Machine learning for Spoken language understanding

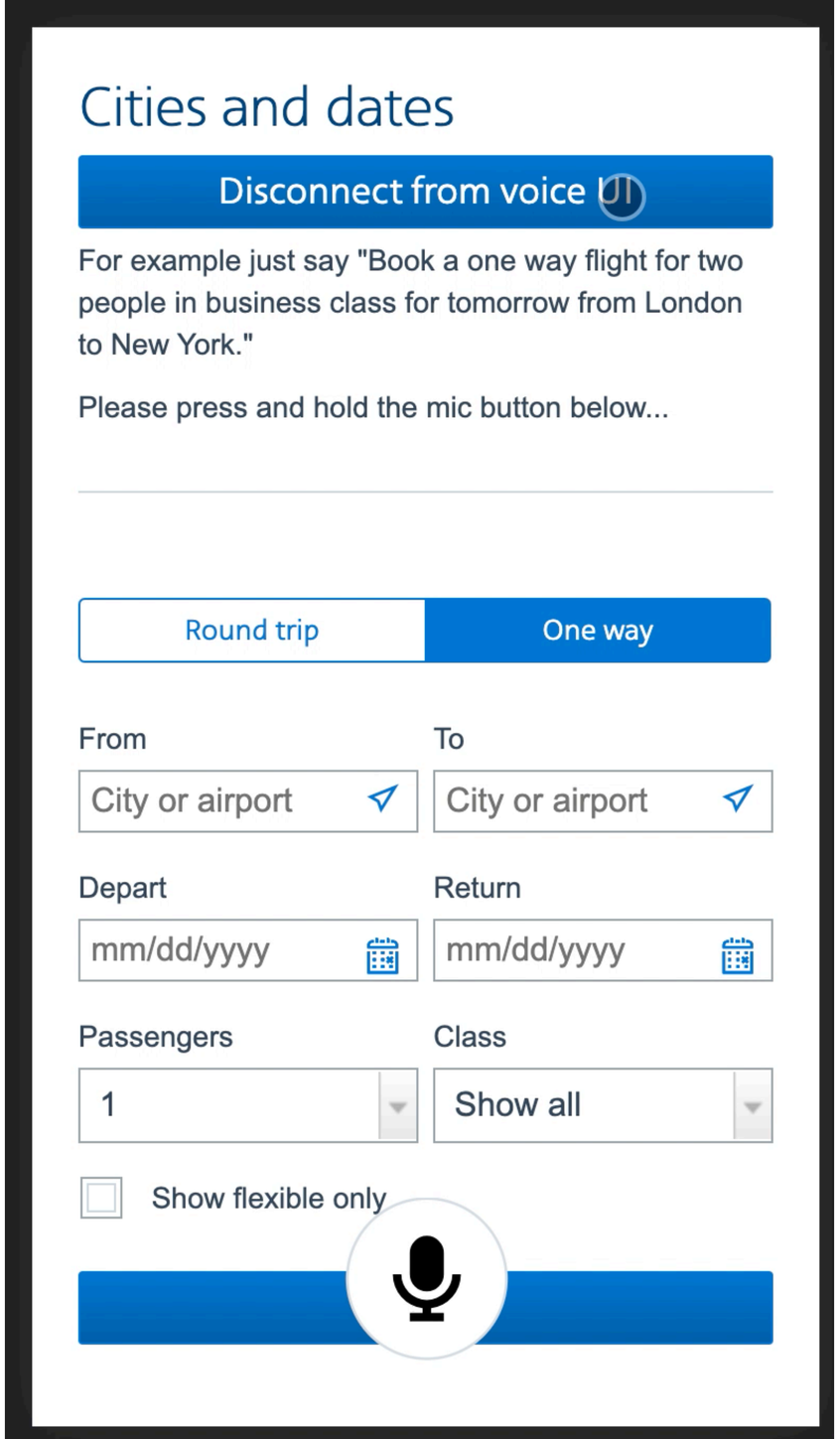
Antti Ukkonen
antti@speechly.com

Please forget **everything** you might know about chatbots or voice assistants.

