Recent advances in applied chatbot technology to be presented at AI Ukraine 2017

Jordi Carrera Ventura
NLP scientist
Telefónica Research & Development
Outline

Overview

Industrial state-of-the-art

Current challenges

Recent advances

Conclusions
Overview
Chatbots are a form of conversational artificial intelligence (AI) in which a user interacts with a virtual agent through natural language messaging in a messaging interface like Slack or Facebook Messenger or a voice interface like Amazon Echo or Google Assistant.

Chatbot
A bot that lives in a chat (an automation routine inside a UI).

Conversational agent

Virtual Assistant
A bot that takes natural language as input and returns it as output.

Chatbot
A virtual assistant that lives in a chat.

1 Generally assumed to have broader coverage and more advanced AI than chatbots.

User> what are you?
Use cases

Usually intended to
• get quick answers to a specific questions over some pre-defined repository of knowledge
• perform transactions in a faster and more natural way.

Can be used to
• surface contextually relevant information
• help a user complete an online transaction
• serve as a helpdesk agent to resolve a customer’s issue without ever involving a human.

Virtual assistants for
Customer support, e-commerce, expense management, booking flights, booking meetings, data science...
What's really important...

Conversational technology should really be about

• dynamically finding the best possible way to browse a large repository of information/actions.

• find the shortest path to any relevant action or piece of information (to avoid the *plane dashboard* effect).
What's really important...

Conversational technology should really be about

• dynamically finding the best possible way to browse a large repository of information/actions.

• find the shortest path to any relevant action or piece of information (to avoid the plane dashboard effect).

• surfacing implicit data in unstructured content ("bocadillo de calamares in Madrid"). Rather than going open-domain, taking the closed-domain and going deep into it.
... and the hype

Macy's On-Call

Welcome to Macy's On-Call at Los Angeles Downtown Plaza! We'll help you find anything you need while you shop. If at any time you'd like to start over, just type "Start over." Happy shopping!

Me

Hi! I'm a man and I'm looking for a leather jacket
... and the hype

Macy's On-Call
Welcome to Macy's On-Call at Los Angeles Downtown Plaza! We'll help you find anything you need while you shop. If at any time you'd like to start over, just type "Start over." Happy shopping!

Me
Hi! I'm a man and I'm looking for a leather jacket

Macy's On-Call
We've got what you're looking for!
Head to Floor 1 near Men's Collections
For more info on colors, sizes & more, click the image below.

Michael Michael Kors Men's Lea...
... and the hype

that's a nice one but what if I want to browse different jackets? is there an easy way of doing it?
... and the hype

Me
Today 09:56 AM

that's a nice one but what if I want to browse different jackets? Is there an easy way of doing it?

Macy's On-Call
Today 09:56 AM

We've got what you're looking for!
Head to Floor 1 near Men's Collections
For more info on colors, sizes & more, click the image below.

Marc New York Men's Leather Moto...
... and the hype

Me

where can I find a pair of shoes that match this jacket?

Today 09:57 AM
... and the hype

Me   Today 09:57 AM
where can I find a pair of shoes that match this jacket?

Macy's On-Call   Today 09:57 AM
We've got what you're looking for!
Head to Floor 1 near Men's Collections
For more info on colors, sizes & more, click the image below.

Michael Kors Men's Leather Rac...
... and the hype
... and the hype

Macy's On-Call

We've got what you're looking for!
Head to Floor 1 near Men's Collections
For more info on colors, sizes & more, click the image below.

Rockport Style Leader 2 Whitne...
... and the hype

That was IBM Watson 😳
Industrial state-of-the-art
Industrial state-of-the-art

Lack of suitable metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Twitter</th>
<th></th>
<th></th>
<th>Ubuntu</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Spearman</td>
<td>p-value</td>
<td>Pearson</td>
<td>p-value</td>
<td>Spearman</td>
<td>p-value</td>
</tr>
<tr>
<td>Greedy</td>
<td>0.2119</td>
<td>0.034</td>
<td>0.1994</td>
<td>0.047</td>
<td>0.05276</td>
<td>0.6</td>
</tr>
<tr>
<td>Average</td>
<td>0.2259</td>
<td>0.024</td>
<td>0.1971</td>
<td>0.049</td>
<td>-0.1387</td>
<td>0.17</td>
</tr>
<tr>
<td>Extrema</td>
<td>0.2103</td>
<td>0.036</td>
<td>0.1842</td>
<td>0.067</td>
<td>0.09243</td>
<td>0.36</td>
</tr>
<tr>
<td>METEOR</td>
<td>0.1887</td>
<td>0.06</td>
<td>0.1927</td>
<td>0.055</td>
<td>0.06314</td>
<td>0.53</td>
</tr>
<tr>
<td>BLEU-1</td>
<td>0.1665</td>
<td>0.098</td>
<td>0.1288</td>
<td>0.2</td>
<td>-0.02552</td>
<td>0.8</td>
</tr>
<tr>
<td>BLEU-2</td>
<td>0.3576</td>
<td>&lt; 0.01</td>
<td>0.3874</td>
<td>&lt; 0.01</td>
<td>0.03819</td>
<td>0.71</td>
</tr>
<tr>
<td>BLEU-3</td>
<td>0.3423</td>
<td>&lt; 0.01</td>
<td>0.1443</td>
<td>0.15</td>
<td>0.0878</td>
<td>0.38</td>
</tr>
<tr>
<td>BLEU-4</td>
<td>0.3417</td>
<td>&lt; 0.01</td>
<td>0.1392</td>
<td>0.17</td>
<td>0.1218</td>
<td>0.23</td>
</tr>
<tr>
<td>ROUGE</td>
<td>0.1235</td>
<td>0.22</td>
<td>0.09714</td>
<td>0.34</td>
<td>0.05405</td>
<td>0.5933</td>
</tr>
<tr>
<td>Human</td>
<td>0.9476</td>
<td>&lt; 0.01</td>
<td>1.0</td>
<td>0.0</td>
<td>0.9550</td>
<td>&lt; 0.01</td>
</tr>
</tbody>
</table>

Table 3: Correlation between each metric and human judgements for each response. Correlations shown in the human row result from randomly dividing human judges into two groups.

Industrial state-of-the-art Benchmark

By Intento (https://inten.to)

Dataset
SNIPS.ai 2017 NLU Benchmark
https://github.com/snipsco/nlu-benchmark

Example intents

SearchCreativeWork (e.g. Find me the I, Robot television show)
GetWeather (e.g. Is it windy in Boston, MA right now?)
BookRestaurant (e.g. I want to book a highly rated restaurant for me and my boyfriend tomorrow night)
PlayMusic (e.g. Play the last track from Beyoncé off Spotify)
AddToPlaylist (e.g. Add Diamonds to my roadtrip playlist)
RateBook (e.g. Give 6 stars to Of Mice and Men)
SearchScreeningEvent (e.g. Check the showtimes for Wonder Woman in Paris)
Industrial state-of-the-art

Benchmark

Methodology

English language.

Removed duplicates that differ by number of whitespaces, quotes, lettercase, etc.

Resulting dataset parameters

7 intents, 15.6K utterances (~2K per intent)

3-fold 80/20 cross-validation.

Most providers do not offer programmatic interfaces for adding training data.
# Industrial state-of-the-art

<table>
<thead>
<tr>
<th>Framework</th>
<th>Since</th>
<th>F1</th>
<th>False positives</th>
<th>Response time</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Watson Conversation</td>
<td>2015</td>
<td>99.7</td>
<td>100%</td>
<td>0.35</td>
<td>2.5</td>
</tr>
<tr>
<td>API.ai (Google 2016)</td>
<td>2010</td>
<td>99.6</td>
<td>40%</td>
<td>0.28</td>
<td>Free</td>
</tr>
<tr>
<td>Microsoft LUIS</td>
<td>2015</td>
<td>99.2</td>
<td>100%</td>
<td>0.21</td>
<td>0.75</td>
</tr>
<tr>
<td>Amazon Lex</td>
<td>2016</td>
<td>96.5</td>
<td>82%</td>
<td>0.43</td>
<td>0.75</td>
</tr>
<tr>
<td>Recast.ai</td>
<td>2016</td>
<td>97</td>
<td>75%</td>
<td>2.06</td>
<td>N/A</td>
</tr>
<tr>
<td>wit.ai (Facebook)</td>
<td>2013</td>
<td>97.4</td>
<td>72%</td>
<td>0.96</td>
<td>Free</td>
</tr>
<tr>
<td>SNIPS</td>
<td>2017</td>
<td>97.5</td>
<td>26%</td>
<td>0.36</td>
<td>Per device</td>
</tr>
</tbody>
</table>
Leaning curve by provider

Vertical bars denote confidence intervals

Logarithmic scale!
Current challenges
So, let's be real...

