Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto

isep
Instituto Superior de Engenharia do Porto

Conversational Systems

Língua Natural e Sistemas Conversacionais

Luiz Faria

Adapted from Natural Language Processing course from HSE University

Task-oriented Dialog Systems

Task-oriented Dialog System

We can talk to a personal assistant:

- Apple Siri
- Google Assistant
- Microsoft Cortana
- Amazon Alexa
- . . .

We can solve these tasks:

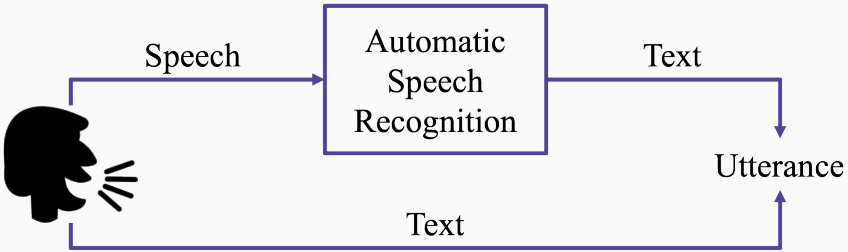
- Set up a reminder
- Find photos of your pet
- Find a good restaurant
- . . .

Task-oriented Dialog System

We can write to a chat bot:

- To book tickets
- To order food
- To contest parking tickets
- To track expenses
- ...

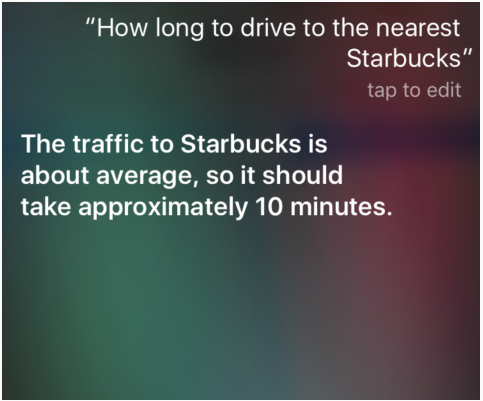
Utterance



- In this course we don't deal with speech recognition

Intent Classification

- What does the user want?
- Which predefined scenario is the user trying to execute?

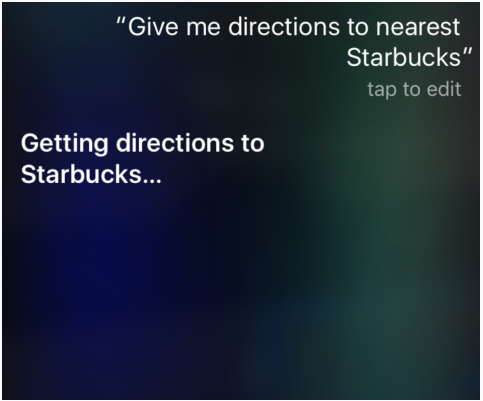


©Apple Siri

Intent: **nav.time.closest**

Different Intents

- And we need to classify them to give correct answers
- This is a classification task and you can measure **accuracy**

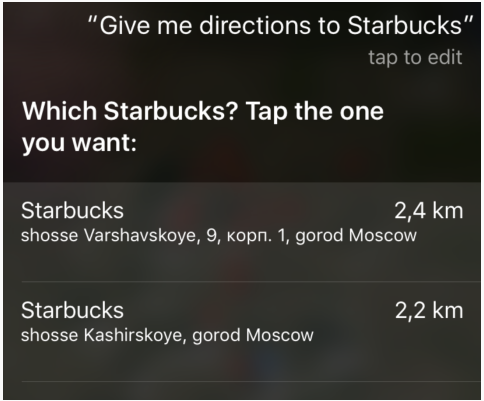


©Apple Siri

Intent: **nav.directions.closest**

Different Intents

- This time assistant needs additional information and initiates dialog



Intent: **nav.directions**

©Apple Siri

Form Filling Approach to Dialog Management

- Think of an intent as a **form** that a user needs to fill in
- Each intent has a set of fields (**slots**) that must be filled in to execute the request

- Example: **nav.directions** intent
  - **@FROM** slot: defaults to current geolocation
  - **@TO** slot: required

- We need a **slot tagger** to extract slots from utterance

Slot Filling/Tagging

- Example:
  - User: Show me the way to History Museum
  - Slot tagger: Show me the way to @TO{History Museum}
- This corresponds to sequence tagging task
- This is token classification task in **BIO** scheme:
  - **B** corresponds to the word at the **beginning** of the slot
  - **I** corresponds to the word **inside** the slot (excluding beginning)
  - **O** corresponds to words **outside** of slots
- Example of **BIO** scheme tagging:

Tokens	Show	me	the	way	to	History	Museum
Tags	O	O	O	O	O	B-TO	I-TO

Slot Filling/Tagging

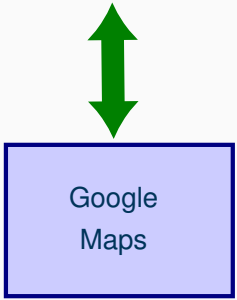
- We **train** it as a sequence tagging task in BIO scheme
- A slot is considered to be correct if its range and type are correct
- **Recall** = $\frac{\textbf{\# correct slots found}}{\textbf{\# true slots}}$
- **Precision** = $\frac{\textbf{\# correct slots found}}{\textbf{\# found slots}}$
- We can **evaluate** slot tagger with $F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

Recall: from all the true slots, we find out which of them are correctly found by our system.
Precision: from all of found slots by our system, we find out which of them are correctly classified slots.
Evaluation of the slot tagger with F1 measure: harmonic mean of precision and recall.

