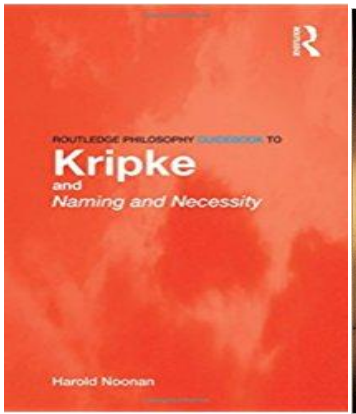
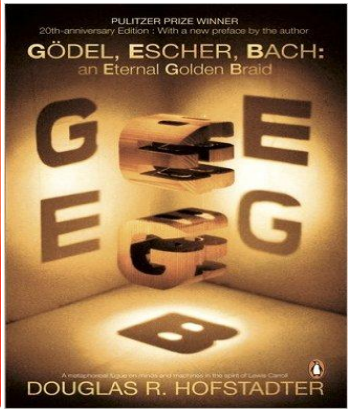
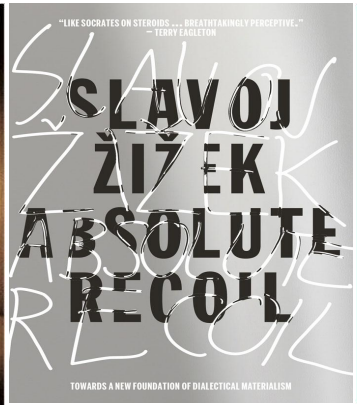
"It is very unlikely that machines will exhibit broadly-applicable intelligence comparable to humans in the next 20 years."

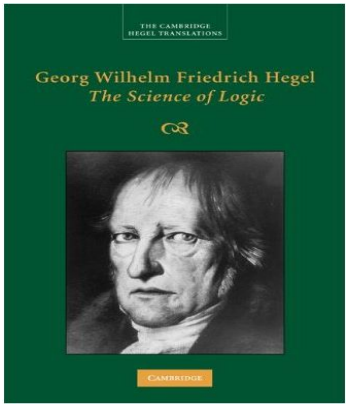
- the government, last week

Philosophy of Mind/ Language

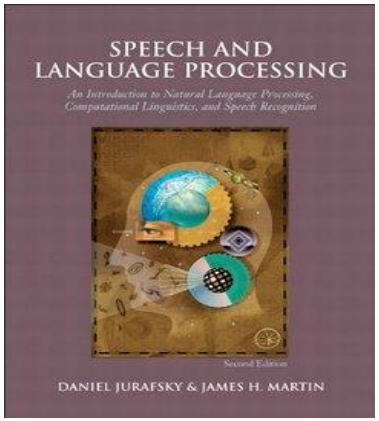


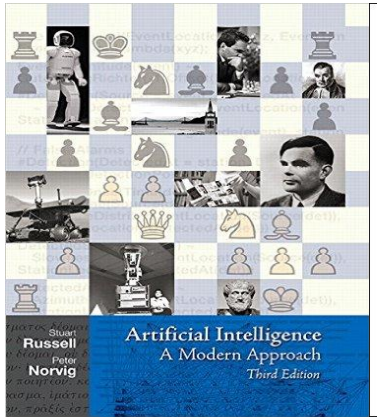


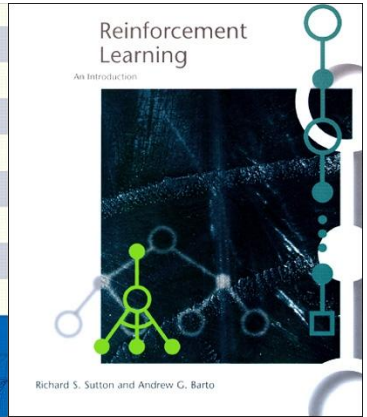




Meets NLP and AI







Winograd Schemas

Most Winograd Schemas have 2 antecedents, and 1 mystery reference.