Welcome to Macy's blah-blah...!

Hi, I'm a guy looking for Leda jackets.

We've got what you're looking for!

That's a nice one, but I want to see different jackets. How can I do that?
So, let's be real...

We've got what you're looking for!
<image product_id=1 item_id=2>

where can i find a pair of shoes that match this yak etc.

We've got what you're looking for!
<image product_id=1 item_id=3>

pair of shoes
(, you talking dishwasher)
So, let's be real...

Welcome to Macy's blah-blah...!

hi im a **guy** looking for **leda jackets**
So, let's be real...

lexical variants: synonyms, paraphrases

**Intent**

I have a problem with my cable.

- My cable does not work.
- Can someone fix my cable?
- I can’t watch TV.
- Has my cable service expired?
- I am not able to get a reception.
- Is there an outage?
- Cable TV is not loading.
- Cable is kaput
- My TV doesn’t work
- I can’t get my cable responding
  
  ... ...
So, let's be real...

lexical variants:
synonyms, paraphrases

Intent
I have a problem with my cable.

My cable does not work.
Can someone fix my cable?
I can’t watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn’t work
I can’t get my cable responding
...
...

My cable does not work.
Can someone fix my cable?

I can’t watch TV.

Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn’t work
I can’t get my cable responding
...
...

Entity
I am not able to watch TV
I am not able to watch television
I am not able to watch cable
I am not able to watch cable TV
I cannot watch TV
I cannot watch television
I cannot watch cable
I cannot watch cable TV
Cannot watch TV
Cannot watch television
Cannot watch cable
Cannot watch cable TV
Can’t watch TV
Can’t watch television
Can’t watch cable
Can’t watch cable TV
So, let's be real...

lexical variants:
synonyms, paraphrases

Intent
I have a problem with my cable.

My cable does not work.
Can someone fix my cable?
I can’t watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn’t work
I can’t get my cable responding
... ...

My cable does not work.
Can someone fix my cable?
I can’t watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn’t work
I can’t get my cable responding
... ...

This can be normalized, e.g.
regular expressions

I am not able to watch TV
I am not able to watch television
I am not able to watch cable
I am not able to watch cable TV
I cannot watch TV
I cannot watch television
I cannot watch cable
I cannot watch cable TV
Can’t watch TV
Can’t watch television
Can’t watch cable
Can’t watch cable TV
So, let's be real...

lexical variants: synonyms, paraphrases

Intent
I have a problem with my cable.

My cable does not work.
Can someone fix my cable?
I can't watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn't work
I can't get my cable responding
... ...
... ...

My cable does not work.
Can someone fix my cable?
I can't watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn't work
I can't get my cable responding
... ...
... ...

I am not able to watch TV
I am not able to watch television
I am not able to watch cable
I am not able to watch cable TV
I cannot watch TV
I cannot watch television
I cannot watch cable
I cannot watch cable TV
Cannot watch TV
Cannot watch television
Cannot watch cable
Cannot watch cable TV
Can't watch TV
Can't watch television
Can't watch cable
Can't watch cable TV

This can also be normalized, e.g. regular expressions
So, let's be real...

lexical variants: synonyms, paraphrases

Intent
I have a problem with my television.

- My cable does not work.
- Can someone fix it?
- I can't watch TV.
- Has my cable service been down?
- I am not able to watch.
- Is there an outage?
- Cable TV is not loading.
- Cable is kaput.
- My TV doesn't work.
- I can't get my cable responding.

1-week project on a regular-expression-based filter.

From 89% to 95% F1 with only a 5% positive rate.

7-label domain classification task using a RandomForest classifier over TFIDF-weighted vectors.

This can also be normalized, e.g., regular expressions.
So, let's be real...

lexical variants: synonyms, paraphrases

Intent
I have a problem with my TV.

My cable does not work.
Can someone fix it?
I can't watch TV.
Has my cable service increased?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput.
My TV doesn't work.
I can't get my cable responding.

... ...

... ...

Entity
I am not able to watch TV
I am not able to watch television
I am not able to watch cable TV

Telefonica

Vs. proposal to handle e.g. negation
as a bi-gram/tri-gram feature engineering problem.

Preferably over a normalized feature space
(can't watch tv, cannot watch cable TV) > RB-neg watch TV

This can also be normalized, e.g. regular expressions
So, let's be real...

lexical variants:
synonyms, paraphrases

Inten
I have a problem with my...

My cable does not...
Can someone fix...
I can’t watch TV...
Has my cable service...
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn’t work
I can’t get my cable responding
... ...

Arguably, providers should be doing it.

All the frameworks currently available on the market force the user to do the work explicitly.

This can also be normalized, e.g. regular expressions
So, let's be real...

**lexical variants:**
synonyms, paraphrases

**Intent**
I have a problem with my cable.

- My cable does not work.
- Can someone fix my cable?
- I can’t watch TV.
- Has my cable service expired?
- I am not able to get a reception.
- Is there an outage?
- Cable TV is not loading.
- Cable is kaput
- My TV doesn’t work
- I can’t get my cable responding
  ... ...
  ... ...

- My internet connection does not work.
- My smartphone does not work.
- My landline does not work.
- My router does not work.
- My SIM does not work.
- My payment method does not work.
- My saved movies does not work.
So, let's be real...

lexical variants:
synonyms, paraphrases

Intent
I have a problem with my cable.

My cable does not work.  
Can someone fix my cable?  
I can’t watch TV.  
Has my cable service expired?  
I am not able to get a reception.  
Is there an outage?  
Cable TV is not loading.  
Cable is kaput  
My TV doesn’t work  
I can’t get my cable responding  
... ...

My internet connection does not work.  
My smartphone does not work.  
My landline does not work.  
My router does not work.  
My SIM does not work.  
My payment method does not work.  
My saved movies does not work.

ConnectionDebugWizard(*)  
TechnicalSupport(Mobile)  
TechnicalSupport(Landline)  
RouterDebugWizard()  
VerifyDeviceSettings  
AccountVerificationProcess  
TVDebugWizard
So, let's be real...

...synonyms, paraphrases

**Telefonica**

Since we associate each intent to an action, the ability to discriminate actions presupposes training as many separate intents, with just as many ambiguities.
So, let's be real...

**Intent**
I have a problem with my cable.

- My cable does not work.
- Can someone fix my cable?
- I can't watch TV.
- Has my cable service expired?
- I am not able to get a reception.
- Is there an outage?
- Cable TV is not loading.
- Cable is kaput
- My TV doesn't work
- I can't get my cable responding
  - ...
  - ...

**Telefónica**

- My internet connection does not work.
- My smartphone does not work.
- My landline does not work.
- My router does not work.
- My SIM does not work.
- My payment method does not work.
- My saved movies does not work.

- ConnectionDebugWizard(*)
- TechnicalSupport(Mobile)
- TechnicalSupport(Landline)
- RouterDebugWizard()
- VerifyDeviceSettings
- AccountVerificationProcess
- TVDebugWizard

It also presupposes exponentially increasing amounts of training data as we add higher-precision entities to our model (every entity * every intent that applies)
So, let's be real...

lexical variants: synonyms, paraphrases

Telefónica

My internet connection does not work.
My smartphone does not work.
My landline does not work.
My router does not work.
My SIM does not work.
My payment method does not work.
My saved movies does not work.

The entity resolution step (which would have helped us make the right decision) is nested downstream in our pipeline.

Any errors from the previous stage will propagate down.

Intent
I have a problem with my cable.

My cable does not work.
Can someone fix my cable?
I can’t watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn’t work
I can’t get my cable responding
... ...
... ...

Any errors from the previous stage will propagate down.
So, let's be real...

Welcome to Macy's blah-blah...!

Hi, I'm a guy looking for LEDA jackets.

We've got what you're looking for!

That's a nice one but, I want to see different jackets... how can I do that.

lexical variants: synonyms, paraphrases

formal variants (typos, ASR errors)

morphological variants

multiword chunking/parsing
So, let's be real...

Welcome to Macy's blah-blah...!

Hi, I'm a guy looking for LEDA jackets.

We've got what you're looking for!