Form Filling Dialog Manager (single turn)

- User: **Give me directions to San Francisco**
  - Intent classifier: nav.directions
  - Slot tagger: @TO{San Francisco}
  - Dialog manager: *all slots are filled, here's the route*
- Agent (assistant): **Here's the route**

Single turn scenario: we give a single utterance to the system and get the result right away



Form Filling Dialog Manager (multi turn)

- User: **Give me directions from Los Angeles**
  - Intent classifier: nav.directions
  - Slot tagger: @FROM{Los Angeles}
  - Dialog manager: *required slot is missing, where to?*
- Agent (assistant): **Where do you want to go?**
- User: **San Francisco**
  - Intent classifier: nav.directions
  - Slot tagger: @TO{San Francisco}
  - Dialog manager: *okay, here's the route*
- Agent (assistant): **Here's the route**

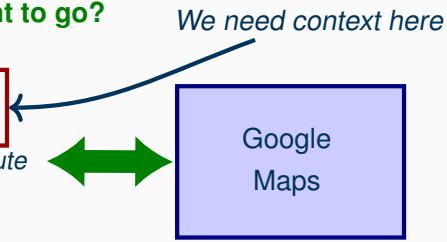
Form Filling Dialog Manager (multi turn)

- User: **Give me directions from Los Angeles**
  - Intent classifier: nav.directions
  - Slot tagger: @FROM{Los Angeles}
  - Dialog manager: *required slot is missing, where to?*
- Agent (assistant): **Where do you want to go?**
- User: **San Francisco**
  - Intent classifier: nav.directions
  - Slot tagger: @TO{San Francisco}
  - Dialog manager: *okay, here's the route*
- Agent (assistant): **Here's the route**

The problem here is that during the second turn, if we don't know the history of the conversation and just see the utterance San Francisco, it's really hard to guess that this is in navigation.directions intent and that San Francisco actually fills "TO" slot. So, we need to add context to our intent classifier and slot tagger and that context is actually some information about what happened previously.

Form Filling Dialog Manager (multi turn)

- User: **Give me directions from Los Angeles**
  - Intent classifier: nav.directions
  - Slot tagger: @FROM{Los Angeles}
  - Dialog manager: *required slot is missing, where to?*
- Agent (assistant): **Where do you want to go?**
- User: **San Francisco**
  - Intent classifier: nav.directions
  - Slot tagger: @TO{San Francisco}
  - Dialog manager: *okay, here's the route*
- Agent (assistant): **Here's the route**



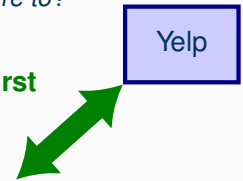
How to Track Context (an easy way)

- Both intent classifier and slot tagger need context (what happened before)
- Let's add simple **features** to both of them:
  - Previous utterance intent as a categorical feature
  - Slots filled in so far with binary feature for each possible slot – so that the system during slot tagging already knows which slots are filled by the user previously and which are not, and that will help it to decide which slot is correct in the utterance it sees
- Improves slot tagger F1 by 0.5%
- Reduces intent classifier error by 6.7%
- However, later we will review a better way: memory networks

http://www.cs.toronto.edu/~aditya/publications/contextual.pdf

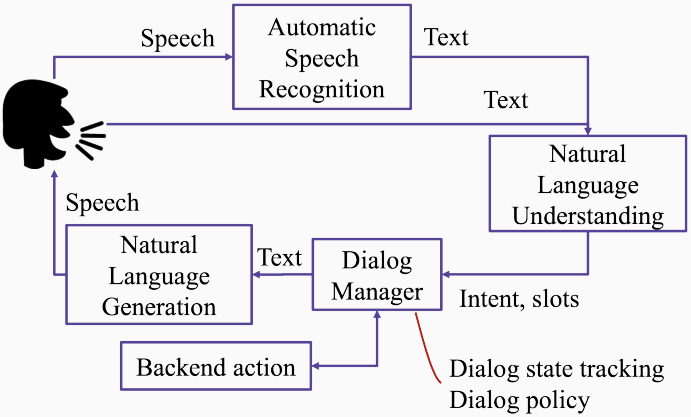
How to Track a Form Switch

- User: **Give me directions from Los Angeles**
  - Intent classifier: nav.directions
  - Slot tagger: @FROM{Los Angeles}
  - Dialog manager: *required slot is missing, where to?*
- Agent (assistant): **Where do you want to go?**
- User: **Forget about it, let's eat some sushi first**
  - Intent classifier: nav.find
  - Slot tagger: @CATEGORY{sushi}
  - Dialog manager: *okay, let's start a new form and find some sushi*
- Agent (assistant): **Okay, here are nearby sushi places**



"Forget about it . . . " – The system needs to understand that the intent has changed and it should forget about all the previous slots and information that it had because that information is no longer needed. The intent classifier gives us "navigation.find" and the "category", which is a slot and it has the value of "sushi".

Task-oriented Dialog System Overview



The speech of the user is the input of the NLU module. This module outputs intents and slots for the utterance. Then the dialog manager is responsible for two tasks. The first one is dialog state tracking – to understand what the user wanted throughout the conversation and track that state. The second one is dialog policy managing. The policy maps the state to a query in order to get some information from the user or request some information from a backend service (like Google Maps or Yelp). After that, the system is ready to give users some information, using a natural language generator.

Young, S.J., Probabilistic methods in spoken–dialogue systems, 2000.

Summary

- Overview of a task-oriented dialog system with form filling
- We evaluate **accuracy** for intent classifier and **F1-measure** for slot tagger
- Next, a closer look at intent classifier and slot tagger

Intent Classifier and Slot Tagger

Intent Classifier

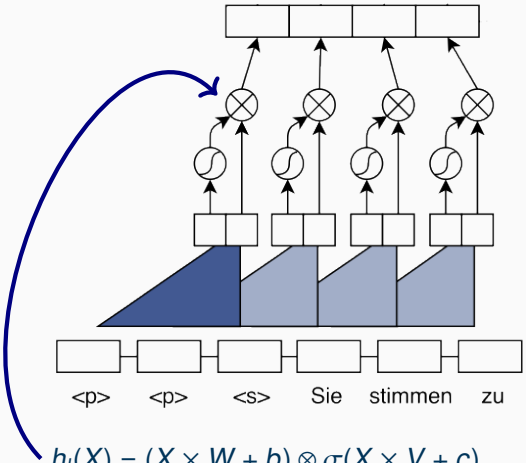
- What we can do:
  - Any model on BOW with n-grams and TF-IDF (classical approaches of text mining)
  - RNN (LSTM, GRU, ...)
  - CNN (1D convolutions)
- CNN can perform better on datasets where the task is essentially a key phrase recognition task as in some sentiment detection datasets

https://arxiv.org/pdf/1702.01923.pdf

Slot Tagger

- A more difficult task
- What we can do:
  - Handcrafted rules like regular expressions
    - Ex: *Take me to . . .* – probably this is a **TO** slot – but it does not scale well
  - Conditional Random Fields (CRF)
  - RNN seq2seq
  - **CNN seq2seq** (it works faster and sometimes it even beats RNN in some tasks – in the next slide)
  - Any seq2seq with attention

CNN for Sequences: Gated Linear Unit



$$h_l(X) = (X \times W + b) \otimes \sigma(X \times V + c)$$

https://arxiv.org/pdf/1612.08083.pdf
https://arxiv.org/pdf/1705.03122.pdf

Using CNN to predict the next token of a sequence (language modelling task)
Let's say we have an input sequence which is pad, pad, then start of sequence and three German words. Usually, we use RNN to predict the next token. Here we will use CNN. Let's say that when we generate the next token, we actually care only about the last three tokens in the sequence that we have seen. Assuming that, we can use convolution to aggregate the information about the last three tokens (the blue triangle), and we actually get some filters in the output. Let's take half of those filters and add them, and the second half, we will pass through sigmoid activation function, and then take an element-wise multiplication of these two halves. What we get is a Gated Linear Unit that allow us to predict the next token based on the context that we had before. But instead of look at only the three last tokens we can look at a large set of tokens like as in a LSTM cell, that has a very long short-term memory.

