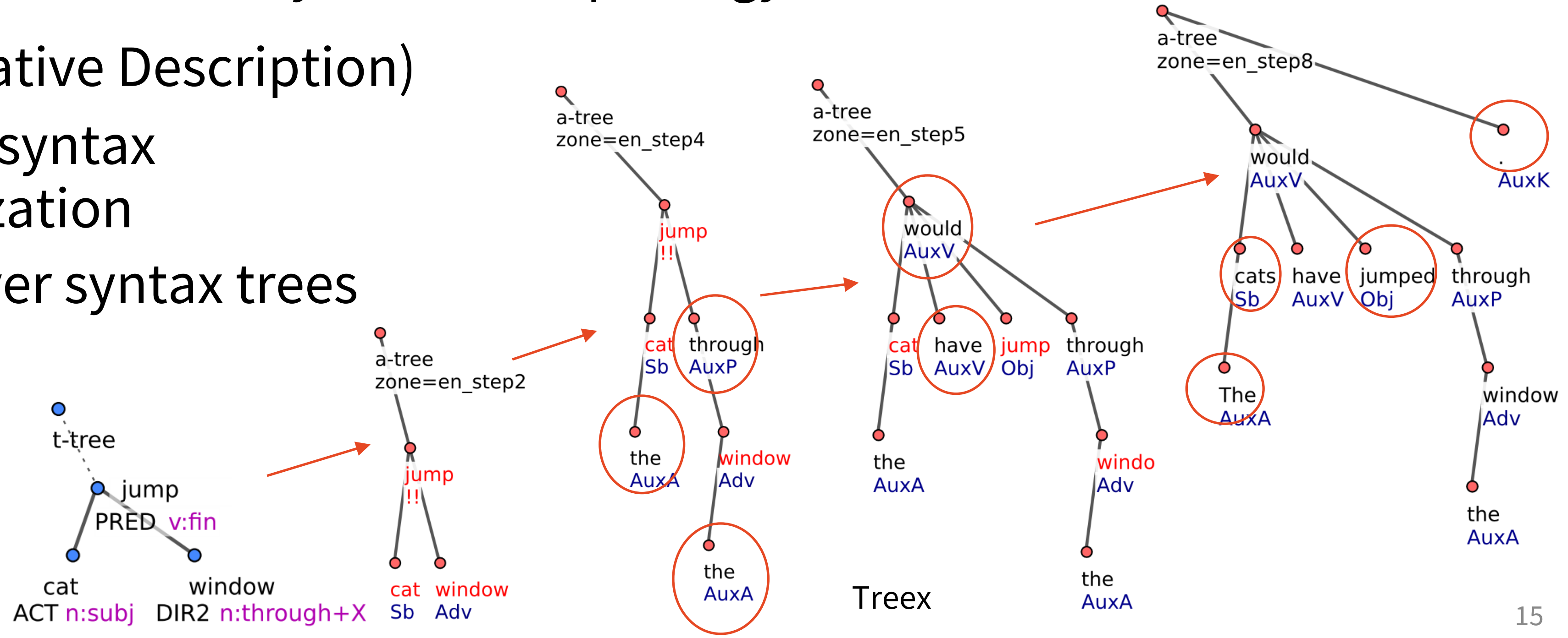
```
Lexicon lexicon = new XMLLexicon("my-lexicon.xml");
NLGFactory nlgFactory = new NLGFactory(lexicon);
Realiser realiser = new Realiser(lexicon);

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.
```

- **SimpleNLG** – no grammar, code to build sentence
  - “do-it-yourself” style – only cares about the grammar
  - system then linearizes
  - built for English, ports to other languages available
- **RealPro** (Meaning-Text-Theory)
  - deep syntax/semantics → surface syntax → morphology
- **Treex** (Functional Generative Description)
  - deep syntax → surface syntax → morphology, linearization
  - Perl code operating over syntax trees

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

(Lavoie & Rambow, 1997)
http://dl.acm.org/citation.cfm?id=974596

(Popel & Žabokrtský 2010; Dušek et al., 2015)
https://ufal.mff.cuni.cz/~popel/papers/2010_icetal.pdf
https://www.aclweb.org/anthology/W15-3009