Hey, I need to buy a men's red leather jacket with straps for my friend.

That's a nice one, but I want to see different jackets. How can I do that?
So, let's be real...

- buy <red jacket>
- where can i find a <jacket with straps>
- looking for a <black mens jacket>
- do you have anything in <leather>
- <present> for a friend

Our training dataset
So, let's be real...

**Our training dataset**

- buy <red jacket>
- where can i find a <jacket with straps>
- looking for a <black mens jacket>
- do you have anything in <leather>
- <present> for a friend

**Predictions at runtime**

- where can i buy <a men's red leather jacket with straps> for my friend
- hi i need <a men's red leather jacket with straps> that my friend can wear
So, let's be real...

Our training dataset

buy <red jacket>

where can i find a <jacket with straps>

looking for a <black mens jacket>

do you have anything in <leather>

<present> for a friend

where can i buy <a men's red leather jacket with straps> for

Incomplete training data will cause entity segmentation issues

hi i need <a men's red leather straps> that my friend can
So, let's be real...

But more importantly, in the intent-entity paradigm we are usually unable to train two entity types with the same distribution:

- where can I buy a <red leather jacket> (jacket, 0.5; wallet, 0.5)
- where can I buy a <red leather wallet> (jacket, 0.5; wallet, 0.5)
So, let's be real...

buy <red jacket>

where can i find a <jacket with straps>

looking for a <black mens jacket>

... and have no way of detecting entities used in isolation:

pair of shoes

["START", 3-gram, "END"]

GetWeather
Goodbye
MakePurchase
Greeting
ReturnPurchase

Our training dataset...
So, let's be real...

Welcome to Macy's blah-blah...!

hi im a guy looking for leda jackets

We've got what you're looking for!

searchItemByText("leather jacket")

searchItemByText("to buy a men's ... pfff TLDR... my friend")

[
  { "name": "leather jacket",
    "product_category_id": 12,
    "gender_id": 0,
    "materials_id": [68, 34, ...],
    "features_id": [],
  },
  ...
  { "name": "leather jacket with straps",
    "product_category_id": 12,
    "gender_id": 1,
    "materials_id": [68, 34, ...],
    "features_id": [8754],
  },
]
So, let's be real...

Welcome to Macy's blah-blah

hi im a guy looking for leda jackets

We've got what you're looking for!

searchItemByText("leather jacket")

searchItemByText("red leather jacket with straps")

[{
    "name": "leather jacket",
    "product_category_id": 12,
    "gender_id": 0,
    "materials_id": [68, 34, ...],
    "features_id": [],
},
...
{
    "name": "leather jacket with straps",
    "product_category_id": 12,
    "gender_id": 1,
    "materials_id": [68, 34, ...],
    "features_id": [8754],
},
]

NO IDs,
NO reasonable expectation of a cognitively realistic answer and
NO "conversational technology".
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Since the conditions express an AND logical operator and we will search for an item fulfilling all of them, we could actually search for the terms in the conjunction in any order.
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given
• any known word, denoted by $w$,
• the vocabulary $V$ of all known words, such that $w_i \in V$ for any $i$,
• search space $D$, where $d$ denotes every document in a collection $D$ such that $d \in D$ and $d = \{w_i, w_{i+1}, ..., w_n\}$, such that $w_{i \leq x \leq n} \in V$,
• a function $\text{filter}(D, w_x)$ that returns a subset of $D$, $Dw_x$ where $w_x \in d$ is true for every $d: d \in Dw_x$, and
• query $q = \{"men's", 'red', 'leather', 'jacket', 'straps'}$,
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

do filter(filter(filter(
    filter(D, 'leather'), 'straps'), "men's"), 'jacket'), 'red'
) =

do filter(filter(filter(
    filter(D, 'straps'), 'red'), 'jacket'), 'leather'), "men's"
) =

do filter(filter(filter(
    filter(D, "men's"), 'leather'), 'red'), 'straps'), 'jacket'
) =
So, let's be real...

Given query $q = \{"men's", 'red', 'leather', 'jacket', 'straps'\}$, find the $x: x \in q$ that satisfies:

$$containsSubString(x, 'leather') \land containsSubString(x, 'straps') \land \ldots \land containsSubString(x, "men's")$$

We cannot use more complex predicates because, in this scenario, the system has no understanding of the internal structure of the phrase, its semantic dependencies, or its syntactic dependencies.

As a result, it will lack the notion of what counts as a partial match.
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

The following partial matches will be virtually indistinguishable:

- leather straps for men's watches (3/5)
- men's vintage red leather shoes (3/5)
- red wallet with leather straps (3/5)
- red leather jackets with hood (3/5)
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given
• any known word, denoted by $w$,
• the vocabulary $V$ of all known words,
• search space $D$, where $d$ denotes every document in a collection $D$ such that $d \in D$ and $d = \{w_i, w_{i+1}, ..., w_n\}$, where $w_i \leq x \leq n \in V$,
• a function $\text{filter}(D, w_x)$ that returns a subset of $D$, $D_{w_x}$, where $w_x \in d$ is true for every $d$: $d \in D_{w_x}$ and
• query $q = \{"men's", 'red', 'leather', 'jacket', 'straps'\}$

This is our problem:
we're using a function that treats equally every $w_x$.

do $\text{filter}(\text{filter}(\text{filter}(\text{filter}(\text{filter}(D, "men's"), 'red'), 'leather'), 'jacket'), 'straps')$
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given
- any known word, denoted by $w_i$
- the vocabulary $V$ of all known words
- search space $D$, where $d$ denotes every document in the collection $D$, such that $d \in D$ and $d = \{w_i, w_{i+1}, \ldots, w_n\}$
- a function $\text{filter}(D, w_x)$ that returns a subset of $D$ where $w_x \in d$ is true for every $d: d \in D_{w_x}$ and
- query $q = \{"men's", 'red', 'leather', 'jacket', 'straps'\}$

We need a function with at least one additional parameter, $r_x$

$\text{filter}(D, w_x, r_x)$

where

$r_x$ denotes the depth at which the dependency tree headed by $w_x$ attaches to the node being currently evaluated.
So, let's be real...

http://nlp.stanford.edu:8080/parser/index.jsp (as of September 19th 2017)

Your query
i need to buy a men's red leather jacket with straps for my friend

Tagging
i/FW need/VBP to/TO buy/VB a/DT men/NNS 's/POS red/JJ leather/NN jacket/NN with/IN straps/
NNS for/IN my/PRP$ friend/NN

Parse
(ROOT
 (S
  (NP (FW i))
  (VP (VBP need)
   (S
    (VP (TO to)
     (VP (VB buy)
      (NP
       (NP (DT a) (NNS men) (POS 's))
       (JJ red) (NN leather) (NN jacket))
      (PP (IN with)
       (NP
        (NP (NNS straps))
        (PP (IN for)
         (NP (PRP$ my) (NN friend))))))))))}
So, let's be real...

http://nlp.stanford.edu:8080/parser/index.jsp (as of September 19th 2017)

Your query
i need to buy a men's red leather jacket with straps for my friend

Tagging
i/FW need/VBP to/TO buy/VB a/DT men/NNS 's/POS red/JJ leather/NN jacket/NN with/IN straps/
NNS for/IN my/PRP$ friend/NN

Parse
(ROOT
  (S
    (NP (FW i))
    (VP (VBP need)
      (S
        (VP (TO to)
          (VP (VB buy)
            (NP
              (NP (DT a) ((NNS men) (POS 's))
                ((JJ red) (NN leather)) (NN jacket))
              (PP (IN with)
                (NP
                  (NP (NNS straps))
                  (PP (IN for)
                    (NP (PRP$ my) (NN friend))))))))))))
So, let's be real...

http://nlp.stanford.edu:8080/parser/index.jsp (as of September 19th 2017)

Your query
i need to buy a men's red leather jacket with straps for my friend

Tagging
i/FW need/VBP to/TO buy/VB a/DT men/NNS 's/POS red/JJ leather/NN jacket/NN with/IN straps/
NNS for/IN my/PRP$ friend/NN

Parse
(ROOT
  (S
    (NP (FW i))
    (VP (VBP need)
      (S
        (VP (TO to)
          (VP (VB buy)
            (NP
              (NP (DT a) ((NNS men) (POS 's))
                ((JJ red) (NN leather)) (NN jacket)))
              (PP (IN with)
                (NP
                  (NP (NNS straps))
                  (PP (IN for)
                    (NP (PRP$ my) (NN friend))))))))))

(red, leather)
(leather, jacket)
(jacket, buy)
So, let's be real...