CNN for Sequences: Results

- They can sometimes beat LSTM in **language modeling:** (lower perplexity)

Model	Test PPL	Hardware
LSTM-1024 (Grave et al., 2016b)	48.7	1 GPU
GCNN-8	44.9	1 GPU
GCNN-14	37.2	4 GPUs

Table 3. Results for single models on the WikiText-103 dataset.

- . . . and **machine translation:**

WMT'14 English-French	**BLEU**
Wu et al. (2016) GNMT (Word 80K)	37.90
Wu et al. (2016) GNMT (Word pieces)	38.95
Wu et al. (2016) GNMT (Word pieces) + RL	39.92
ConvS2S (BPE 40K)	40.51

https://arxiv.org/pdf/1612.08083.pdf
https://arxiv.org/pdf/1705.03122.pdf

CNN for Sequences: Speed Benefit

- They work faster than RNN:
  - During **training** we can process all time steps in parallel
  - During **testing** encoder can do the same
  - During **testing** we get higher throughput thanks to convolution optimizations in GPUs

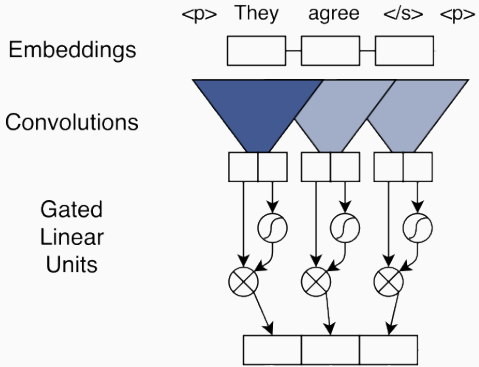
	BLEU	Time (s)
GNMT GPU (K80)	31.20	3,028
GNMT CPU 88 cores	31.20	1,322
GNMT TPU	31.21	384
ConvS2S GPU (K40) $b = 1$	33.45	327
ConvS2S GPU (M40) $b = 1$	33.45	221
ConvS2S GPU (GTX-1080ti) $b = 1$	33.45	142
ConvS2S CPU 48 cores $b = 1$	33.45	142

Translation generation speed during testing

https://arxiv.org/pdf/1705.03122.pdf

The problem with RNN is that it has a hidden state and that state changes through iterations – this do not allow perform calculations in parallel, because every step depends on the previous. We can overcome this with CNN because during training, we can process all time steps in parallel. So, we apply the same convolutional filters but we do that at each time step, and they are independent and we can do that in parallel. During testing, the encoder can actually do the same because there is no that dependence on the previous outputs and we use only our input tokens, and we can apply that convolutions and get our hidden states in parallel.

CNN for Sequences: Encoder



In a sequence-to-sequence task, it's usual to use a bi-directional encoder, so that the sequence is processed in both directions. The advantage of convolutions is that we can make that convolutional filters symmetric, so that we can look at the context at the left and at the right to the same time. So, it is easy to make a bi-directional encoder with CNNs. And it still works in parallel since there is no dependence on hidden states, it just applies all of that multiplications in parallel.

- Bi-directional encoder is easy
- Works in parallel for all time steps

https://arxiv.org/pdf/1705.03122.pdf

ATIS Dataset

- Airline Travel Information System
- Collected in 90s
- 5,000 context independent utterances (one turn dialogs) – a simple dialog manager can be used
- 17 intents, 127 slot labels (like *“from location”, “to location”, “departure time”, . . .*)
- State-of-the-art: 1.79% intent error, 95.9 slots F1

Returning to the intent classifier and slot tagger, we will consider the ATIS dataset to overview these tasks.

Utterance	show	flights	from	Seattle	to	San	Diego	tomorrow
Slots	O	O	O	B-fromloc	O	B-toloc	I-toloc	B-depart_date
Intent	Flight							

http://www.isca-speech.org/archive_v0/Interspeech_2016/pdfs/1352.PDF
https://www.microsoft.com/en-us/research/wp-content/uploads/2010/12/SLT10.pdf

Joint Training of Intent Classifier and Slot Tagger

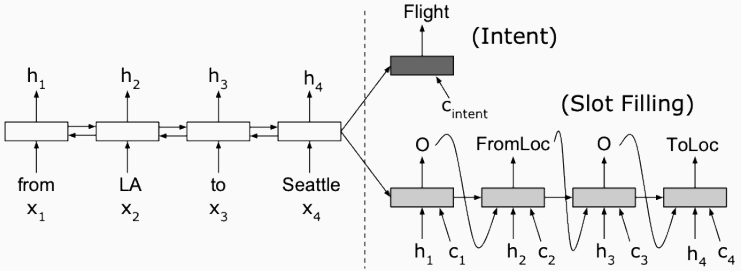
- They both analyze the same sequence
- What if we learn representations suitable for both tasks?
- That results in more supervision and higher quality of both

Another thing is that we can learn the intent classifier and slot tagger jointly. We don't need to train the two tasks separately. We can train this supertask, because it can learn representations that are suitable for both tasks, and this time, we provide more supervision for the training and we get a higher quality as a result.

http://www.isca-speech.org/archive_v0/Interspeech_2016/pdfs/1352.PDF

Joint Training of Intent Classifier and Slot Tagger

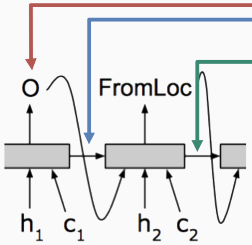
- Encoder-decoder architecture for joint intent detection and slot filling
- Encoder is a bi-directional LSTM
- With aligned inputs (h_i on the right) and attention (c_i on the right)



http://www.isca-speech.org/archive/Interspeech_2016/pdfs/1352.PDF

Sequence-to-sequence model, with a bi-directional encoder. The last hidden state of the encoder is used for decoding the slot tags, and at the same time we can use it to decode the intent. Both tasks are trained end-to-end, ensuring a higher quality model. In the decoder, we will use the hidden states from the encoder (called aligned inputs h_i), and attention vectors c_i.