Trainable Realizers

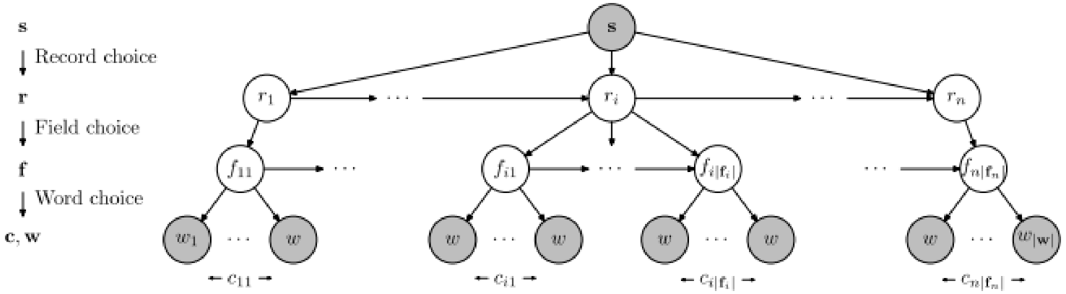
- **Overgenerate & Rerank**
  - same approach as for sentence planning
  - assuming a flexible handcrafted realizer (e.g., OpenCCG)
  - underspecified input → more outputs possible ← this means the grammar may be smaller
  - generate more & use statistical reranker, based on:
    - n-gram language models — NITROGEN (Langkilde & Knight, 1998) https://www.aclweb.org/anthology/P98-1116 HALOGEN (Langkilde-Geary, 2002) https://www.aclweb.org/anthology/W02-2103
    - Tree language models — FERGUS (Bangalore & Rambow, 2000) https://aclweb.org/anthology/C00-1007
    - expected text-to-speech output quality — (Nakatsu & White, 2006) https://www.aclweb.org/anthology/P06-1140
    - personality traits & alignment/entrainment — CRAG (Isard et al., 2006) https://www.aclweb.org/anthology/W06-1405
  - more variance, but at computational cost
- **Grammar/Procedural-based**
  - same as RealPro or TectoMT, but predict each step using a classifier

StuMaBa (Bohnet et al., 2010)
https://www.aclweb.org/anthology/C10-1012

Non-Neural End-to-End NLG

- NLG as language models
  - hierarchy of language models (HMM/MEMM/CRF style)
  - DA → slot → word level
- NLG using context-free grammars
  a) "language models" by probabilistic CFGs
     - approximate search for best CFG derivation

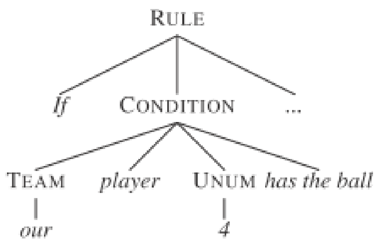
  b) synchronous PCFGs – MRs & text
     - "translation" with hierarchical phrase-based system
     - parsing MR & generating text



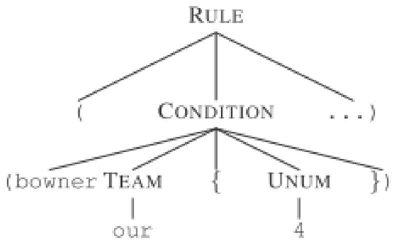
(Oh & Rudnicky, 2002) https://doi.org/10.1016/S0885-2308(02)00012-8
(Angeli et al., 2010) https://www.aclweb.org/anthology/D10-1049
(Liang et al., 2009) https://www.aclweb.org/anthology/P09-1011
(Mairesse et al., 2010) https://www.aclweb.org/anthology/P10-1157
(Mairesse & Young, 2014) https://www.aclweb.org/anthology/J14-4003

rule	prob./parameter
1. $S \rightarrow R(start)$	$[Pr = 1]$
2. $R(r_i.t) \rightarrow FS(r_j, start)\ R(r_j.t)$	$[P(r_j.t \mid r_i.t) \cdot \lambda]$
3. $R(r_i.t) \rightarrow FS(r_j, start)$	$[P(r_j.t \mid r_i.t) \cdot \lambda]$
4. $FS(r, r.f_i) \rightarrow F(r, r.f_j)\ FS(r, r.f_j)$	$[P(f_j \mid f_i)]$
5. $FS(r, r.f_i) \rightarrow F(r, r.f_j)$	$[P(f_j \mid f_i)]$
6. $F(r, r.f) \rightarrow W(r, r.f)\ F(r, r.f)$	$[P(w \mid w_{-1}, r, r.f)]$
7. $F(r, r.f) \rightarrow W(r, r.f)$	$[P(w \mid w_{-1}, r, r.f)]$
8. $W(r, r.f) \rightarrow \alpha$	$[P(\alpha \mid r, r.f, f.t, f.v)]$
9. $W(r, r.f) \rightarrow g(f.v)$	$[P(g(f.v).mode \mid r, r.f, f.t = int)]$