Hey I need to buy a men's red leather jacket with straps for my friend.

Given
- any known word, denoted by $w_i$
- the vocabulary $V$ of all known words
- search space $D$, where $d$ denotes every document in a collection $D$ such that $d \in D$ and $d = \{w_i, w_i+1, ..., w_n\}$
- a function $\text{filter}(D, w_x)$ that returns a subset of $D$, $D_{w_x}$, where $w_x \in d$ is true for every $d$: $d \in D_{w_x}$
- query $q = \{"men's", 'red', 'leather', 'jacket', 'straps'\}$

The output of this function will no longer be a subset of search results but a ranked list.

This list can be naively ranked by

$$\sum_{x=1}^{n} (1 / r_x)$$

although much more advanced scoring functions can be used.
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:
q: "red leather jacket"
a₁: "black leather jacket"
a₂: "red leather wallet"
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:
q: "red leather jacket"
a₁: "black leather jacket"
a₂: "red leather wallet"

>>> t1 = dependencyTree(a₁)
>>> t1 depth("black")
3
>>> t1 depth("jacket")
1
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:
q: "red leather jacket"
a1: "black leather jacket"
a2: "red leather wallet"

```python
>>> results = filter(D, "jacket", t1.depth("jacket"))
>>> results
[(Jacket1, 1.0),
 (Jacket2, 1.0),
 ..., 
 (Jacketn, 1.0 ) ]
```
So, let's be real...

Hey i need to buy a men's red leather jacket with straps for my friend

Given:
q: "red leather jacket"
a₁: "black leather jacket"
a₂: "red leather wallet"

```python
>>> results = filter(results, "leather", t1.depth("leather"))
```
```python
>>> results
[(Jacket₁, 1.5),
 (Jacket₂, 1.5),
 ...,  
 (Jacketₙ₋ₓ, 1.5 )
]
```
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:
q: "red leather jacket"
a₁: "black leather jacket"
a₂: "red leather wallet"

>>> results = filter(results, "red", t1.depth("red"))
>>> results
[ ( Jacket₁, 1.5 ),
  ( Jacket₂, 1.5 ),
  ...,
  ( Jacketₙ₋ₓ, 1.5 ) ]
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:
q: "red leather jacket"
a1: "black leather jacket"
a2: "red leather wallet"

```python
>>> t2 = dependencyTree(a2)
>>> t2.depth("red")
3
>>> t2.depth("jacket")
None
```
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:
q: "red leather jacket"
a1: "black leather jacket"
a2: "red leather wallet"

```python
>>> results = filter(D, "jacket", t2.depth("jacket"))
>>> results
[]
```
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:
q: "red leather jacket"
a₁: "black leather jacket"
a₂: "red leather wallet"

```python
>>> results = filter(results, "leather", t2.depth("leather"))
>>> results
[ ( Wallet1, 0.5 ),
  ( Wallet2, 0.5 ),
  ...
  ( Strap_n, 0.5 )
]```
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:
q: "red leather jacket"
a₁: "black leather jacket"
a₂: "red leather wallet"

>>> results = filter(results, "red", t2.depth("red"))
>>> results
[ ( Wallet₁, 0.83 ),
  ( Wallet₂, 0.83 ),
  ...
  ( Strapₙ - x, 0.83 )
]
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Results for $q_1$:
"black leather jacket"

[ ( Jacket_1, 1.5 ),
  ( Jacket_2, 1.5 ),
  ...
  ( Jacket_{n-x}, 1.5 )
]

Results for $q_2$:
"red leather wallet"

[ ( Wallet_1, 0.83 ),
  ( Wallet_2, 0.83 ),
  ...
  ( Strap_{n-x}, 0.83 )
]
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

However, the dependency tree is not always available, and it often contains parsing issues (cf. Stanford parser output).

An evaluation of parser robustness over noisy input reports performances down to an average of 80% across datasets and parsers.

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

👍 [ red leather jacket ]
So, let's be real...

Hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

👍 [ [red leather] jacket ]
👍 [ red [leather jacket] ]
So, let's be real...

Hey I need to buy a men's red leather jacket with straps for my friend.

In order to make this decision:

- [red leather] jacket
- red [leather jacket]
- 👍 men's [leather jacket]
So, let's be real...

Hey I need to buy a men's red leather jacket with straps for my friend!

In order to make this decision

- [red leather] jacket
- red [leather jacket]
- men's [leather jacket]
- men's leather jacket
- men's [leather jacket]
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

[ [red leather] jacket ]
[ red [leather jacket] ]
[ men's [leather jacket] ]

😭 [ [men's leather] jacket ]

... the parser already needs as much information as we can provide regarding head-modifier attachments.
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

[ [red leather] jacket ]  2 2 1
[ red [leather jacket] ]  2 1 1
[ men's [leather jacket] ]  2 1 1
[ [men's leather] jacket ]  2 2 1
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

[ [red leather] jacket ]  2 2 1  
[ red [leather jacket] ]  2 1 1  
[ men's [leather jacket] ]  2 1 1  
[ [men's leather] jacket ]  2 2 1  

Same pattern, different score. How to assign a probability such that less plausible results are buried at the bottom of the hypotheses space? (not only for display purposes, but for dynamic programming reasons)
So, let's be real...

Hey I need to buy a men's red leather jacket with straps for my friend.

In order to make this decision:

<table>
<thead>
<tr>
<th></th>
<th>Probability</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>[red leather] jacket</td>
<td>2 2 1</td>
<td>p(red</td>
</tr>
<tr>
<td>red [leather jacket]</td>
<td>2 1 1</td>
<td></td>
</tr>
<tr>
<td>men's [leather jacket]</td>
<td>2 1 1</td>
<td>p(men's</td>
</tr>
<tr>
<td>[men's leather] jacket</td>
<td>2 2 1</td>
<td></td>
</tr>
</tbody>
</table>
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

That's a lot of data to model. Where will we get it from?

- [red leather] jacket [2 2 1]
- red [leather jacket] [2 1 1]
- men's [leather jacket] [2 1 1]
- [men's leather] jacket [2 2 1]
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

That's a lot of data to model. Where will we get it from?

\[
p (\text{Color} | \text{ClothingItem}) \sim p (\text{Color} | \text{Material})
\]

\[
p (\text{Person} | \text{ClothingItem}) > p (\text{Person} | \text{Material})
\]
So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

If our taxonomy is well built, semantic associations acquired for a significant number of members of each class will propagate to new, unseen members of that class.

With a sufficiently rich NER detection pipeline and a semantic relation database large enough, we can rely on simple inference for a vast majority of the disambiguations.

More importantly, we can weight candidate hypotheses dynamically in order to make semantic parsing tractable over many possible syntactic trees.
So, let's be real...

Welcome to Macy's blah-blah...

Hi, I'm a guy looking for LEDA jackets.

We've got what you're looking for!

That's a nice one but, I want to see different jackets. How can I do that?

Lexical variants: synonyms, paraphrases

Formal variants (typos, ASR errors)

Morphological variants

Multiword

Entity recognition

Entity linking

Hierarchical structure
So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that
So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

A single utterance

QualitativeEvaluation(User_u, ClothingItem_i)

BrowseCatalogue(User_u, filter=[ClothingItem_n, j])

HowTo(User_u, Command_{Context})
So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

\[ I = [i_1, i_2, i_3] \]
\[ L = [|i_1|, |i_2|, |i_3|] \]
\[ P = [p(i_1), p(i_2), p(i_3)] \]

random.choice(I)

\[ x \text{ if } x \in I \wedge \]
\[ \text{length}(x) = \text{argmax}(L) \]

\[ x \text{ if } x \in I \wedge \]
\[ p(x) = \text{argmax}(P) \]

QualitativeEvaluation(User_u, ClothingItem_i)

BrowseCatalogue(User_u, filter=[ClothingItem_n, |J|])

HowTo(User_u, Command_c[Context])
So, let's be real...

```python
I = [i_1, i_2, i_3]
L = [|i_1|, |i_2|, |i_3|]
P = [p(i_1 | q), p(i_2 | q), p(i_3 | q)]

random.choice(I)

x if x ∈ I ∧
   length(x) = argmax(L)

x if x ∈ I ∧
   p(x | q) = argmax(P)
```

However, this approach will still suffer from segmentation issues due to passing many more arguments than expected for resolving any specific intent.

that's a nice one but, i want to see different jackets .... how can i do that
So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

A standard API will normally provide these definitions as the declaration of its methods.