Attention in Decoder



Decoder state s_i: $s_i = f(s_{i-1}, y_{i-1}, h_i, c_i)$

Attention c_i (g is a feed forward network):

$$c_i = \sum_{j=1}^{T} \alpha_{i,j} h_j \qquad \alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{k=1}^{T} \exp(e_{i,k})}$$

$$e_{i,k} = g(s_{i-1}, h_k)$$

Attention weights (the darker the higher) when predicting the slot label for the last word "noon":

	O	O	B-fromloc. city_name	O	B-toloc. city_name	O	O	B-depart_time. time_relative	B-depart_time. period_of_day
$\alpha_{T,j}$	flight	from	cleveland	to	dallas	that	leves	before	noon

http://www.isca-speech.org/archive_v0/Interspeech_2016/pdfs/1352.PDF

Joint Training Results

- Better performance on ATIS dataset:

Training	Slots F1	Intent % error
Independent training for slot filling	95.78	-
Independent training for intent detection	-	2.02
Joint training for slot filling and intent detection	**95.87**	**1.57**

- Works faster than two separate models because we have only one encoder and we reuse that information for the two tasks

http://www.isca-speech.org/archive_v0/Interspeech_2016/pdfs/1352.PDF

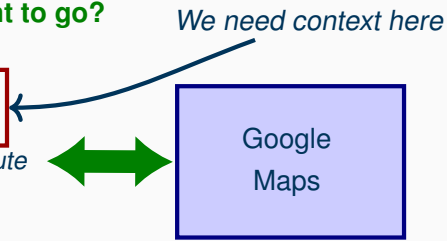
Summary

- We've overviewed different options for intent classifier and slot tagger training
- People start to use CNN for sequence modeling and sometimes get better results than with RNN
- Joint training can be beneficial in terms of speed and performance
- Next, we'll take a look at context utilization in NLU (intent classifier and slot tagger)

Utilizing Context in NLU

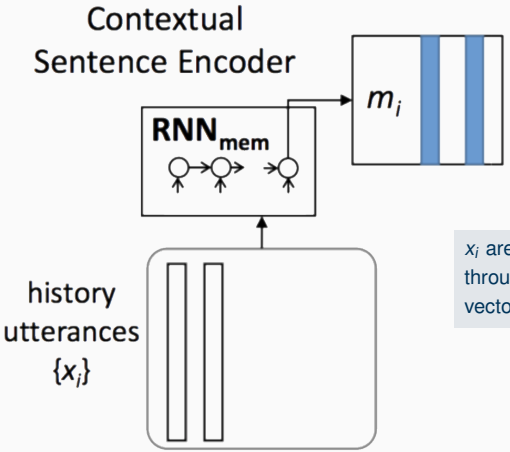
We Need Context to Handle Multi-turn Dialogs

- User: **Give me directions from Los Angeles**
  - Intent classifier: nav.directions
  - Slot tagger: @FROM{Los Angeles}
  - Dialog manager: *required slot is missing, where to?*
- Agent (assistant): **Where do you want to go?**
- User: **San Francisco**
  - Intent classifier: nav.directions
  - Slot tagger: @TO{San Francisco}
  - Dialog manager: *okay, here's the route*
- Agent (assistant): **Here's the route**



After the first user's sentence, the intent classifier got the user's intent, the slot tagger filled the @FROM slot, and the DM detected a missing slot (@TO) and ask the user about the destination. After receiving the next user's sentence (San Francisco), it is desirable that the system understands that San Francisco is the @TO slot that we are waiting, and the intent didn't change. For that, the system needs to keep a context. A proper way to do this is called memory networks.

Storing All Previous Utterances in "Memory"

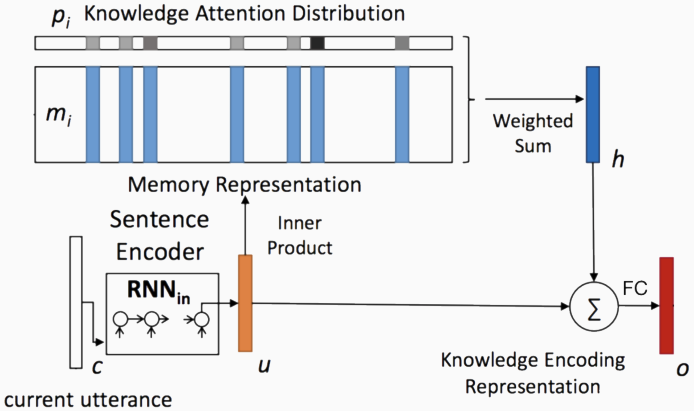


$$m_i = \text{RNN}_{\text{mem}}(x_i)$$

x_i are the previous utterances. Each utterance x_i is passed through a special RNN, that will encode them into memory dense vectors. The memory m contains the encoded utterances.

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/06/IS16_

What Knowledge is Relevant to New Utterance?



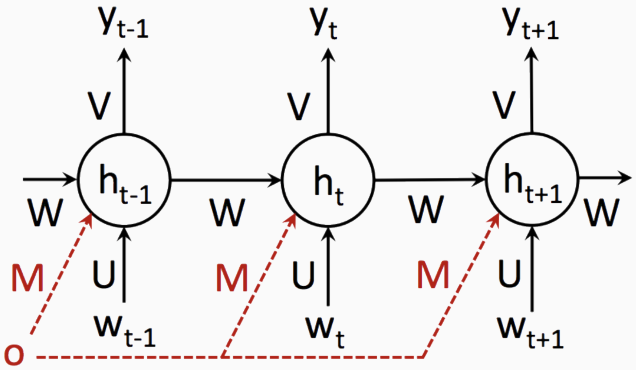
$$u = \mathrm{RNN}_{\mathrm{in}}(c)$$

The current utterance *c* is encoded in a *u* vector with the same size of memory vectors, using the RNN_{in}. The representation of the current utterance (*u*) is matched with each of the memory vectors, using the dot product and then applying softmax. This way, we get a knowledge attention distribution, representing what previous knowledge is relevant to the current utterance. The attention weights are used to get a weighted sum of all the memory vectors (vector *h*). Next, we add *h* vector to the utterance representation *u* through a fully connected layer and get the final vector *o* which is the knowledge encoding of the current utterance and the knowledge that we had before.