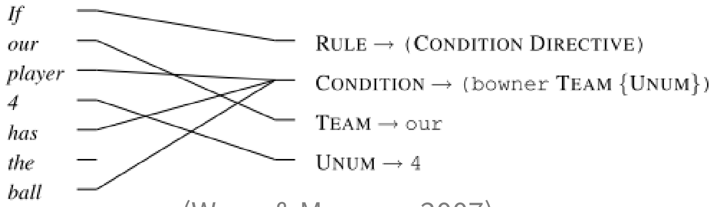
(Konstas & Lapata, 2012)
https://www.aclweb.org/anthology/P12-1039



(a) English



(b) CLANG



(Wong & Mooney, 2007)
https://www.aclweb.org/anthology/N07-1022

Neural Generation: Seq2seq RNNs (see NLU for RNN intro)

token representation: **embeddings**
= vectors of ~100-1000 numbers

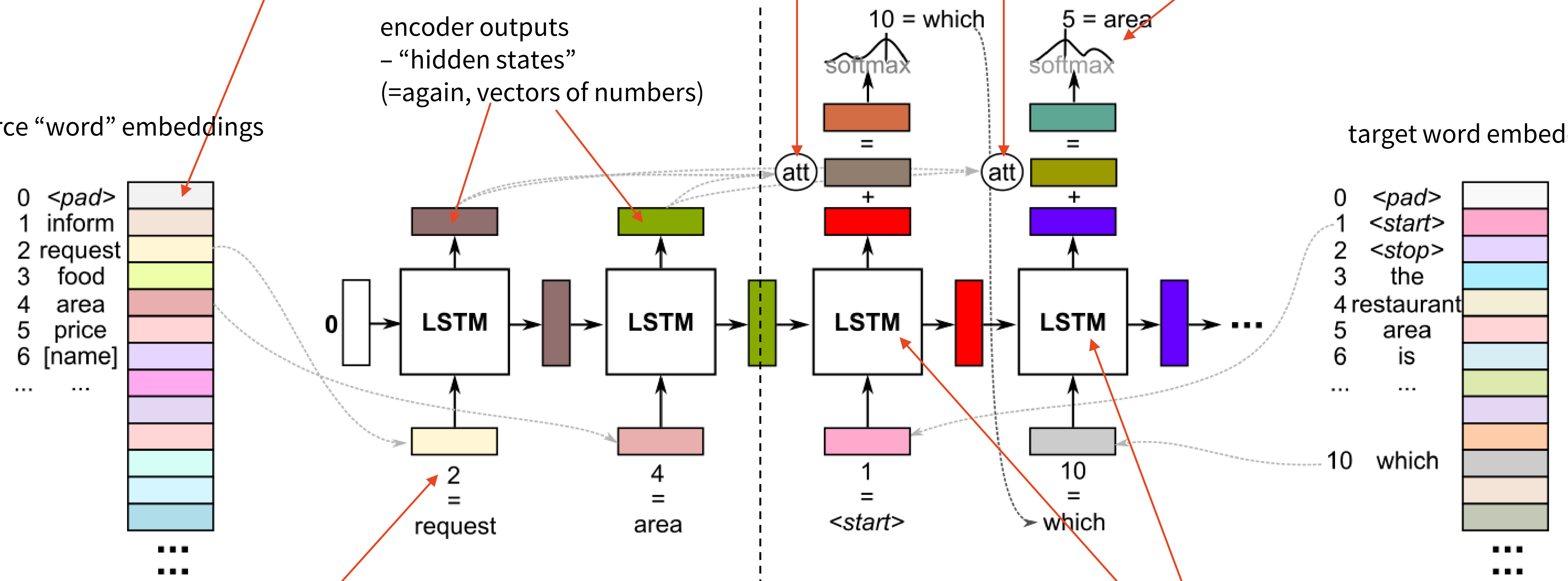
attention = weighted combination
(weights different for each step)