```
SetPreference(User u, SetRating(ClothingItem i, k, 1))

\forall(ClothingItem n, m | n = i, k = *)

GetInfo(User u, argmax(Command c(Context)))
```
So, let's be real...

We can map the argument signatures to the nodes of the taxonomy we are using for linguistic inference.

that's a nice one but, i want to see different jackets .... how can i do that
So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

The system will then be able to link API calls to entity parses dynamically.

- SetPreference(\textit{User}_u, \textit{SetRating(ClothingItem}_i, k, 1) )
- \( \forall(ClothingItem_{n,m} \mid n = i, k = *) \)
- GetInfo(\textit{User}_u, \text{argmax(Command}_c^{\text{Context}}) )
Over time, new methods added to the API will be automatically supported by the pre-existing linguistic engine.

So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

SetPreference(User_u, SetRating(ClothingItem_i, k, 1))

∀(ClothingItem_{n, m} | n = i, k = *)

GetInfo(User_u, argmax(Command_c^{Context}))
So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

Over time, new methods added to the API will be automatically supported by the pre-existing linguistic engine.
So, let's be real...

If the system can reasonably solve nested semantic dependencies hierarchically,

it can also be extended to handle multi-intent utterances in a natural way.
So, let's be real...

1. **Convert to abstract entity types**
   - i need to pay my bill and i cannot log into my account

2. **Activate applicable dependencies/grammar rules**
   - Resolve attachments and rank candidates by semantic association, keep top-\(n\) hypotheses
   - (fewer and less ambiguous thanks to the previous step)

3. **Resolve attachments and rank candidates by semantic association, keep top-\(n\) hypotheses**
   - i need to \[Payment(x)\] and \[Issue(y)\]
So, let's be real...

AND usually joins ontologically coordinate elements, e.g. *books and magazines versus humans and Donald Trump*
So, let's be real...

Even if

\[ p_0 = p(\text{Payment} | \text{Issue}) > u \]

we will still split the intents because here we have

\[ p_y = p(\text{Payment}, x | \text{Issue}, y) \]

and normally \( p_y < p_0 \) for any irrelevant case (i.e., the posterior probability of the candidate attachment will not beat the baseline of its own prior probability)
So, let's be real...

I.e., *Payment* may have been attached to *Issue* if no other element had been attached to it before.

*Resolution order*, as given by attachment probabilities, matters.
So, let's be real...

I need to [Payment(x)] and i [Issue(y)]

I need to [Payment(x)] and i [Issue(y)]

Multi-intents solved 😎
So, let's be real...

no! forget the last thing i said

go back!

third option in the menu

i want to modify the address I gave you
So, let's be real...

We need:

• semantic association measures between the leaves of the tree in order to attach them to the right nodes and be able to stop growing a tree at the right level,

• a syntactic tree to build the dependencies, either using a parser or a PCFG/LFG grammar (both may take input from the previous step, which will provide the notion of verbal valence: Person Browse Product),

• for tied analyses, an interface with the user to request clarification.
Recent advances
Supersense inference

GFT (Google Fine-Grained) taxonomy

Supersense inference

FIGER taxonomy

Supersense inference

FB = Freebase
T = OurTaxonomy\(^1\)
owem = OurWordEmbeddingsModel

for every document \(d\) in \(D\)^2:
  for every sent \(s\) in \(d\):
    sentence_vector = None
    for every \(position, \text{term}, \text{tag}\) tuple in TermExtractor^3(s):
      if not sentence_vector:
        sentence_vector = vectorizeSentence(s, position)
      owem[ T[tag] ].update( sentence_vector )

---

\(^1\) FB labels are manually mapped to fine-grained labels.
\(^2\) \(D = 133,000\) news documents.
\(^3\) TermExtractor = parser + chunker + entity resolver that assigns Freebase types to entities.

Supersense inference

FB = Freebase
T = OurTaxonomy
owem = OurWordEmbeddingsModel

**Table: Features, Description, and Example**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
<td>The syntactic head of the mention phrase</td>
<td>&quot;Obama&quot;</td>
</tr>
<tr>
<td>Non-head</td>
<td>Each non-head word in the mention phrase</td>
<td>&quot;Barack&quot;, &quot;H.&quot;</td>
</tr>
<tr>
<td>Cluster</td>
<td>Word cluster id for the head word</td>
<td>&quot;s9&quot;</td>
</tr>
<tr>
<td>Characters</td>
<td>Each character trigram in the mention head</td>
<td>&quot;-ob&quot;, &quot;oba&quot;, &quot;bam&quot;, &quot;ama&quot;, &quot;ma:&quot;</td>
</tr>
<tr>
<td>Shape</td>
<td>The word shape of the words in the mention phrase</td>
<td>&quot;A a A a&quot;</td>
</tr>
<tr>
<td>Role</td>
<td>Dependency label on the mention head</td>
<td>&quot;suby&quot;</td>
</tr>
<tr>
<td>Context</td>
<td>Words before and after the mention phrase</td>
<td>&quot;B:who&quot;, &quot;A:flow&quot;</td>
</tr>
<tr>
<td>Parent</td>
<td>The head's lexical parent in the dependency tree</td>
<td>&quot;picked&quot;</td>
</tr>
<tr>
<td>Topic</td>
<td>The most likely topic label for the document</td>
<td>&quot;politics&quot;</td>
</tr>
</tbody>
</table>

sentence_vector = vectorizeSentence(s, position)

owem[ T[tag] ].update( sentence_vector )

---

1 FB labels are manually mapped to fine-grained labels.
2 D = 133,000 news documents.
3 TermExtractor = parser + chunker + entity resolver that assigns Freebase types to entities.

### Supersense inference

<table>
<thead>
<tr>
<th>Method</th>
<th>P</th>
<th>R</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLAT</td>
<td>79.22</td>
<td>60.18</td>
<td>68.40</td>
</tr>
<tr>
<td>BINARY</td>
<td>80.05</td>
<td>62.20</td>
<td>70.01</td>
</tr>
<tr>
<td>WSABIE</td>
<td><strong>80.58</strong></td>
<td>66.20</td>
<td>72.68</td>
</tr>
<tr>
<td>K-WSABIE</td>
<td>80.11</td>
<td><strong>67.01</strong></td>
<td>72.98</td>
</tr>
</tbody>
</table>

Fact-driven generation

$W = \text{Topic description, raw text about the topic, e.g. } "\text{Rogers was born in Latrobe, Pennsylvania in 1928...}" \text{ (such as Wikipedia)}$

$F = \text{Facts associated with the topic (as triples from FreeBase), such as:}$

$\ a42 = (\text{Fred}_\text{Rogers}, \text{Place}_\text{of}_\text{Birth}, \text{Latrobe}_\text{Pennsylvania})$
$\ a83 = (\text{Fred}_\text{Rogers}, \text{Year}_\text{of}_\text{Birth}, 1928)$
$\ a0 = (\text{Fred}_\text{Rogers}, \text{Topic}_\text{Itself}, \text{Fred}_\text{Rogers})$

The corpus contains pairs $(W_k, F_k)_k^K$
Fact-driven generation

Two sources of output:

Knowledge words $O_a$ of a fact $a$ is all words $(o_{a1}, o_{a2}, ..., o_{aN})$.
   If selected, these words are copied to the output.

Global vocabulary $V$ containing frequent words.
   Words describing relationships (e.g., “married to”) are common and thus can be generated via the vocabulary $V$ not via copy.

Fact-driven generation

During training:

Simple string matching to map words in $W$ to facts in $F$.
Since not all words are associated with a fact (e.g., words like, is, a, the, have), the *Not-a-Fact (NaF)* type is used to generate those words.

... Fred_Rogers was born in **Pennsylvania**...

$t-4$ $t-3$ $t-2$ $t-1$ $t$

\[
p(O \mid born, a, Place_of_Birth, o_{ax}) > p(V \mid born, a, Place_of_Birth, o_{ax})
\]

(in the embeddings space)

Fact-driven generation

During training:

Simple string matching to map words in W to facts in F. Since not all words are associated with a fact (e.g., words like, is, a, the, have), the Not-a-Fact (NaF) type is used to generate those words.

... Fred_Rogers was born in Pennsylvania...

\[
p(O \mid \text{born, a, Place_of_Birth, o}_{ax}) > \quad p(V \mid \text{born, a, Place_of_Birth, o}_{ax})
\]

Fact-driven generation

During training:

Simple string matching to map words in $W$ to facts in $F$.
Since not all words are associated with a fact (e.g., words like, is, a, the, have), the Not-a-Fact (NaF) type is used to generate those words.

... **Fred_Rogers** was born in **Pennsylvania**...

\[
p(O \mid \text{born}, a, \text{Place_of_Birth}, o_{ax}) > p(V \mid \text{born}, a, \text{Place_of_Birth}, o_{ax})
\]

Currently fact activated

Fact-driven generation

During training:

Simple string matching to map words in W to facts in F.
Since not all words are associated with a fact (e.g., words like, is, a, the, have), the Not-a-Fact (NaF) type is used to generate those words.

... Fred_Rogers was born in Pennsylvania...

\[ p(O \mid \text{born, a, Place_of_Birth, o}_{ax}) > p(V \mid \text{born, a, Place_of_Birth, o}_{ax}) \text{ (in the embeddings space)} \]

Fact word belong to this topic (as positive example)

Fact-driven generation

During inference:

\[
P(a|h_t) = \frac{\exp(k_{\text{fact}}^T F[a])}{\sum_{a' \in \mathcal{F}} \exp(k_{\text{fact}}^T F[a'])},
\]

\[a_t = \arg\max_{a \in \mathcal{F}} P(a|h_t),\]

\[a_t = F[a_t].\]

Output the fact word associated with the relation that maximizes the similarity with the state at \(t\)

Semantic trees

Google DeepMind's Neural Turing Machines...

... applied to a Question and Answer task.


\[ q = \text{"Which city hosted the longest game before the game in Beijing?"} \]
Semantic trees

\[ q = "Which city hosted the longest game before the game in Beijing?" \]

Semantic trees

q = "Which city hosted the longest game before the game in Beijing?"

Which city hosted the longest game before the game in Beijing?

argmax of the column whose title name's embedding matches embedding for input word 'longest'

Semantic trees

$q = "Which city hosted the longest game before the game in Beijing?"

Which city hosted| the longest game| before| the game in Beijing?

less than over the column whose title name's embedding matches embedding for input word 'before'

