Lecture 4: Dialog system design. GUS and frame-based dialog systems. Alexa Skills Kit.
Background survey results

- Computer Science: 67.2%
- Electrical Engineering
- Mathematics
- Symbolic Systems
- Linguistics
- Psychology
- Management Science and Engineering
- Economics

- Freshmen: 55.7%
- Sophomore: 8.2%
- Junior
- Senior
- Masters
- PhD
Background survey results

Previously Taken (%)

- EE 364a: Convex Optimization I
- CS 329S: Machine Learning Systems Design
- CS236G: Generative Adversarial Networks
- CS 231N: Convolutional Neural Networks for Visual...
- CS 231A: Computer Vision: From 3D Reconstruction to...
- CS 230: Deep Learning
- CS 229: Machine Learning
- CS224W: Machine Learning with Graphs
- CS 224U: Natural Language Understanding
- CS 224N: Natural Language Processing with Deep Learning
- CS 221: Artificial Intelligence Principles
- CS 124: Natural Language
Background survey results

Currently Taking (%)

- EE 364a: Convex Optimization I
- CS 229: Machine Learning Systems Design
- CS 236G: Generative Adversarial Networks
- CS 231N: Convolutional Neural Networks for Visual...
- CS 231A: Computer Vision: From 3D Reconstruction to...
- CS 230: Deep Learning
- CS 229: Machine Learning
- CS 224W: Machine Learning with Graphs
- CS 224U: Natural Language Understanding
- CS 224N: Natural Language Processing with Deep Learning
- CS 221: Artificial Intelligence Principles
- CS 124: Natural Language
Background survey results

Seen this concept before? (%)

- Reinforcement Learning (MDPs, POMDPs, Bellman Eqns, etc.): 70%
- Hidden Markov Models: 60%
- Mathematics for computing gradients for Neural Networks: 80%
- Transformer / Encoder-Decoder architectures with attention: 80%
- Recurrent neural networks or LSTMs: 80%
- Convolutional Neural Networks (CNNs): 90%
- Multiple layer Neural Networks (DNNs): 80%
- Matrix multiplication: 90%
- Linear Algebra: 90%
Background survey results

Technology Experience (%)

- Training/optimizing a neural network
- Implementing a Hidden Markov Model in code
- Experiments with reinforcement learning approaches / models
- PyTorch for Neural Network training
- Tensorflow for Neural Network training
- PyTorch Lightning Framework
- Jupyter notebook for experiments or homeworks
- Weights & Biases experiment management tools
- Google Cloud Platform
- Amazon Web Services
- Microsoft Azure Cloud Compute
Background survey results

Goals for the course? (%)

- Gain experience to work in industry focused on Spoken Language / NLP
- No specific usage / academic interest only
- Build personal projects with speech-enabled tools
- Learn more about deep learning theory and recent architectures
- Learn to use PyTorch and train DNNs
- Learn to build Alexa skills
- Apply material from the course to your current job / main project
- Incorporate aspects of spoken language analysis or dialog agents into your research area
- Build towards a research paper in this area
Outline

- Dialog system design
- GUS and frame-based systems
- NLU and NLG considerations
- Alexa skills kit overview
Spoken Dialog Agent Conceptual Architecture

Speech Recognition → Natural Language Understanding → Dialogue Manager → Task Manager

→ Text-to-Speech Synthesis → Natural Language Generation
System design considerations

- Goal and scope of overall system?
  - What tasks/actions are supported?
  - What state do the dialog/task managers track?
- What level of interaction complexity?
  - Initiative? Back-tracking? NLU support for paraphrasing?
- Need a solution for each module (ASR, TTS, NLU, NLG, task/dialog manager)
- What is the interface / data structure between modules?
  - e.g. Does ASR module send transcripts only? Emotion labels? Audio?
Dialog System Design: User-centered Design

Gould and Lewis 1985

1. Study the user and task
2. Build simulations "Wizard of Oz study"
3. Iteratively test the design on users
Rough system design process

1. Overall system goal.
2. Define set of task actions system can perform
3. Create example interactions
4. Define dialog manager approach (actions + dialog acts/state of system)
5. Choose NLU approach matching complexity of tasks and approach to initiative + dialog acts
6. Define NLG approach and dialog state -> NLG interface
7. Create a dialog policy (choosing next dialog action and sending to NLG)
8. Choose ASR/TTS approach. Update NLU/NLG if needed
System design considerations