The trophy doesn't fit into the brown suitcase because it's too [small / large]

The lawyer asked the witness a question, but he was reluctant to [repeat / answer] it

It was a summer day, and the dog was sitting in the middle of the lawn. After a while, it got up and moved to a spot under the tree, because it was [hot / cooler]

The painting in Mark's living room shows an oak tree. It is to the right of [the bookcase / a house]

Winograd Schemas

It's supposed to test for "common sense" in AI.

- AKA anaphora resolution or coreference resolution

Annual competition is mostly disaster

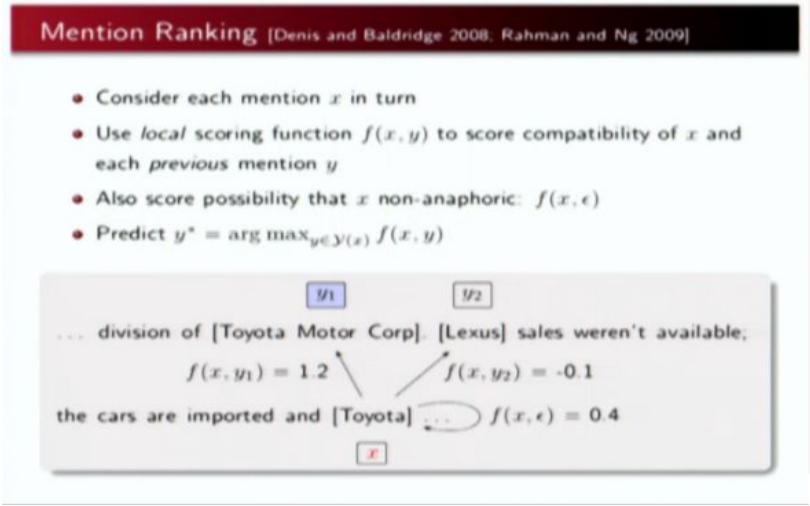
- best score 58% accuracy (vs. 44% if randomly guessing, i.e. 14% better than random)

Other competitor guy

I will enter the next competition!

- bring glory to General Assembly

Standard stats approach to coreference resolution



Current Evaluation Results

The scores of the dcoref code in v3.6.0 (CoNLL 2011 shared task winner descendant) on the CoNLL 2011 Shared Task dev data set, measured on 2016/02/07 using the v4 scorer (used for the 2011 evaluation).

	MUC			B cubed			CEAF (M)			CEAF (E)			BLANC			
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	Avg F1
conllst2011 dev	62.1	59.3	60.7	74.2	67.7	70.8	59.4	59.4	59.4	46.1	48.9	47.5	79.6	72.4	75.4	59.56

* Automatic mention detection used. Avg F1 = (MUC + B cubed + CEAFE)/3.

The scores of the dcoref code in v3.6.0 (CoNLL 2011 shared task winner descendant) on the CoNLL 2011/2012 Shared Task dev data sets, measured on 2016/02/07 using the v8.01 scorer (current in 2016).

	MUC			B cubed			CEAF (M)			CEAF (E)			BLANC			
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	Avg F1
conllst2011 dev	62.1	59.3	60.7	56.2	48.6	52.1	58.0	57.5	57.8	48.9	53.5	51.1	54.1	47.2	50.1	54.62
conllst2012 dev	65.9	64.1	65.0	58.7	50.9	54.5	59.2	59.6	59.4	48.6	54.3	51.3	59.5	53.7	56.1	56.92

* Automatic mention detection used. Avg F1 = (MUC + B cubed + CEAFE)/3.

Does terribly on Winograd schema-like questions, but has more general purpose, more applicable

2 Approaches

(The good stuff is towards the end)

1st approach:

Throw deep learning at it:

- The idea: A *general-purpose* pronoun disambiguator, trained on millions of sentences, can abstract itself to solve at least 14% of Winograd Schemas.
- Fair assumption due to RNN-innovations

Throw deep learning approach - Data

1 GB of unpublished contemporary novels ; 100s millions sentences

High specificity; sentences must have 2 antecedents + pronoun, and not be ambiguous: 1 of out 100 sentences are usable.