probability distribution
over the whole vocabulary

encoder outputs
– “hidden states”
(=again, vectors of numbers)

source “word” embeddings

target word embeddings



vocabulary is numbered

encoder

decoder

cells: identical (compound) neural layers
input: prev. output + token embedding

(Bahdanau et al., 2015) http://arxiv.org/abs/1409.0473

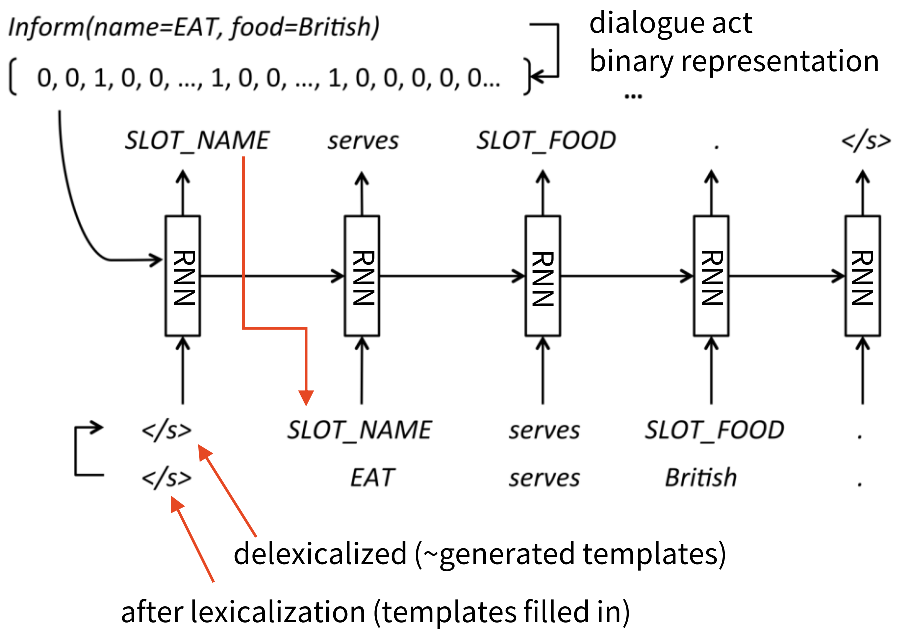
Neural End-to-End NLG: RNNs

- Unlike previous, doesn't need alignments
  - no need to know which word/phrase corresponds to which slot

`name [Loch Fyne], eatType[restaurant], food[Japanese], price[cheap], familyFriendly[yes]`

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

- 1st system: RNN language model conditioned on DA (~decoder only)
  - input: binary-encoded DA
    - 1 if intent/slot-value present, 0 if not
    - delexicalized: much fewer values, shorter vector
  - modified LSTM cells
    - input DA passed in every time step
  - generating delexicalized texts word-by-word
    - i.e. decoder only