```
| less than(Year)          |
| argmax(Duration)         |
| select column           |
```

Semantic trees

$q = "Which city hosted the longest game before the game in Beijing?"

Which city hosted the longest game before the game in Beijing?

over all the rows, row containing the value whose embedding matches the embedding for input word 'Beijing'

| select where |
| less than(Year) |
| argmax(Duration) |
| select column |

Semantic trees

q = "Which city hosted the longest game before the game in Beijing?"
KB = knowledge base consisting of tables that store facts
encoder = bi-directional Recurrent Neural Network
v = encoder(q)     # vector of the encoded query
kb_encoder = KB table encoder

Semantic trees

\[ q = "\text{Which city hosted the longest game before the game in Beijing?}\" \]

KB = knowledge base consisting of tables that store facts
encoder = bi-directional Recurrent Neural Network
\[ v = \text{encoder}(q) \quad \# \text{ vector of the encoded query} \]

kb_encoder = KB table encoder

\[ e_{mn} = \tanh(W \cdot [L[w_{mn}]; f_n] + b) \]

\[ f_n = \text{embedding of the name of the } n\text{-th column} \]
\[ w_{mn} = \text{embedding of the value in the } n\text{-th column of the } m\text{-th row} \]
\[ [x ; y] = \text{vector concatenation} \]
\[ W = \text{weights (what we will e.g. SGD out of the data)} \]
\[ b = \text{bias} \]

Semantic trees

\[
q = "\text{Which city hosted the longest game before the game in Beijing?}\"
\]

\[
\text{KB = knowledge base consisting of tables that store facts}
\]

\[
\text{encoder = bi-directional Recurrent Neural Network}
\]

\[
v = \text{encoder}(q) \quad \# \text{ vector of the encoded query}
\]

\[
\text{kb\_encoder = KB table encoder}
\]

\[
\begin{align*}
e_{mn} = \tanh(W \cdot [L[w_{mn}]; f_n] + b)
\end{align*}
\]

\[
f_n = \text{embedding of the name of the } n\text{-th column}
\]

\[
w_{mn} = \text{embedding of the value in the } n\text{-th column of the } m\text{-th row}
\]

\[
[x ; y] = \text{vector concatenation}
\]

\[
W = \text{weights (what we will e.g. SGD out of the data)}
\]

\[
b = \text{bias}
\]

\[
\text{normalizeBetween0and1(}
\]

\[
\text{# the weights that minimize the loss for this column}
\]

\[
[ \{ \text{sydney... australia... kangaroo...} \} + \{ \text{city... town... place...} \} ] +
\]

\[
\text{somePriorExpectations}
\]

\[
)\]

Semantic trees

$q = "Which city hosted the longest game before the game in Beijing?"
KB = knowledge base consisting of tables that store facts
codercer = bi-directional Recurrent Neural Network
$v = encoder(q)$ # vector of the encoded query
$kb_{encoder} = KB$ table encoder

$$e_{mn} = \tanh(W \cdot [L[w_{mn}], f_n] + b)$$

$f_n$ = embedding of the name of the $n$-th column
$w_{mn}$ = embedding of the value in the $n$-th column of the $m$-th row
$[x ; y] = \text{vector concatenation}$
$W = \text{weights (what we will e.g. SGD out of the data)}$
$b = \text{bias}$

\begin{align*}
\text{normalizeBetween0and1(} & \text{penaltiesOrRewards} * \\
\{ \text{sydney... australia... kangaroo...} + \{\text{city... town... place...} \} \} + \\
\text{somePriorExpectations} & \text{)} \end{align*}

Semantic trees

\[ q = "\text{Which city hosted the longest game before the game in Beijing?}\" \]

KB = knowledge base consisting of tables that store facts
encoder = bi-directional Recurrent Neural Network
\[ v = \text{encoder}(q) \quad \# \text{ vector of the encoded query} \]
kb_encoder = KB table encoder

\[
e_{mn} = \tanh(W \cdot [L[w_{mn}]; f_n] + b)
\]

\( f_n \) = embedding of the name of the \( n \)-th column
\( w_{mn} \) = embedding of the value in the \( n \)-th column of the \( m \)-th row
\[ [x ; y] = \text{vector concatenation} \]
\( W \) = weights (what we will e.g. SGD out of the data)
\( b \) = bias

\[
\text{normalizeBetween0and1}(\text{penaltiesOrRewards} \ast \left[ \{\text{sydney... australia... kangaroo...}\} + \{\text{city... town... place...}\} \right] + \text{somePriorExpectations})
\]


Risk of false positives?

"olympic games near kangaroos" 👍

Already as probabilities!!
Which city hosted the 2012 Olympic Games?

<table>
<thead>
<tr>
<th>City</th>
<th>Year</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barcelona</td>
<td>1992</td>
<td>15</td>
</tr>
<tr>
<td>Atlanta</td>
<td>1996</td>
<td>18</td>
</tr>
<tr>
<td>Sydney</td>
<td>2000</td>
<td>17</td>
</tr>
<tr>
<td>Athens</td>
<td>2004</td>
<td>14</td>
</tr>
<tr>
<td>Beijing</td>
<td>2008</td>
<td>16</td>
</tr>
<tr>
<td>London</td>
<td>2012</td>
<td>16</td>
</tr>
<tr>
<td>Rio de Janeiro</td>
<td>2016</td>
<td>18</td>
</tr>
</tbody>
</table>

Semantic trees

Olympic Games

<table>
<thead>
<tr>
<th>City</th>
<th>Year</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barcelona</td>
<td>1992</td>
<td>15</td>
</tr>
<tr>
<td>Atlanta</td>
<td>1996</td>
<td>18</td>
</tr>
<tr>
<td>Sydney</td>
<td>2000</td>
<td>17</td>
</tr>
<tr>
<td>Athens</td>
<td>2004</td>
<td>14</td>
</tr>
<tr>
<td>Beijing</td>
<td>2008</td>
<td>16</td>
</tr>
<tr>
<td>London</td>
<td>2012</td>
<td>16</td>
</tr>
<tr>
<td>Rio de Janeiro</td>
<td>2016</td>
<td>18</td>
</tr>
</tbody>
</table>

\[
T^{(i)} = \quad \text{Which city hosted the 2012 Olympic Games?}
\]

\[
y^{(i)} = \quad \text{Which city hosted the 2012 Olympic Games?}
\]

\[
Q^{(i)} = \quad \text{Which city hosted the 2012 Olympic Games?}
\]

\[
RNN-encoded = \quad p(\text{vector("column:city") | [ vector("which"), vector("city")... ]}) > p(\text{vector("column:city") | [ vector("city"), vector("which")... ]})
\]

Semantic trees

$Q^{(l)} = \text{Which city hosted the 2012 Olympic Games?}$

$RNN-encoded = \begin{align*} & p(\text{vector("column:city") | [ vec("which"), vec("city")... ]}) > \hfill p(\text{vector("column:city") | [ vec("city"), vec("which")... ]}) \hfill \text{Executor layer L} \\
& \quad p(\text{vector("column:year") | [...] vec("2012"), vec("olympic")... ]}) > \hfill p(\text{vector("column:year") | [...] vec("olympic"), vec("2012")... ]}) \hfill \text{Executor layer L - 1} \\
& \quad p(\text{vector("FROM") | [...] vec("olympic"), vec("games")... ]}) > \hfill p(\text{vector("FROM") | [...] vec("games"), vec("olympic")... ]}) \hfill \text{Executor layer L - 2} \quad \end{align*}$

As many as SQL operations

Semantic trees

\[ Q^{(i)} = \text{Which city hosted the 2012 Olympic Games?} \]

\[ RNN-encoded = \]

\[ \text{Executor layer } L \]