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/06/IS16_

Tagging Current Utterance with Knowledge

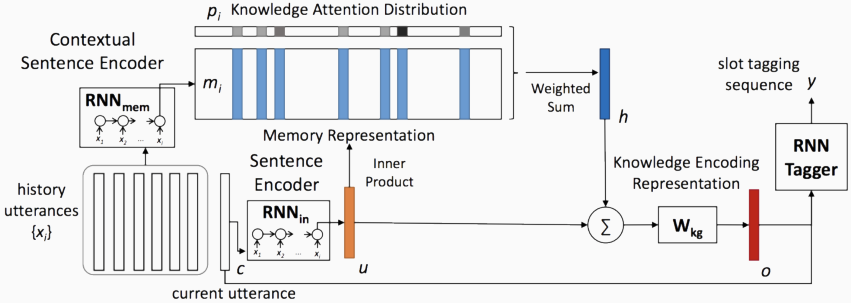
- We add knowledge representation in final RNN tagger:



The *o* vector accumulates all the context of the dialog that we had before. So, we can use it in a RNN for tagging. This knowledge vector is added to the input on every step of the RNN tagger. This *o* vector doesn't change over the RNN steps. The use of the context allow us to get a better quality in the tagging process.

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/06/IS16_

How to Track Context (with memory networks)



Chen et al, 2016

- We encode previous utterances to store them in **"memory"** as dense vectors
- We use **attention** mechanism to retrieve relevant prior knowledge about the conversation

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/06/IS16_

How to Track Context (with memory networks)

- Evaluation results for slot tagger:
  - Multi-turn dataset
  - F1-measure

Model	First turn	Other turns	Overall
RNN tagger wo context	55.8	45.7	47.4
Memory Network	73.2	65.7	67.1

Summary

- We can make the NLU context-aware with memory networks
- Next, we'll take a look at lexicon utilization in our NLU (e.g. a list of cities or a list of music artists, used to fill the slots)

Utilizing Lexicon in the NLU

Why do We Want to Use Lexicon?

- Let's take ATIS dataset
- It has finite set of cities in training
- Will the model work for a new city?
- We have **a list of all cities**, why not use it?

- Another example
- Imagine we need to fill a slot "music artist"
- We have **all music artists** in the database like musicbrainz.org
- How can we use it?

Adding Lexicon Features to Input Words

- Let's **match every n-gram** of input text against entries in our lexicon

Take me to San Francisco

- A match is successful when **the n-gram matches the prefix or postfix** of an entry and is at least half the length of the entry

Matches:

"San" → "San Antonio"
"San" → "San Francisco"
"San Francisco" → "San Francisco"

- When there are multiple **overlapping matches**:
  - Prefer exact matches over partial
  - Prefer longer matches over shorter
  - Prefer earlier matches in the sentence over later

https://arxiv.org/pdf/1511.08308v4.pdf

Matches Encoding

We will use **BIOES** coding (Begin, Inside, Outside, End, Single)

- B – if token matches the beginning of some entity
- B, I – if two tokens match as prefix
- I, E – if two tokens match as postfix
- S – if matched single token entity
- ...

Example for 4 lexicon dictionaries (LOCation, MISCellaneous, ORGanization, and PERSon):

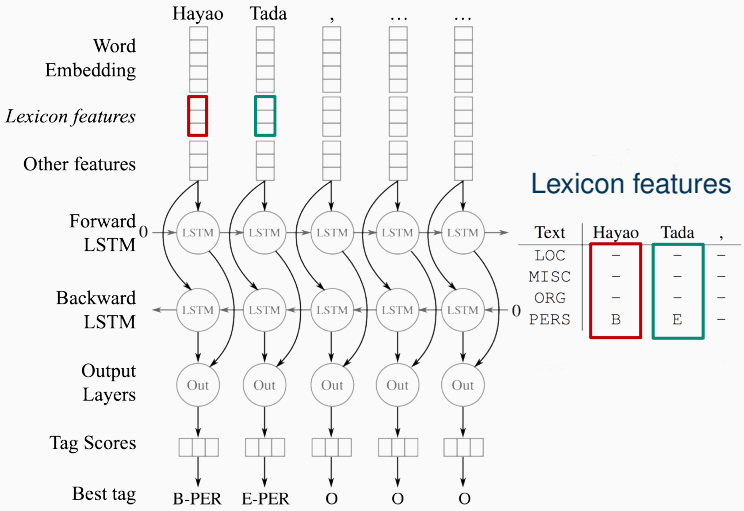
Text	Hayao	Tada	,	commander	of	the	Japanese	North	China	Area	Army
LOC	–	–	–	–	–	B	I	–	S	–	–
MISC	–	–	–	S	B	B	I	S	S	S	S
ORG	–	–	–	–	–	B	I	B	I	I	E
PERS	B	E	–	–	–	–	–	–	S	–	–

B, I, O, E, S letters are later encoded as one-hot vectors

https://arxiv.org/pdf/1511.08308v4.pdf

How can we use that matching in the model? We will use a coding named BIOES.
A particular aspect of this encoding is that we can have a full match even if we don't have an entry in the lexicon. Let's consider that the lexicon has the entry "North China History Museum" but don't have any country area army entities. Considering those two entities, we can combine the postfix from the second one (Area Army) with the prefix from the first match (North China) and it will still give us the same BIOES encoding. So, we can make new entities that we haven't seen before.

Adding These Features to the Model



Lexicon features

Text	Hayao	Tada	,
LOC	–	–	–
MISC	–	–	–
ORG	–	–	–
PERS	B	E	–

How can we use the lexicon information in the model?
We have a word embedding for every token, and to each word embedding, we can add lexicon information. Considering the table from the previous slide, let's take the two first words. For each word we will take the corresponding column and convert it in a one hot encoding, to decode that BIOES letters into numbers. this vector will be concatenated with the embedding vector for the word. The resulting vector will be used as input of a bi-directional LSTM. This LSTM will predict tags for the slot tagger. This approach allow to embed lexicon information into the model.

https://arxiv.org/pdf/1511.08308v4.pdf

Lexicom Helps

CoNLL-2003 Named Entity Recognition task:

Model	CoNLL-2003		
	Prec.	Recall	F1
FFNN + emb + caps + lex	89.54	89.80	89.67 (± 0.24)
BLSTM	80.14	72.81	76.29 (± 0.29)
BLSTM-CNN	83.48	83.28	83.38 (± 0.20)
BLSTM-CNN + emb	90.75	91.08	90.91 (± 0.20)
BLSTM-CNN + emb + lex	91.39	**91.85**	**91.62** (± 0.33)

The use of lexicon information in a Named Entity Recognition task allow to improve Precision, Recall and F1 measure a little bit, about 1%. So, it will be helpful to implement these lexicon features for a real world dialogue system.