- Not all systems require support for complex interactions
- Frameworks like Alexa force some choices about multiple modules to simplify overall development
- ASR/TTS components often be treated as black-box, but great systems are sensitive to ASR uncertainty
- Okay to redefine/combine modules based on problem (e.g. a smart NLG module might simplify dialog manager)
Frame-based dialog agents

- One or more **frames**
  - Each a collection of **slots**
  - Each slot having a **value**
- **Domain ontology** is full collection of frames for a (sub)system
- A frame defines the task data structure
The Frame

- A set of **slots**, to be filled with information of a given **slot type**
- Each associated with a **question** to the user

<table>
<thead>
<tr>
<th>Slot</th>
<th>Type</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORIGIN</td>
<td>city</td>
<td>What city are you leaving from?</td>
</tr>
<tr>
<td>DEST</td>
<td>city</td>
<td>Where are you going?</td>
</tr>
<tr>
<td>DEP DATE</td>
<td>date</td>
<td>What day would you like to leave?</td>
</tr>
<tr>
<td>DEP TIME</td>
<td>time</td>
<td>What time would you like to leave?</td>
</tr>
<tr>
<td>AIRLINE</td>
<td>line</td>
<td>What is your preferred airline?</td>
</tr>
</tbody>
</table>
Frame-based dialogue agents

1977:

GUS, A Frame-Driven Dialog System¹

Daniel G. Bobrow, Ronald M. Kaplan, Martin Kay, Donald A. Norman, Henry Thompson and Terry Winograd

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, U.S.A.

Artificial Intelligence Journal, 1977

- Still the industrial state of the art
- Alexa roughly follows GUS architecture
GUS: Hello. My name is GUS. I can help you plan a simple trip by air. Where do you want to go?
Client: I want to go to San Diego on May 28
GUS: What time do you want to leave?
Client: I must be in San Diego before 10 am
GUS: Would you like P.S.A. flight 102 that arrives at 9.15 am?
Client: What is the next flight?
GUS: Air California flight 310 that leaves at 8.30 am and arrives at 10.00 am
Client: I’ll take the first one
GUS: In what name should I make a reservation, please?
Client: Dan Bobrow
GUS: I have confirmed the following flight: P.S.A. flight 102 on Wednesday May 28 from San Jose to San Diego that leaves at 7.30 am and arrives at 9.15 am.
GUS: What date do you want to return on?
Client: On Friday in the evening.
GUS: Would you like the flight that leaves at 7.45 pm?
Client: That’s fine.
GUS: I have confirmed the following flight: P.S.A. flight 307 on Friday May 30 from San Diego to San Jose that leaves at 7.45 pm and arrives at 9.30 pm. Thank you for calling. Goodbye.
Slot types can be complex, hierarchical

- The type *DATE*

```
DATE
  MONTH NAME
  DAY (BOUNDED-INTEGER 1 31)
  YEAR INTEGER
WEEKDAY (MEMBER (SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY))
```
Simple action policy for frame-based dialog

Consider a trivial airline travel system:
  Ask the user for a departure city
  Ask for a destination city
  Ask for a time
  Ask whether the trip is round-trip or not
Finite State Dialog Manager

1. What city are you leaving from?
2. Where are you going?
3. What date do you want to leave?
4. Is it a one-way trip?
   - Yes: Do you want to go from <FROM> to <TO> on <DATE>?
   - No: What date do you want to return?
     - Yes: Do you want to go from <FROM> to <TO> on <DATE> returning on <RETURN>?
     - No: Book the flight

Finite-state dialog managers

- System completely controls the conversation with the user.
- It asks the user a series of questions
- Ignoring (or misinterpreting) anything the user says that is not a direct answer to the system’s questions
Frames and mixed initiative

- System asks questions of user, filling any slots that user specifies
  - When frame is filled, do database query
- If user answers 3 questions at once, system can fill 3 slots and not ask these questions again!
- Frame structure guides dialog
Mixed Initiative

- Conversational initiative can shift between system and user
- Simplest kind of mixed initiative: use the structure of the **frame** to guide dialogue