The problem with current voice tech…

- Voice assistants (Siri, Alexa, Google's assistant, etc.) try to mimic a human. This is a hard problem.
- The user experience can be cumbersome, because the interaction between the user and device is limited to a slow "conversation".
  - If the device does not understand, the user gets feedback of this only after they finish talking and the device has responded (in voice!) something.
- But: voice input itself is not a bad idea, it's just the current implementations of it are not so great. (As an analogy: Think of pre-iPhone touchscreen phones…)

Our technology enables developers to easily build responsive multi-modal user interfaces where voice seamlessly integrates with touch input and visual feedback.



https://www.youtube.com/watch?v=XWqHV1a32LM

DEPARTMENT | PRODUCT | BRAND | COLOR | SIZE | SORT BY

https://www.youtube.com/watch?v=xI68NT8D1m8

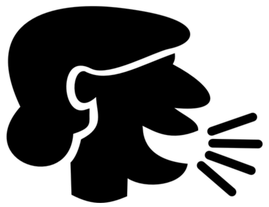
Some definitions….

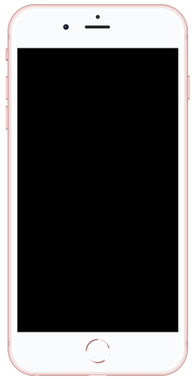
- An **utterance** is something that a user says.
  For example: “turn off the living room lights”

- The **intent** of an utterance is **the task the user wants to get done**. Above the user most likely wants to `turn_off` something.
  There might also be a `turn_on` intent, as well as others, depending on what other actions the application can do.

- **Entities** are **“parameters” of the intent**. The `turn_off` intent might take as parameters the `device` to be turned off, and the `location` where the device is. (Which would be “lights” and “living room” in case of this example.)

“i’m looking for a blue winter coat”





Designs and implements the **front end** UI.

Speechly SDKs

{entity_name: **“color”**,
entity_value: **”blue”**}

{entity_name: **“category”**,
entity_value: **”jackets”**}

Speechly API

Entities are returned in **real-time** while the user is speaking.

Defines **intents** and **entities** that are relevant for the application.

Speechly ML Engine

Model deployed to production.

Autonomously trains **customised ML models** for speech recognition and intent / entity detection.

Configuring the application

- This is done by providing examples of utterances the user might say using a simple markdown like syntax to annotate the entities:

```
*add_search_filter I'm looking for a [blue](color) [winter coat](category)
```

Intent name

Entity value

Entity name

(Obviously users might say the same thing in a number of ways…)

- Note that the entities are completely contextual! You can define exactly those entities that are relevant in your application. (A lot of "off the shelf" NLP technology relies on predefined entities, e.g. numbers, dates, locations, etc.)

The Speechly Annotation Language

Example of a configuration for a very simple photo editing app

```
filter = [vintage|faded|sepia|classic|kodachrome|technicolor|polaroid|black and white|grayscale]
property = [brightness | luminosity | light | contrast | saturation | color]
increase_cmd = [increase|add more|more]
decrease_cmd = [decrease|reduce|less]
add_filter_cmd = [add | activate | make it | make it look | make it like | change it to]
del_filter_cmd = [remove | deactivate]

*undo [go back | go one step back | undo | i don't want that | no i don't want that | cancel that | cancel]

*add_filter $add_filter_cmd $filter(filter)
*remove_filter $del_filter_cmd $filter(filter)

*increase $increase_cmd $property(property)
*decrease $decrease_cmd $property(property)

*increase $increase_cmd $property(property) {and} *increase $increase_cmd $property(property)
*decrease $decrease_cmd $property(property) {and} *decrease $decrease_cmd $property(property)

*increase $increase_cmd $property(property) {and} *decrease $decrease_cmd $property(property)
*decrease $decrease_cmd $property(property) {and} *increase $increase_cmd $property(property)

*add_filter $add_filter_cmd $filter(filter) {and} *increase $increase_cmd $property(property)
*add_filter $add_filter_cmd $filter(filter) {and} *decrease $decrease_cmd $property(property)

*increase $increase_cmd $property(property) {and} *add_filter $add_filter_cmd $filter(filter)
*decrease $decrease_cmd $property(property) {and} *add_filter $add_filter_cmd $filter(filter)
```

Machine learning challenges at Speechly

- We must develop a state-of-the-art **ASR** system.
  - This requires **thousands of hours of transcribed speech data**, and training a model can take up to **two weeks on several GPUs**.
- We must train **domain-specific models** for **intent** and **entity** detection.
  - They must train in a **few minutes**,
    and it must be done with rather **limited data**.
- All of our models must be **streaming**, meaning they must make their predictions without seeing the input until the end.

So how does that relate to what is out there in the world?