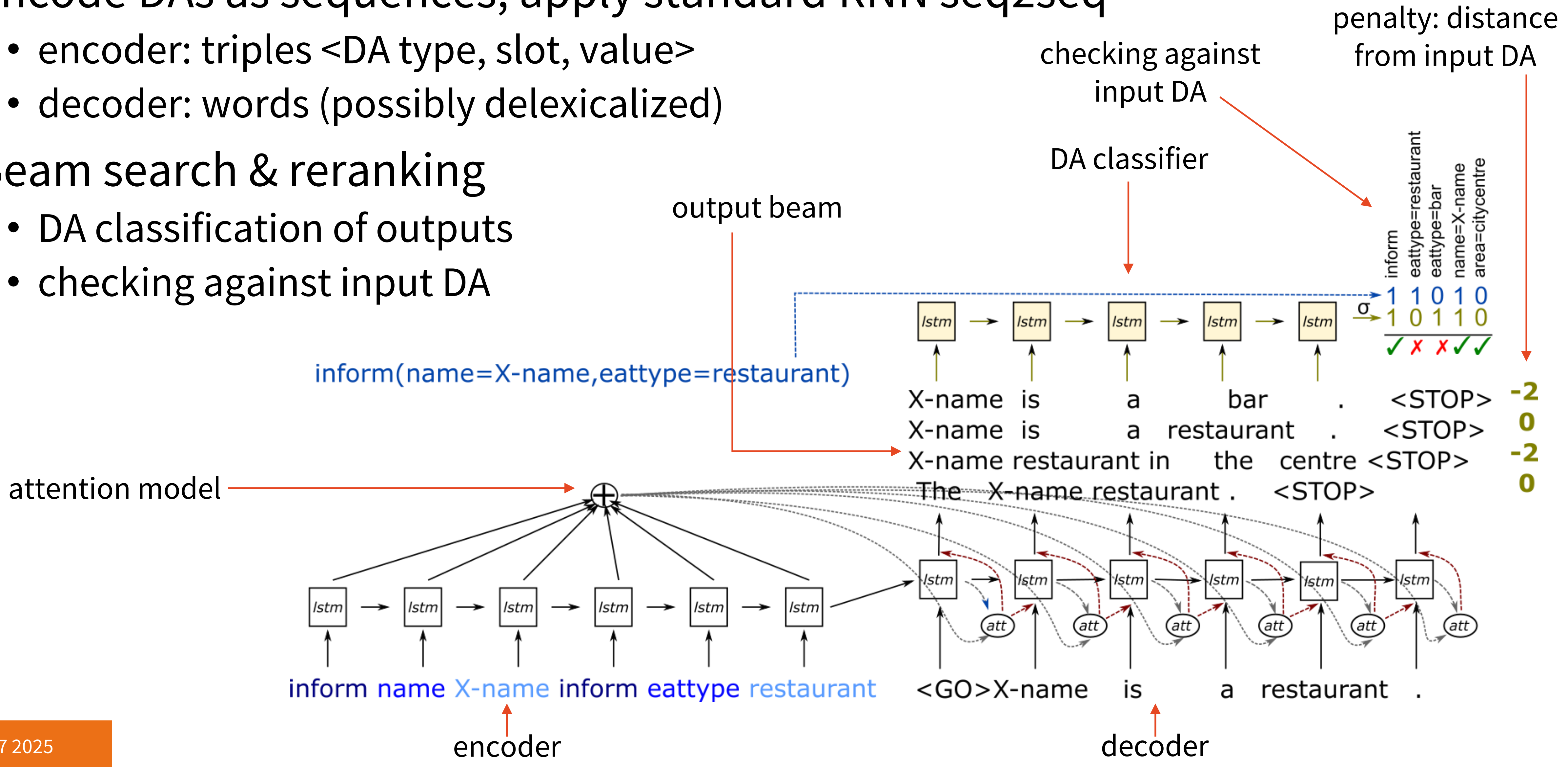


(Wen et al, 2015; 2016) http://aclweb.org/anthology/D15-1199 http://arxiv.org/abs/1603.01232

Seq2seq NLG with reranking (TGen)

(Dušek & Jurčíček, 2016)
https://aclweb.org/anthology/P16-2008

- Encode DAs as sequences, apply standard RNN seq2seq
  - encoder: triples <DA type, slot, value>
  - decoder: words (possibly delexicalized)
- Beam search & reranking
  - DA classification of outputs
  - checking against input DA