<table>
<thead>
<tr>
<th>Slot</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORIGIN</td>
<td>What city are you leaving from?</td>
</tr>
<tr>
<td>DEST</td>
<td>Where are you going?</td>
</tr>
<tr>
<td>DEPT DATE</td>
<td>What day would you like to leave?</td>
</tr>
<tr>
<td>DEPT TIME</td>
<td>What time would you like to leave?</td>
</tr>
<tr>
<td>AIRLINE</td>
<td>What is your preferred airline?</td>
</tr>
</tbody>
</table>
NLU and NLG with frame-based systems
Natural Language Understanding for filling dialog slots

1. Domain classification
   Asking weather? Booking a flight?
   Programming alarm clock?

2. Intent Determination
   Find a Movie, Show Flight, Remove Calendar Appt

3. Slot Filling
   Extract the actual slots and fillers
Natural Language Understanding for filling slots

Show me morning flights from Boston to SF on Tuesday.

DOMAIN: AIR-TRAVEL
INTENT: SHOW-FLIGHTS
ORIGIN-CITY: Boston
ORIGIN-DATE: Tuesday
ORIGIN-TIME: morning
DEST-CITY: San Francisco
Wake me tomorrow at six.

**DOMAIN:** ALARM-CLOCK  
**INTENT:** SET-ALARM  
**TIME:** 2017-07-01 0600-0800
Rule-based Slot-filling

Write regular expressions or grammar rules

Wake me (up) | set (the|an) alarm | get me up

Do text normalization

Time consuming and brittle NLU capabilities
Machine learning for slot-filling

I want to fly to San Francisco on Monday afternoon please

Use 1-of-N classifier for Domain/Intent. Use CRF/sequence model to tag words/phrases with slot names

- **Input:**
  - features like word N-grams

- **Output:**
  - Domain: AIRLINE
  - Intent: SHOWFLIGHT
  - Destination-City: “San Francisco”
  - Depart-Date: “Monday”
More sophisticated algorithm for slot filling: IOB Tagging

- IOB Tagging
  - tag for the beginning (B) and inside (I) of each slot label,
  - plus one for tokens outside (O) any slot label.
  - \(2n + 1\) tags, where \(n\) is the number of slots.

```
B-DESTINATION
I-DESTINATION
B-DEPART_TIME
I-DEPART_TIME
O
```

```
0 0 0 0 0 B-DES I-DES 0 B-DEPTIME I-DEPTIME 0
I want to fly to San Francisco on Monday afternoon please
```
More sophisticated algorithm for slot filling: IOB Tagging

- IOB Tagging is done by a sequence model
- Typical:

Extracted strings can then be normalized (San Francisco -> SFO)
Generation Component (NLG)

- **Content Planner**
  - Decides what content to express to user
    (ask a question, present an answer, etc)
  - Often merged with dialogue manager

- **Language Generation**
  - Chooses syntax and words

- **TTS**

- **In practice**: Template-based w/most words prespecified
  What time do you want to leave CITY-ORIG?
  Will you return to CITY-ORIG from CITY-DEST?
More sophisticated NLG

- Dialogue manager builds representation of meaning of utterance to be expressed
- Passes this to a “generator”
- Increasingly generators are deep learning systems
- Mixing chatbot-like NLG constrained to convey dialog representation can improve user satisfaction
Alexa Skills Kit

- A *Skill* is a top level command for Alexa.
  - “Alexa open 224S Homework 2”
  - Skill $\rightarrow$ **domain ontology**
- A skill contains *intents* which are distinct task actions.
  - Intent $\rightarrow$ **frame**
  - Design intents with built-in capabilities per intent and ASK interaction model in mind
- Each intent contains *slots* which each have a *slot type* and take on a *slot value*
- Not quite this simple (e.g. ASK **built-in intents** are not simple to define in the frame/slot abstraction)
Alexa Skills Kit

- Dialog management is complex, partially handled with built-in features (clarification, value verification, cancel skill, etc).
- NLU through grammars and examples.
  - ASK trains models for you based on examples
  - Many rich slot types (dates, numbers, lists)
- Task management is custom! ASK provides a dialogue API to your web server, you implement server-side task execution.
- NLG is template-based with ASK adding variety
- ASR/TTS handled by ASK. Interface is text/transcripts
- Overall framework is API/SDK oriented like web dev
Alexa domain classification

Figure 1: The overall architecture of the personalized dynamic domain classifier.