- Speech recognition is a **sequence-to-sequence mapping** problem.
  (Sounds like a case for deep neural networks!)
- Intent detection is a **text classification** problem.
  (Sounds like we need some classifiers!)
- Entity detection is a **sequence tagging** problem.
  (Sounds like we might need some conditional random fields!)
- But again, those must work in a streaming setting.
  And be trainable in an autonomous system without any human involvement.

Case example: Streaming intent detection

Possible intents are: `turn_on` **and** `turn_off`**.**

turn
turn the
turn the lights
turn the lights in
turn the lights in the
turn the lights in the living
turn the lights in the living room
turn the lights in the living room **on**

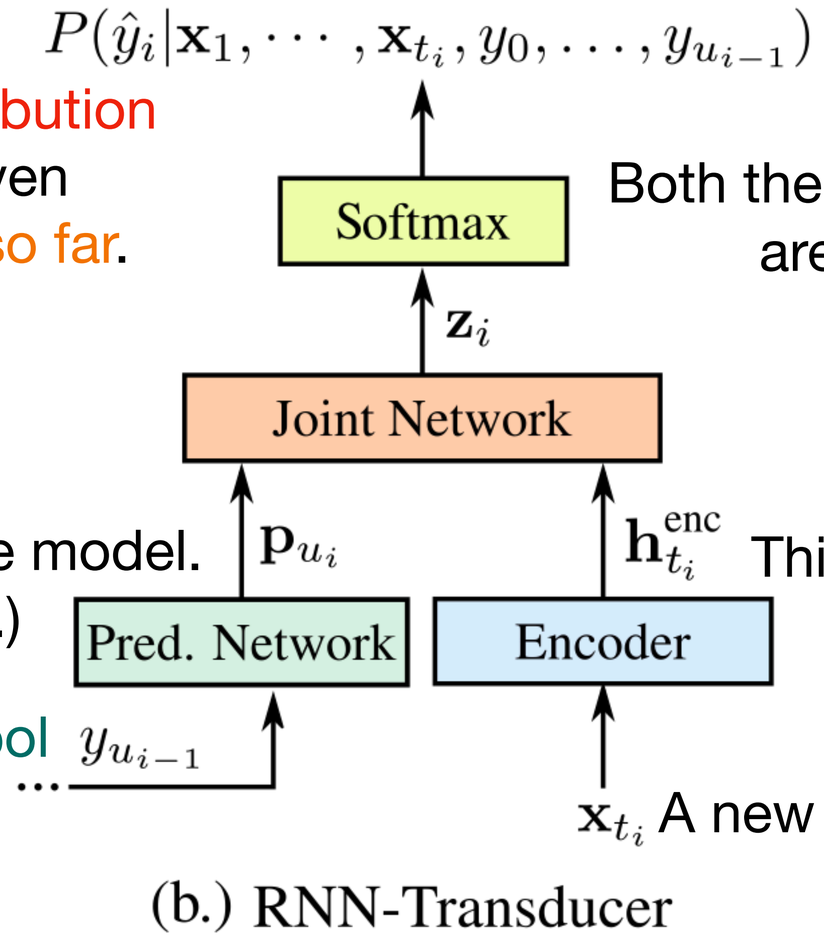
The model must learn to wait and keep listening until it is certain enough that it can make a correct prediction.

The configuration (nor any other data) has any explicit information about when the model can stop.

We use a simple model inspired by reinforcement learning to solve this.

Another example: End-to-end ASR

using Recurrent Neural Network Transducers (Graves, ICML2012)



(b.) RNN-Transducer

Output is a probability distribution over the next symbol given what the model has seen so far.

Both the **predictor network** and **encoder** are relatively "simple" but **BIG** deep learning models.

This is a "neural" language model. (Kind of like GPT-x.)

This is an audio feature extractor.

The previous output symbol goes in here…

A new frame of audio goes in here…

(He et al 2018, "Streaming end-to-end speech recognition for mobile devices")

About ways of working…

- We build our in-house technology from scratch, but are heavily inspired by recent research.
- We probably could publish some of our ML stuff, but things are hectic as is…
- Everyone at Speechly is also a software engineer.
  (E.g. no separate data scientist roles.)
- We use PyTorch for ML, Docker/Kubernetes/etc on AWS/GCP for infra.
  And a ton of post-it notes for planning.
- Keep things simple but not too simple.

The most important thing, however, is….

Data

Thanks!