\[ p(\text{vector("column:city") } | \{ \text{vec("which"), vec("city")... } \}) > p(\text{vector("column:city") } | \{ \text{vec("city"), vec("which")... } \}) \]

\[ \text{Executor layer } L - 1 \]

\[ p(\text{vector("column:year") } | \{ \text{... vec("2012"), vec("olympic")... } \}) > p(\text{vector("column:year") } | \{ \text{... vec("olympic"), vec("2012")... } \}) \]

\[ \text{Executor layer } L - 2 \]

\[ p(\text{vector("FROM") } | \{ \text{... vec("olympic"), vec("games")... } \}) > p(\text{vector("FROM") } | \{ \text{... vec("games"), vec("olympic")... } \}) \]

Each executor applies the same set of weights to all rows (a single operation)

Semantic trees

\[ Q^{(i)} = \text{Which city hosted the 2012 Olympic Games?} \]

\[ RNN-encoded = p(\text{vector("column:city") | [ vec("which"), vec("city")... ]}) \succ \]

**Executor layer \( L \)**

\[ p(\text{vector("column:city") | [ vec("city"), vec("which")... ]}) \]

\[ p(\text{vector("column:year") | [ ... vec("2012"), vec("olympic")... ]}) \succ \]

**Executor layer \( L - 1 \)**

\[ p(\text{vector("column:year") | [ ... vec("olympic"), vec("2012")... ]}) \]

\[ p(\text{vector("FROM") | [ ... vec("olympic"), vec("games")... ]}) \succ \]

**Executor layer \( L - 2 \)**

\[ p(\text{vector("FROM") | [ ... vec("games"), vec("olympic")... ]}) \]

*Before, After* are just temporal versions of < and >. Neural Turing Machines learn to sort accordingly.

Semantic trees

\[
Q^{(i)} = \text{Which city hosted the 2012 Olympic Games?}
\]

\[
\text{RNN-encoded} = \quad p(\text{vector("column:city")} | \text{[ vec("which"), vec("city")... ]}) > \quad p(\text{vector("column:city")} | \text{[ vec("city"), vec("which")... ]})
\]

\[
\text{Executor layer } L \quad \quad p(\text{vector("column:year")} | \text{[ ... vec("2012"), vec("olympic") ... ]}) > \quad p(\text{vector("column:year")} | \text{[ ... vec("olympic"), vec("2012") ... ]})
\]

\[
\text{Executor layer } L - 1 \quad \quad p(\text{vector("FROM")} | \text{[ ... vec("olympic"), vec("games") ... ]}) > \quad p(\text{vector("FROM")} | \text{[ ... vec("games"), vec("olympic") ... ]})
\]

Candidate vector at layer \( L \) for row \( m \) of our database

\[
\begin{align*}
\mathbf{r}_m^L &= f_k^L(\mathbf{R}_m, \mathcal{F}_T, \mathbf{q}, \mathcal{M}^{L-1}) \\
&= \sum_{n=1}^{N} \tilde{w}(f_n, q, g^{L-1})e_{mn}
\end{align*}
\]

Memory layer with the last output vector generated by executor \( L - 1 \)

Semantic trees

\[ Q(i) = \text{Which city hosted the 2012 Olympic Games?} \]

\[ \text{RNN-encoded} = p(\text{vector("column:city") | [ vec("which"), vec("city")... ]}) > p(\text{vector("column:city") | [ vec("city"), vec("which")... ]}) \]

Executor layer \( L \)

\[ p(\text{vector("column:year") | [ ... vec("2012"), vec("olympic") ... ]}) > p(\text{vector("column:year") | [ ... vec("olympic"), vec("2012") ... ]}) \]

Executor layer \( L - 1 \)

\[ p(\text{vector("FROM") | [ ... vec("olympic"), vec("games") ... ]}) > p(\text{vector("FROM") | [ ... vec("games"), vec("olympic") ... ]}) \]

As an attention mechanism, cumulatively weight the vector by the output weights from executor \( L - 1 \)

Evaluate a candidate:

\[ r_m^\ell = f_{R_m}^\ell(R_m, F_T, I, \mathcal{M}^{\ell-1}) = \sum_{n=1}^{N} \tilde{\omega}(f_n, q, g^{\ell-1}) e_{mn} \]

Row \( m \), \( R = \text{our table} \)

Set of columns

Semantic trees

\[ Q^{(i)} = \text{Which city hosted the 2012 Olympic Games?} \]

**RNN-encoded**

\[ p(\text{vector("column:city") | [ vec("which"), vec("city")... ]}) > p(\text{vector("column:city") | [ vec("city"), vec("which")... ]}) \]

**Executor layer \( L \)**

\[ p(\text{vector("column:year") | [ ... vec("2012"), vec("olympic") ... ]}) > p(\text{vector("column:year") | [ ... vec("olympic"), vec("2012") ... ]}) \]

**Executor layer \( L - 1 \)**

\[ p(\text{vector("FROM") | [ ... vec("olympic"), vec("games") ... ]}) > p(\text{vector("FROM") | [ ... vec("games"), vec("olympic") ... ]}) \]

**Executor layer \( L - 2 \)**

This allows the system to restrict the reference at each step.

**Evaluate a candidate:**

\[ r_m^\ell = f_{\alpha}^\ell(R_m, F_T, I, \mathcal{M}^{\ell-1}) = \sum_{n=1}^{N} \omega(f_n, q, g^{\ell-1}) e_{mn} \]

Set of columns

Semantic trees

\[ Q(t) = \text{Which city hosted the 2012 Olympic Games?} \]

\[ RNN-encoded = \quad p(\text{vector("column:city") } | \text{ [ vec("which"), vec("city")... ]}) > \]

\[ \text{Executor layer } L = \quad p(\text{vector("column:city") } | \text{ [ vec("city"), vec("which")... ]}) \]

\[ \text{Executor layer } L - 1 = \quad p(\text{vector("column:year") } | \text{ [ ... vec("2012"), vec("olympic") ... ]}) > \]

\[ p(\text{vector("column:year") } | \text{ [ ... vec("olympic"), vec("2012") ... ]}) \]

\[ \text{Executor layer } L - 2 = \quad p(\text{vector("FROM") } | \text{ [ ... vec("olympic"), vec("games") ... ]}) > \]

\[ p(\text{vector("FROM") } | \text{ [ ... vec("games"), vec("olympic") ... ]}) \]

Evaluate a candidate:

\[ r_m^l = f_{\pi}^l(R_m, F_T, L, M_{l-1}) = \sum_{n=1}^{N} \tilde{w}(f_n, q, g_{l-1})e_{mn} \]

Set of columns

As we weight the vector in the direction of each successive operation, it gets closer to any applicable answer in our DB.

Row \( m \), \( R = \) our table

# Semantic trees

<table>
<thead>
<tr>
<th></th>
<th>MIXTURED-25K</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SEMPRE</td>
<td>N2N</td>
<td>SbS</td>
<td>N2N - OOV</td>
</tr>
<tr>
<td>SELECT-WHERE</td>
<td>93.8%</td>
<td>96.2%</td>
<td>99.7%</td>
<td>90.3%</td>
</tr>
<tr>
<td>SUPERLATIVE</td>
<td>97.8%</td>
<td>98.9%</td>
<td>99.5%</td>
<td>98.2%</td>
</tr>
<tr>
<td>WHERE_SUPERLATIVE</td>
<td>34.8%</td>
<td>80.4%</td>
<td>94.3%</td>
<td>79.1%</td>
</tr>
<tr>
<td>NEST</td>
<td>34.4%</td>
<td>60.5%</td>
<td>92.1%</td>
<td>57.7%</td>
</tr>
<tr>
<td>Overall Acc.</td>
<td>65.2%</td>
<td>84.0%</td>
<td>96.4%</td>
<td>81.3%</td>
</tr>
</tbody>
</table>

SbS is N2N with **bias**.

SEMPRE is a toolkit for training semantic parsers, which map natural language utterances to denotations (answers) via intermediate logical forms. Here’s an example for querying databases. [https://nlp.stanford.edu/software/sempre/](https://nlp.stanford.edu/software/sempre/)

Semantic trees

<table>
<thead>
<tr>
<th></th>
<th>Mixtured-25K</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SEMPRE</td>
<td>N2N</td>
<td>SbS</td>
<td>N2N - OOV</td>
</tr>
<tr>
<td>SELECT-WHERE</td>
<td>93.8%</td>
<td>96.2%</td>
<td>99.7%</td>
<td>90.3%</td>
</tr>
<tr>
<td>SUPERLATIVE</td>