(Kim et al, 2018)
## ASK interaction schema

### Interaction Model

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>languageModel</td>
<td>object</td>
<td>Conversational primitives for the skill</td>
<td>yes</td>
</tr>
<tr>
<td>dialog</td>
<td>object</td>
<td>Rules for conducting a multi-turn dialog with the user</td>
<td>no</td>
</tr>
<tr>
<td>prompts</td>
<td>array</td>
<td>Cues to the user on behalf of the skill for eliciting data or providing feedback</td>
<td>no</td>
</tr>
</tbody>
</table>

### languageModel

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>invocationName</td>
<td>string</td>
<td>Invocation name of the skill</td>
<td>yes</td>
</tr>
<tr>
<td>intents</td>
<td>array</td>
<td>Intents and their slots</td>
<td>yes</td>
</tr>
<tr>
<td>types</td>
<td>array</td>
<td>Custom slot types</td>
<td>no</td>
</tr>
<tr>
<td>modelConfiguration</td>
<td>object</td>
<td>Optional settings for the interaction model. Available in supported locales.</td>
<td>no</td>
</tr>
</tbody>
</table>

### languageModel_intents

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>Name of the intent. For details about intent names, see Intent and slot name requirements.</td>
<td>yes</td>
</tr>
<tr>
<td>slots</td>
<td>array</td>
<td>List of slots within the intent.</td>
<td>no</td>
</tr>
<tr>
<td>samples</td>
<td>array</td>
<td>Sample utterances for the intent</td>
<td>no</td>
</tr>
</tbody>
</table>
This example shows a portion of the intent object for a PlanMyTrip intent. The utterances for the intent are in interactionModel.languageModel.intents[].samples. Each slot has its own samples array. For brevity, other properties within interactionModel and languageModel are not shown.

```json
{
  "interactionModel": {
    "languageModel": {
      "intents": [
        {
          "name": "PlanMyTrip",
          "slots": [
            {
              "name": "travelDate",
              "type": "AMAZON.DATE",
              "samples": ["I am taking this trip on {travelDate}", "on {travelDate}", "{travelDate}\] },
            {
              "name": "toCity",
              "type": "AMAZON.US_CITY",
              "samples": ["I'm going to {toCity}", "{toCity}\] },
            {
              "name": "fromCity",
              "type": "AMAZON.US_CITY",
              "samples": ["I'm starting from {fromCity}\] },
            {
              "name": "travelMode",
              "type": "LIST_OF_TRAVEL_MODES",
              "samples": ["I am going to {travelMode}\], "{travelMode}\] },
            {
              "name": "activity",
              "type": "LIST_OF_ACTIVITIES",
              "samples": ["{activity}\", "I plan to {activity}\] } ]
        }]
      }
    }
  }
}
```

"samples": ["{toCity}\", "I want to travel from {fromCity} to {toCity} \{travelDate\}\", "i want to visit {toCity}\", "i am going on trip on \{travelDate\}\", "i'm \{travelMode\} from {fromCity} to {toCity}\", "i'm \{travelMode\} to {toCity} to \{activity\}\", "plan a trip\", "plan a trip to {toCity}\", "plan a trip starting from {fromCity}\", "I'd like to leave on \{travelDate\}\", "I'd like to leave on the \{travelDate\}\", "I'd like to fly out of {fromCity}\] ] ] ]}
Alexa Conversations (new in 2020)

When you build an Alexa Conversations skill, you create the following components that train Alexa Conversations how to interact with your user.

(About, ASK docs)
Intents / ice_cream

Sample Utterances (15)

- What might a user say to invoke this intent?

  - I would like **[num_scoops]** scoops of **[flavor]** in a **[container]** with **[toppings_one]** and **[toppings_two]**
  - I would like **[num_scoops]** scoops of **[flavor]** in a **[container]** with **[toppings_one]** please
  - A **[container]** with **[num_scoops]** scoops of **[flavor]**
  - One ice cream please
  - A **[container]** with **[flavor]** **[num_scoops]** scoops