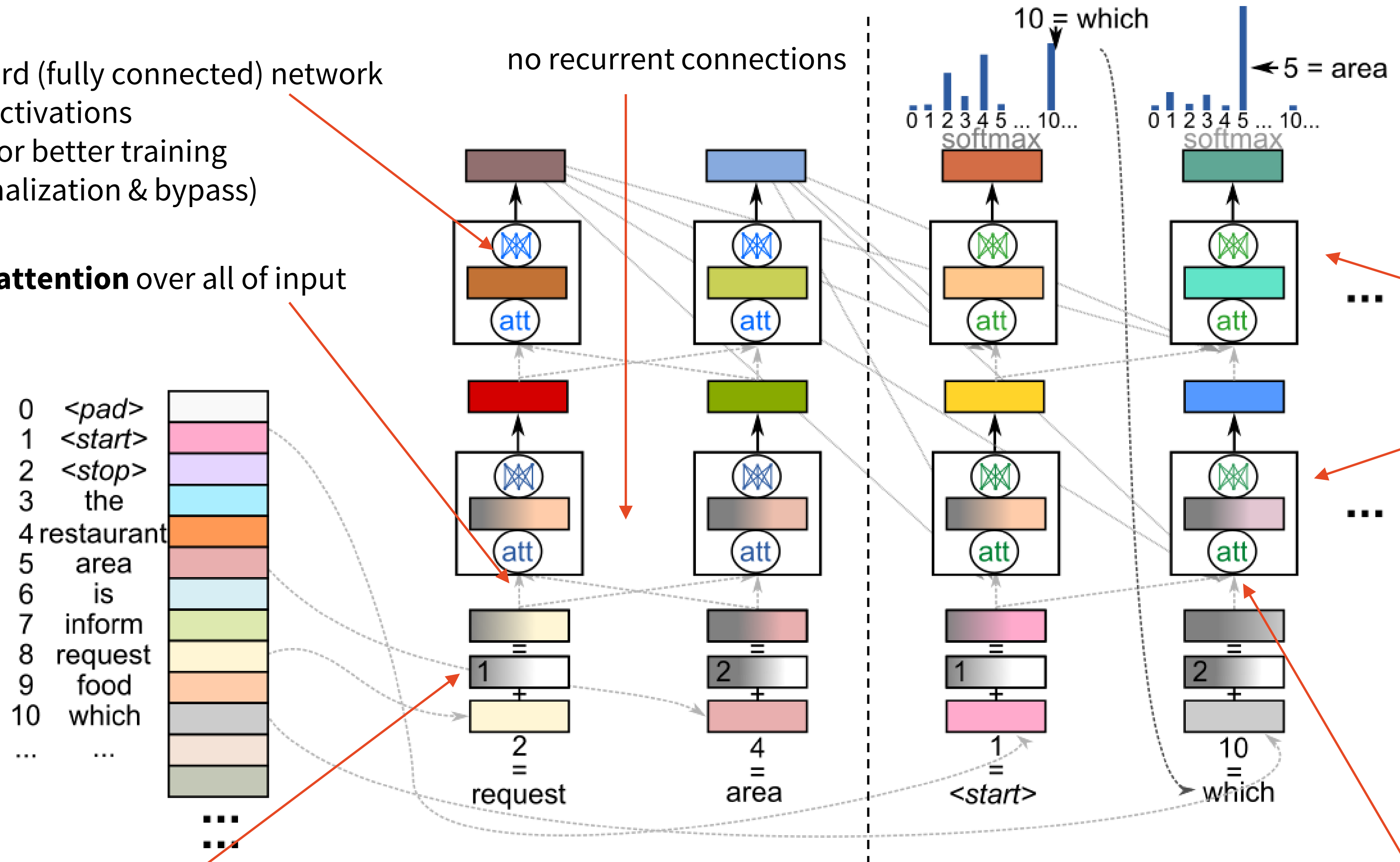


Transformer = seq2seq, with feed-forward & attention nets (instead of RNN)

- no RNN → parallel training → faster, allows larger models (more layers)

feed-forward (fully connected) network
- ReLU activations
- tricks for better training
- (+normalization & bypass)

attention over all of input



uses more layers ("blocks") (6-100, only 2 pictured)