Training Details

- We can **sample** our **lexicon** dictionaries so that our model learns the context of entities as well as lexicon features
- This procedure helps **to detect unknown entities at testing**
- We can augment the dataset replacing slot values with values from the same lexicon:

Take me to **San Francisco**

Take me to **Washington**

We can sample the lexicon dictionaries so that the model learns not only the lexicon features but also the context of the words. In the sentence “Take me to San Francisco”, the word that comes after the phrase “take me to” is most likely a @TO slot. We want the model to learn those features as well because in real world, we can see entities that were not in the vocabulary before, and the lexicon features will not work. So, this sampling procedure gives us an ability to detect unknown entities during testing. With lexicon dictionaries, we can also augment the data set because we can replace the slot values by some other values from the same lexicon. For instance, “Take me to San Francisco” becomes “Take me to Washington” because we can replace San Francisco’s slot value with Washington.

Summary

- Lexicon features can be used to further improve the NLU, helping to detect entities and unknown entities like “South China Area Army”
- Next, we’ll take a look at Dialog Manager (DM)

Dialog Manager (state tracking)

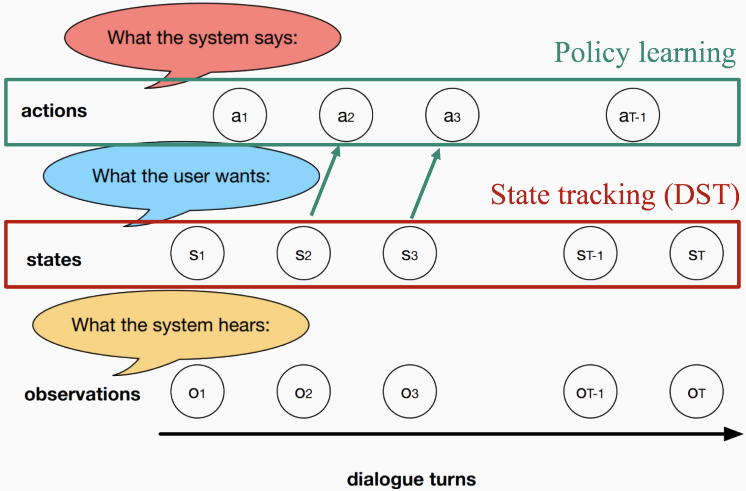
Dialog Manager

DM are responsible for two tasks:

- **State tracker** (requires hand-crafted states)
  - Queries the external database or knowledge base
  - Tracks the evolving state of the dialog
  - Constructs the state estimation after every utterance from the user
- **Policy learner**
  - Takes the state estimation as input and chooses the next best dialog action

https://arxiv.org/pdf/1703.07055.pdf

State Tracking and Policy Learning



According to the state, the system must react based on a policy (rule)

↑

From the observations, the Dialog Manager updates the state of the dialog (tracks the state of the dialogue); with every new utterance, the user can specify more details or change its intent, affecting the state - the state describes what the user wants

↑

From dialog turns, the system get information from user (observations)

DSTC 2 Dataset

- Dialog State Tracking Challenge, collected in 2013
- **Human-computer** dialogs about finding a restaurant in Cambridge
  - 3324 telephone-based dialogs, people were recruited using Amazon Mechanical Turk
  - *Several dialog systems used: Markov Decision Process (MDP) or Partially Observed Markov Decision Process (POMDP) for tracking the dialog state, and a hand-crafted policy*
- Labeling procedure followed these principles:
  - First, utterances transcription using Amazon Mechanical Turk – Annotation by heuristics
  - Then, checked & corrected by hand

https://github.com/matthen/dstc/blob/3c7b1eaa6ba08ed712e8cf562c6cd66f121554a0/

DSTC 2 Dataset

How the dialog state is defined in this dataset?

- **Dialog state consists of:**
  - ***Goals***: A distribution over the values of each informable slot in the task – the slot is an informable if the user can give it in the utterance as a constraint for the search
  - ***Method***: A distribution over methods: by name, by constraints, by alternatives or finished – these are the methods that we need to track
  - ***Requested slots*** (the user can also request some slots from the system): A probability for each requestable slot that it has been requested by the user and the system should inform it
- The dataset is marked up in terms of user dialog acts: inform, request, negate, confirm, . . . (the act corresponds to the intent)
  - "What part of town is it?" → `request(area)`
- Method is inferred from act and goals:
  - `inform(food=chinese)` → "by constraints" (search food by "chinese")

DSTC 2 Dialog Excerpt

Utterance	I'm looking for an expensive restaurant with venetian food
Goals	food=venetian, pricerange=expensive
Method	byconstraints
Requested slots	[]

Utterance	Is there one with thai food?
Goals	**food=thai**, pricerange=expensive
Method	byconstraints
Requested slots	[]

Utterance	Can I have the address?
Goals	food=thai, pricerange=expensive
Method	byconstraints
Requested slots	[**addr**]

Dialog example. User: "I'm looking for an expensive restaurant with Venetian food" The state becomes food=Venetian, price range=expensive, and the method is by constraints, and no slots were requested. Then, when the dialog progresses, the user says,"Is there one with Thai food?" Now we need to change the state, all the rest is the same, but food is now Thai. And then, when the user is okay with the options that we have provided, it asks, "Can I have the address?"The state of the dialog is the same, but this time, the requested slots is the address. The three components goals, method, and requested slots are the context that we need to track off to every utterance from the user.

DSTC 2 Results

- Best results after competition:
  - Goals: 65% correct *combinations*
  - Method: 97% correct
  - Requested slots: 95% correct

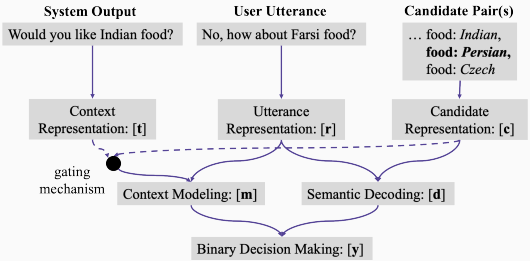
Results of the competition that was held after the collection of this dataset. Regarding the goals, the best solution had 65 percent correct combinations that means that every slot and every value is guessed correctly (that happened in 65% of times). For the method, it has 97% accuracy and requested slots have 95% accuracy. So, slot tagging is the most difficult part. How can you do that state tracking?