<td>97.8%</td>
<td>98.9%</td>
<td>99.5%</td>
<td>98.2%</td>
</tr>
<tr>
<td>WHERE SUPERLATIVE</td>
<td>34.8%</td>
<td>80.4%</td>
<td>94.3%</td>
<td>79.1%</td>
</tr>
<tr>
<td>NEST</td>
<td>34.4%</td>
<td>60.5%</td>
<td>92.1%</td>
<td>57.7%</td>
</tr>
<tr>
<td>Overall Acc.</td>
<td>65.2%</td>
<td>84.0%</td>
<td>96.4%</td>
<td>81.3%</td>
</tr>
</tbody>
</table>

SbS is N2N with **bias**.

SEMPRE is a toolkit for training semantic parsers, which map natural language utterances to denotations (answers) via intermediate logical forms. Here’s an example for querying databases. [https://nlp.stanford.edu/software/sempre/](https://nlp.stanford.edu/software/sempre/)

SbS is N2N with bias.

SEMPRE is a toolkit for training semantic parsers, which map natural language utterances to denotations (answers) via intermediate logical forms. Here’s an example for querying databases. https://nlp.stanford.edu/software/sempre/

Re-entry

**Task:** given a set of facts or a story, answer questions on those facts in a logically consistent way.

**Problem:** Theoretically, it could be achieved by a language modeler such as a recurrent neural network (RNN).

However, their memory (encoded by hidden states and weights) is typically too small, and is not compartmentalized enough to accurately remember facts from the past (knowledge is compressed into dense vectors).

RNNs are known to have difficulty in memorization tasks, i.e., outputting the same input sequence they have just read.

Re-entry

**Task:** given a set of facts or a story, answer questions on those facts in a logically consistent way.

- Jason went to the kitchen.
- Jason picked up the milk.
- Jason travelled to the office.
- Jason left the milk there.
- Jason went to the bathroom.

Where is the milk?

Re-entry

Memory networks

At the reading stage...

F = set of facts (e.g. recent context of conversation; sentences)
M = memory
for fact \( f \) in F:  # any linguistic pre-processing may apply
    store \( f \) in the next available memory slot of M

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." 
Re-entry

Memory networks

At the inference stage...

\[ o_2 = O_2(x, m) = \arg \max_{i=1,\ldots,N} s_O([x, m_{o_1}], m_i) \]

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." 
Re-entry

Memory networks

At the inference stage...

F = set of facts (e.g. recent context of conversation; sentences)
M = memory
O = inference module (oracle)
x = input
q = query about our known facts
k = number of candidate facts to be retrieved
m = a supporting fact (typically, the one that maximizes support)

\[ o_2 = O_2(x, m) = \arg\max_{i=1,\ldots,N} s_O([x, m_{o_i}], m_i) \]

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks."
Memory networks

At the inference stage, $t_1$...

$$o_2 = O_2(x, m) = \arg \max_{i=1,\ldots,N} s_{O_2} [x, m_{o_1}] m_i$$

Candidate space for $k = 2$

- jason go kitchen
- jason get milk
- jason go office
- jason drop milk
- jason go bathroom

argmax over that space

Re-entry

Memory networks

At the inference stage, $t_2$...

$$o_2 = O_2(x, m) = \arg\max_{i=1,...,N} s_O(x, m_{o_1} m_i)$$

Candidate space for $k = 2$

- jason go kitchen
- jason get milk
- jason go office
- jason drop milk
- jason go bathroom

argmax over that space

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." 
Re-entry

Memory networks

At the inference stage, $t_2$...

$$o_2 = O_2(x, m) = \arg \max_{i=1, \ldots, N} s_Q[x, m_{o_1}, m_i]$$

<table>
<thead>
<tr>
<th>jason go kitchen</th>
<th>jason go kitchen</th>
</tr>
</thead>
<tbody>
<tr>
<td>jason get milk</td>
<td>jason get milk</td>
</tr>
<tr>
<td>jason go office</td>
<td>jason go office</td>
</tr>
<tr>
<td>jason drop milk</td>
<td>jason drop milk</td>
</tr>
<tr>
<td>jason go bathroom</td>
<td>jason go bathroom</td>
</tr>
</tbody>
</table>

$jason, get, milk = \text{does not contribute as much new information as } jason \text{ go office}$

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." 
Re-entry

Memory networks

Training is performed with a margin ranking loss and stochastic gradient descent (SGD).

Simple model.

Easy to integrate with any pipeline already working with structured facts (e.g. Neural Inquirer). Adds a temporal dimension to it.

All previous research in coreference resolution applies.

Answering the question about the location of the milk requires comprehension of the actions *picked up* and *left*.

## Re-entry

### Memory networks

<table>
<thead>
<tr>
<th>QACNN</th>
<th>CBT-NE</th>
<th>CBT-CN</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>69.4</td>
<td>66.6</td>
<td>63.0</td>
<td></td>
</tr>
<tr>
<td>75.4</td>
<td>71.0</td>
<td>68.9</td>
<td>4 Mar 16</td>
</tr>
<tr>
<td>76.1</td>
<td>72.0</td>
<td>71.0</td>
<td>7 Jun 16</td>
</tr>
<tr>
<td>74.0</td>
<td>71.8</td>
<td>70.6</td>
<td>7 Jun 16</td>
</tr>
<tr>
<td>77.4</td>
<td>71.9</td>
<td>69.0</td>
<td>5 Jun 16</td>
</tr>
</tbody>
</table>


Results on MovieQA

<table>
<thead>
<tr>
<th>Source: Antoine Bordes, Facebook AI Research, LXMLS Lisbon July 28, 2016</th>
</tr>
</thead>
</table>

- **Standard QA System on KB**
  - KB: 78.5%
  - IE: 63.4%
  - Wikipedia: 69.9%

- **No Knowledge (embeddings)**
  - KB: 93.9%
  - IE: 68.3%
  - Wikipedia: 76.2%

Memory Networks

Key-Value Memory Networks
Conclusions
Enable semantic parsing using a neuralized architecture with respect to some knowledge base serving as ground truth.

Use memory networks to effectively keep track of the entities throughout the conversation and ensure logical consistency.

Use a modular neuralized architecture to compute answers for unseen questions dynamically.

Extend the neuralized architecture's domain by integrating support for supersenses. Enable inference at training and test time.

Extend memory networks by integrating support for supersenses. Enable inference at training and test time.
Summary of desiderata

Remember?

Conversational technology should really be about

• dynamically finding the best possible way to browse a large repository of information/actions.

• find the shortest path to any relevant action or piece of information (to avoid the plane dashboard effect).

• surfacing implicit data in unstructured content ("bocadillo de calamares in Madrid"). Rather than going open-domain, taking the closed-domain and going deep into it.
We've got what you're looking for!
Head to Floor 1 near Men's Collections
For more info on colors, sizes & more, click the image below.