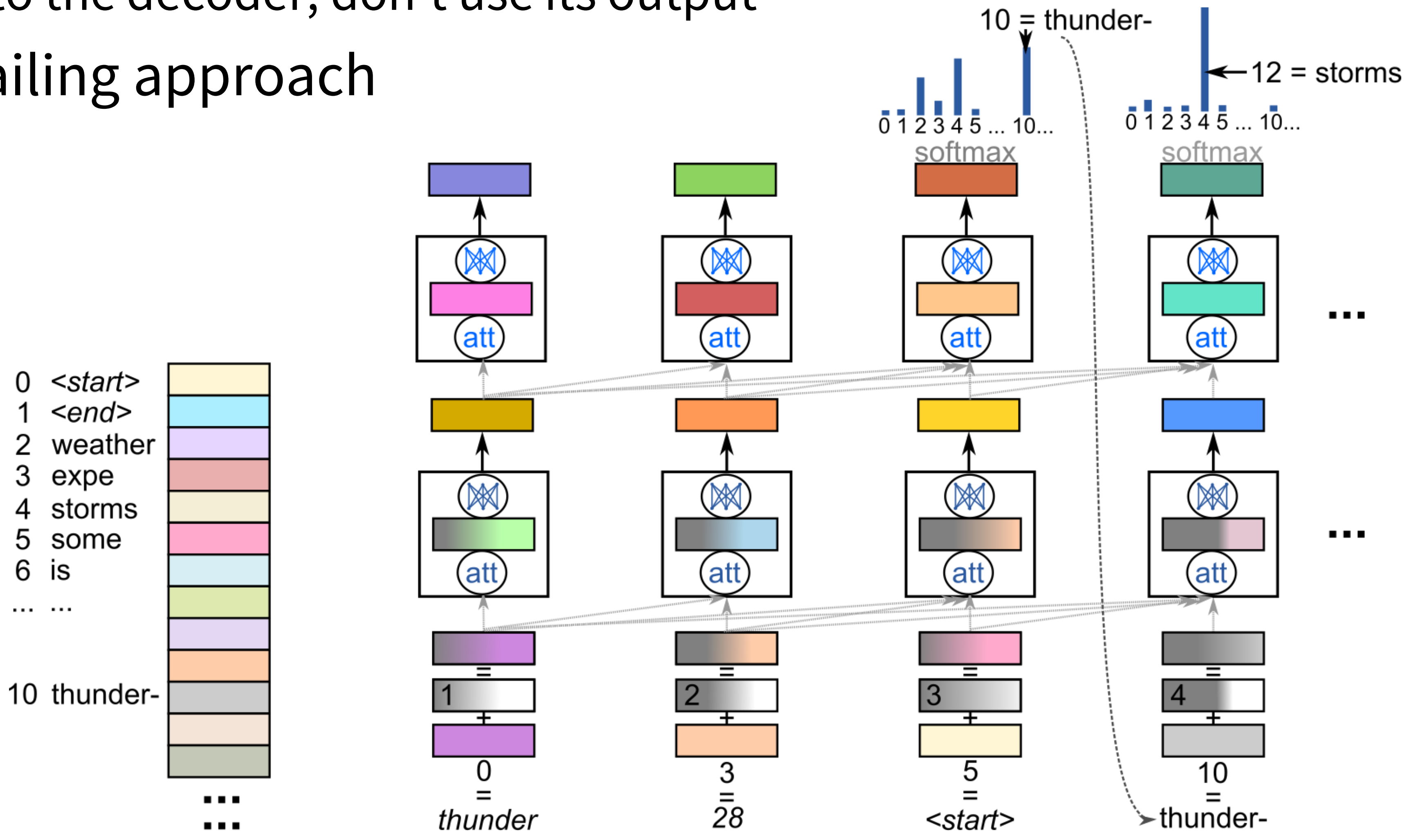
positional encoding (fixed / trained, indicate position in sentence)

attention over all of input & output generated so far (**self-attention**)

(Vaswani et al., 2017) http://arxiv.org/abs/1706.03762

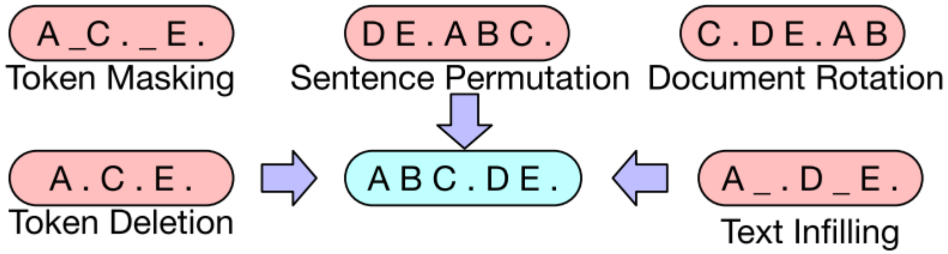
Transformer Decoders = seq2seq with a decoder model only

- Prompting = force-decoding
  - feed something into the decoder, don't use its output
- Currently the prevailing approach



Transformers & Pretrained Language Models

- **Pretrained language models** – on large data w/o annotation (**self-supervised**)
  - guess masked word (encoder only: **BERT**)
  - generate next word (decoder only: **GPT-2**)
  - fix distorted sentences (both: **BART**, **T5**)



(Lewis et al., 2020) https://aclanthology.org/2020.acl-main.703/

- Can be **finetuned** for your domain & task (just continue training)
  - less data than w/o pretraining, extremely fluent
  - i.e. finetune for MR → text, can learn implicit copying