Rule-based State Tracking

- Train a good NLU (intents and slots)
- Make simple hand-crafted rules for dialog state change

The state can easily change during the dialog. So, if we train a good NLU, which gives us intents and slots, then we can define hand-crafted rules for dialog state change. If the user mentions a new slot, we just add it to the state. However we can track the state better using neural networks.

Neural Belief Tracker



Joint NLU/DST

https://arxiv.org/pdf/1606.03777.pdf

This architecture uses the previous system output, “Would you like some Indian food?”. Then, it takes the current utterance from the user like, “No, how about Farsi food?”. Then, the system output and user utterance are embedded to come up with a current state of the dialog. And this is done in the following way. First, we embed the context (the system output on the previous state). Then, we embed the user utterance and we also embed candidate pairs for the slot and values, like food-Indian, food-Persian, . . . Next, we use a context modelling network that takes the information about system output, about candidate pairs, uses some type of gating, and uses the user utterance to come up with the idea whether this user utterance effects the context or not. And also, there is a second part which does semantic decoding, so it takes user utterance, the candidate pairs for slot and values, and they decide whether they match or not. And finally, we have a final binary decision making whether these candidate pairs match the user utterance provided the previous system output. This way, we solve NLU and dialog state tracking simultaneously in a joint model.

Neural Belief Tracker Results

DST Model	DSTC2		WOZ 2.0	
	Goals	Requests	Goals	Requests
Delexicalisation-based Model	69.1	95.7	70.8	87.1
Delexicalisation-based Model + Semantic Dictionary	72.9*	95.7	83.7*	87.6
Neural Belief Tracker: NBT-DNN	72.6*	96.4	**84.4***	91.2*
Neural Belief Tracker: NBT-CNN	**73.4***	**96.5**	84.2*	**91.6***

Using Neural Belief Tracker architecture with Convolutional Neural Networks, we can improve accuracy for goals and request accuracy. Solving the task of NLU and dialog state tracking simultaneously, we can get better results.

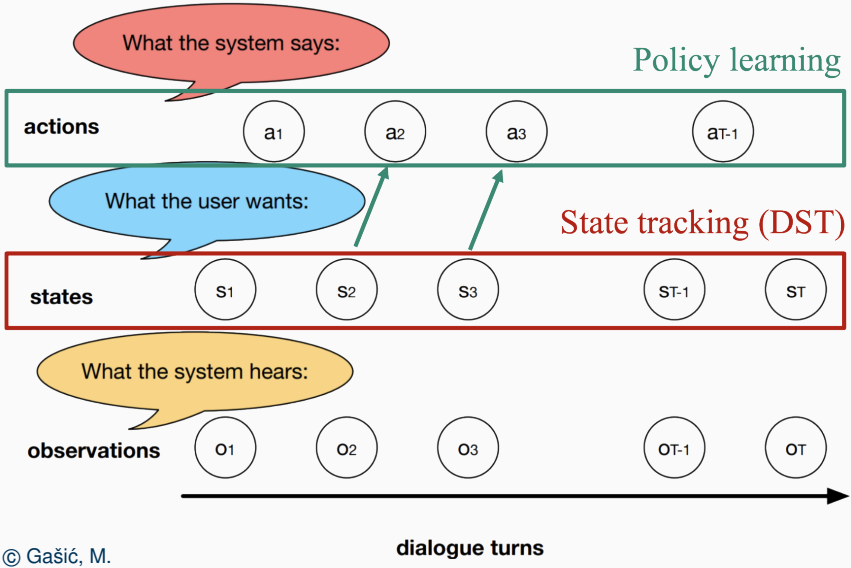
https://arxiv.org/pdf/1606.03777.pdf

Summary

- We've overviewed a state tracker of a DM
- We've discussed the datasets for DM training
- State tracking can be done by hand rules having a good NLU
- Or we can do better with neural network approaches
- Next, we'll deal with dialog policies in DM

Dialog Manager (policy learner)

State Tracking and Policy Learning



Dialog Policy

- A mapping between Dialog state → Agent act
- Policy execution examples:

`inform(location="780 Market St")`
The nearest one is at 780 Market St

`request(location)`
What is the delivery address?

Policy execution examples. A system might inform the user that the location is 780 Market Street. The user will hear it as of the following, “The nearest one is at 780 Market Street”. Another example is that the system might request location of the user. And the user will see it as, “What is the delivery address?”

Simple Approach: Hand Crafted Rules

- We have NLU and state tracker
- We can come up with hand crafted rules for policy

Optimizing Dialog Policies with ML

Supervised learning:

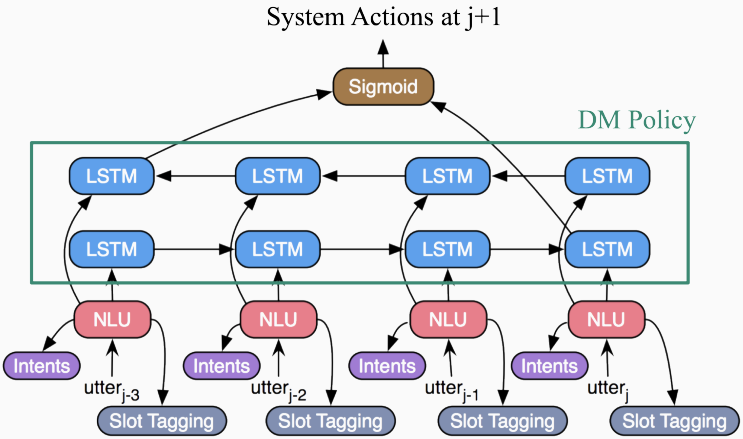
- We train to imitate the observed actions of an expert
- Often requires a large amount of expert-labeled data
- Even with a large amount of training data, parts of the dialogue state space may not be well-covered in the training data

Reinforcement learning:

- Given only a reward signal, the agent can optimize a dialogue policy through interaction with users
- RL can require many samples from an environment, making learning from scratch with real users impractical

https://arxiv.org/pdf/1703.07055.pdf

Joint NLU and DM



Example of a model that does joint NLU and dialog management policy optimization. We have four utterances that we got from the user so far. We pass each of them through NLU which gives us intents and slot tagging, and we can also take the hidden vector, the hidden representation of that phrase from the NLU, and we can use it for a consecutive LSTM that will actually come up with an idea what system action can be executed. Here, we don't need dialog state tracking because we don't have a state. The state here is replaced with a state of the LSTM. Then we can learn a classifier on top of that LSTM, that will output the probability of the next system action.