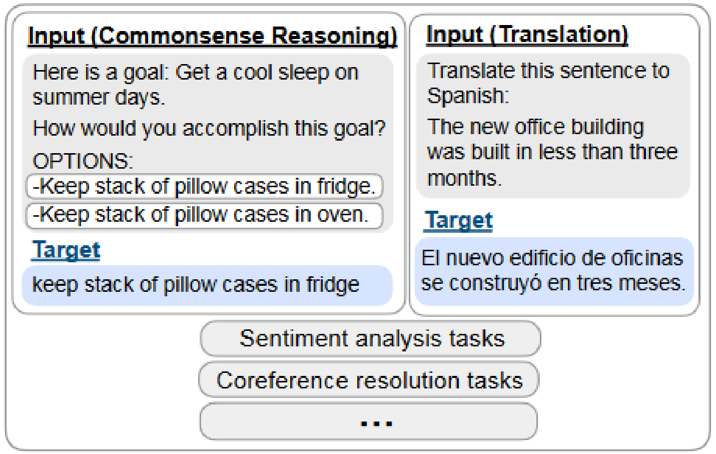
- Lot of them released online, plug-and-play
  - incl. multilingual versions (mBART, mT5)

https://github.com/huggingface/transformers

Large Language Models

(Zhao et al., 2023) http://arxiv.org/abs/2303.18223

(Wei et al., 2022) https://arxiv.org/abs/2109.01652



- Transformer decoder models (slightly updated)
- Large (10-100B params, pretrained on trillions of words)
- **Instruction tuning** – finetune on problems & solutions
- Reinforcement learning from human feedback (**RLHF**)
  1) generate lots of solutions for instructions
  2) pay humans to rate them
  3) learn a rating model (another LM: instruction + solution → score)
  4) use rating model score as reward in RL
  - main point: **reward is global** (not token-by-token) – RL-free alternatives exist
  - somewhat safer (low reward for bad behavior)
- Can just use **prompting**, no need for finetuning (though you can still can)
  - just feed in instructions/questions/example → LLM generates solution

(Axelsson and Skantze, 2023) http://arxiv.org/abs/2307.07312

https://nlpnewsletter.substack.com/p/instruction-tuning-vol-1
https://huggingface.co/blog/rlhf

Problems with neural NLG

(Dušek et al., 2020)
http://arxiv.org/abs/1901.07931

- Checking the **semantics**
  - neural models tend to forget input / make up irrelevant stuff
  - reranking / decoding changes work, but not perfectly
- Generally **hard to control** (especially LLMs)
  - sensitive to prompts – prompt engineering may be required
  - parsing replies "Sure, here's the sentence you wanted…"
- Need quite a lot of data (except for LLMs with prompting)
- Diversity & complexity of outputs
  - still can't match humans
  - needs specific tricks to improve this
- Still might be more hassle than writing up templates 😏

(Kasner & Dušek, 2024)
https://aclanthology.org/2024.acl-long.651

Summary

Deep Reinforcement Learning

- same as plain RL – agent + states, actions, rewards – just Q or π 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 π 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
- neural: **RNN** / **Transformer**, **pretrained models**, **LLMs**

Thanks

Contact us:

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

Labs at 3:40pm in S1

Get these slides here:

http://ufal.cz/npfl123

References/Inspiration/Further:

- Matiisen (2015): Demystifying Deep Reinforcement Learning: https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/
- Karpathy (2016): Deep Reinforcement Learning – Pong From Pixels: http://karpathy.github.io/2016/05/31/rl/
- David Silver's course on RL (UCL): http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- Sutton & Barto (2018): Reinforcement Learning: An Introduction (2nd ed.): http://incompleteideas.net/book/the-book.html
- Milan Straka's course on RL (Charles University): http://ufal.mff.cuni.cz/courses/npfl122/
- Deep RL for NLP tutorial: https://sites.cs.ucsb.edu/~william/papers/ACL2018DRL4NLP.pdf
- Mnih et al. (2013): Playing Atari with Deep Reinforcement Learning: https://arxiv.org/abs/1312.5602
- Mnih et al. (2015): Human-level control through deep reinforcement learning: https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
- Gatt & Krahmer (2017): Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation http://arxiv.org/abs/1703.09902
- My PhD thesis (2017), especially Chapter 2: http://ufal.mff.cuni.cz/~odusek/2017/docs/thesis.print.pdf