Joint NLU and DM Results

Model	DM	NLU
Baseline (CRF + SVMs)	7.7	33.1
Pipeline-BLSTM	12.0	36.4
Joint Model	**22.8**	**37.4**

Frame level accuracies on DSTC 4
(it counts only when the whole frame parse is correct)

Comparing 3 models. The first one is a classical approach where a Conditional Random Field is used for slot tagging and a SVM for action classification. The values represent the frame level accuracies, that means that we need to be accurate about everything in the current frame that we have after every utterance. Then, another model is Pipeline-BLSTM, where the NLU is trained separately and then a bidirectional LSTM for dialog policy optimization on top of that model. But these models are trained separately. In the third option these two models, NLU and bidirectional LSTM which was in blue in the previous slide, are trained end to end, jointly, increasing the dialog manager accuracy and improve NLU as well.

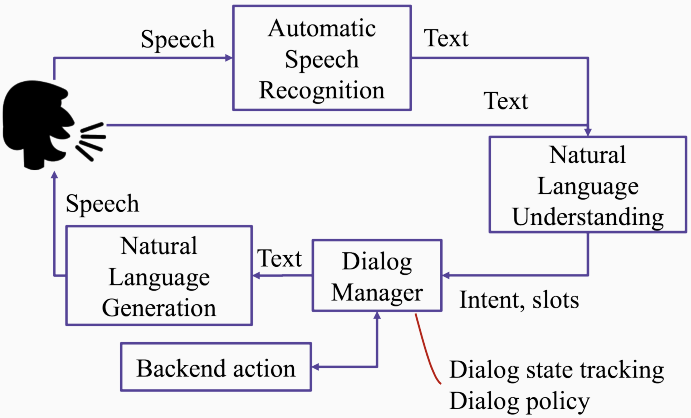
https://arxiv.org/pdf/1612.00913.pdf

Summary

- Dialog policy can be done by hand crafted rules
- Or in a supervised way
- Or in a reinforcement learning way

Overview

Task-oriented Dialog System Overview



The speech of the user is the input of the NLU module. This module outputs intents and slots for the utterance. Then the dialog manager is responsible for two tasks. The first one is dialog state tracking – to understand what the user wanted throughout the conversation and track that state. The second one is dialog policy managing. The policy maps the state to a query in order to get some information from the user or request some information from a backend service (like Google Maps or Yelp). After that, the system is ready to give users some information, using a natural language generator.

Young, S.J., Probabilistic methods in spoken–dialogue systems, 2000.

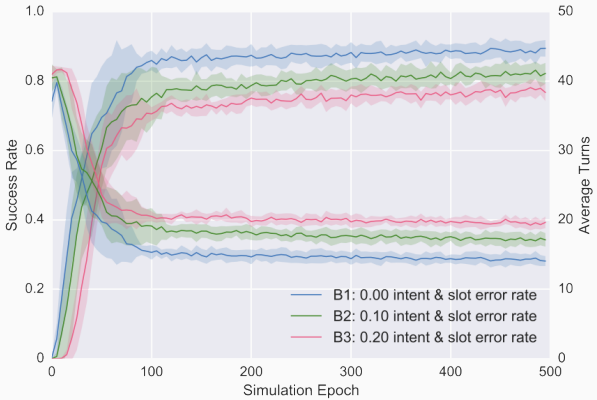
NLU and DM

- Slot Tagger and Intent Classifier in NLU can be trained separately
- Or jointly with better results
- We can use hand-crafted rules sometimes (e.g. dialog policy)
- But learning from data actually works better

NLU and DM

- **NLU**:
  - Turn-level metrics: intent accuracy, slots F1
- **DM**:
  - Turn-level metrics: state tracking accuracy, number of turns, ...
  - Dialog-level metrics: task success rate, reward, ...

NLU Evaluation

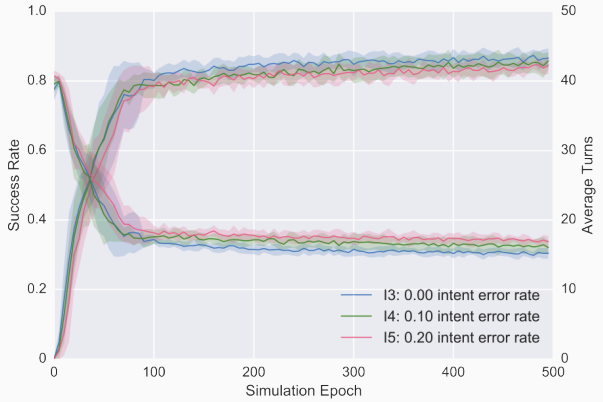


Learning curves for different NLU error

https://arxiv.org/pdf/1703.07055.pdf

NLU evaluation considering that intent classification and the slot tagger are trained separately, in order to understand how the errors of NLU affect the final quality of the Dialog Manager.

NLU Evaluation

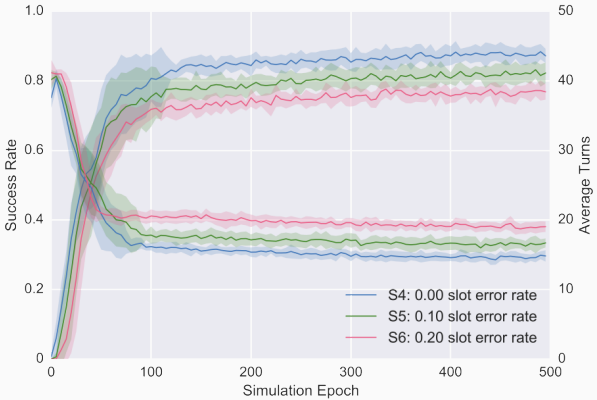


Intent error rate analysis

https://arxiv.org/pdf/1703.07055.pdf

The intent error rate has not a significative influence in the dialog success rate and the the dialogues don't become that much longer when the intent error increases.

NLU Evaluation



Slot error rate analysis

https://arxiv.org/pdf/1703.07055.pdf

It seems that the increase of slot error contributes more to the reduction of the dialog success rate when compared with the intent error rate.