"The Professor appreciates John because he works hard" → "[A]-ant appreciates [B]-ant because [???] works hard" → [B]

Standardize NER (locations, date, quantity etc.), collocations, light verb constructs, heuristics

Stanford Co-Reference resolver as assiter (next page)

text	val
[A] wasn't sure what [???] wanted [A] to recal...	[B]
[A] wasn't sure what [B] wanted [???] to recal...	[A]
[A] wasn't sure what [B] wanted [A] to recall ...	[B]
Though the clouds darken the sun... " [???] wo...	[A]
Though the clouds darken the sun... " [A] word...	[B]
Though the clouds darken the sun... " [A] word...	[B]
Though the clouds darken the sun... " [A] word...	[B]
Though the clouds darken the sun... " [A] word...	[A]
Though the clouds darken the sun... " [A] word...	[B]
[A]-ant opened [???] eyes and found [B]-ant st...	[A]

Why LSTM-RNN?

Can abstract out from its data to new situations

“The white blood cells destroyed the virus” vs. “The virus destroyed the white blood cells”.

It “remembers” stuff said early in a sentence or paragraph.

Lexical features

Deep learning - LSTM results

60k training examples sentences ~ 45 min.

3 million originally parsed ~ 5 hours

Stochastic gradient descent

Dropout

Notice validation vs. train (next page)

80% accuracy typically epoch 7-8 (high bias, so varies)

66% null model - distribution of A to B (66% vs. 33%)

```
---------------------------------
Run id: ZB1L4P
Log directory: /tmp/tflearn_logs/
---------------------------------
Training samples: 50000
Validation samples: 10811
--
Training Step: 834  | total loss: 0.62067
| Adam | epoch: 001 | loss: 0.62067 - acc: 0.6882 | val_loss: 0.60568 - val_acc: 0.7034 -- iter: 50000/50000
--
Training Step: 1668  | total loss: 0.60710
| Adam | epoch: 002 | loss: 0.60710 - acc: 0.6623 | val_loss: 0.58910 - val_acc: 0.6887 -- iter: 50000/50000
--
Training Step: 2502  | total loss: 0.58272
| Adam | epoch: 003 | loss: 0.58272 - acc: 0.6620 | val_loss: 0.55811 - val_acc: 0.6763 -- iter: 50000/50000
--
Training Step: 3336  | total loss: 0.58533
| Adam | epoch: 004 | loss: 0.58533 - acc: 0.6679 | val_loss: 0.56932 - val_acc: 0.6763 -- iter: 50000/50000
--
Training Step: 4170  | total loss: 0.57210
| Adam | epoch: 005 | loss: 0.57210 - acc: 0.6412 | val_loss: 0.54646 - val_acc: 0.6763 -- iter: 50000/50000
--
Training Step: 5004  | total loss: 0.38031
| Adam | epoch: 006 | loss: 0.38031 - acc: 0.8034 | val_loss: 0.42697 - val_acc: 0.7784 -- iter: 50000/50000
--
Training Step: 5838  | total loss: 0.26603
| Adam | epoch: 007 | loss: 0.26603 - acc: 0.8811 | val_loss: 0.50989 - val_acc: 0.7836 -- iter: 50000/50000
--
Training Step: 6672  | total loss: 0.14468
| Adam | epoch: 008 | loss: 0.14468 - acc: 0.9415 | val_loss: 0.69982 - val_acc: 0.7861 -- iter: 50000/50000
--
Training Step: 7506  | total loss: 0.14143
| Adam | epoch: 009 | loss: 0.14143 - acc: 0.9522 | val_loss: 0.67460 - val_acc: 0.7792 -- iter: 50000/50000
--
Training Step: 8340  | total loss: 0.06908
| Adam | epoch: 010 | loss: 0.06908 - acc: 0.9720 | val_loss: 0.99620 - val_acc: 0.7743 -- iter: 50000/50000
```

Approach 1 - Limitations

Need million + training examples - timely to prepare

Not that cognitively plausible IMO, are more interesting approaches

Most likely will never solve 100% winograd schemas, if 50%

The AI approach

Classic AI plus some ML:

- For any Winograd Schema there are learned frame(s) that represents the “background knowledge” that makes it pretty obvious what the pronoun refers to.
- Learn how to ‘reason’ with the frames

AI approach - representing language

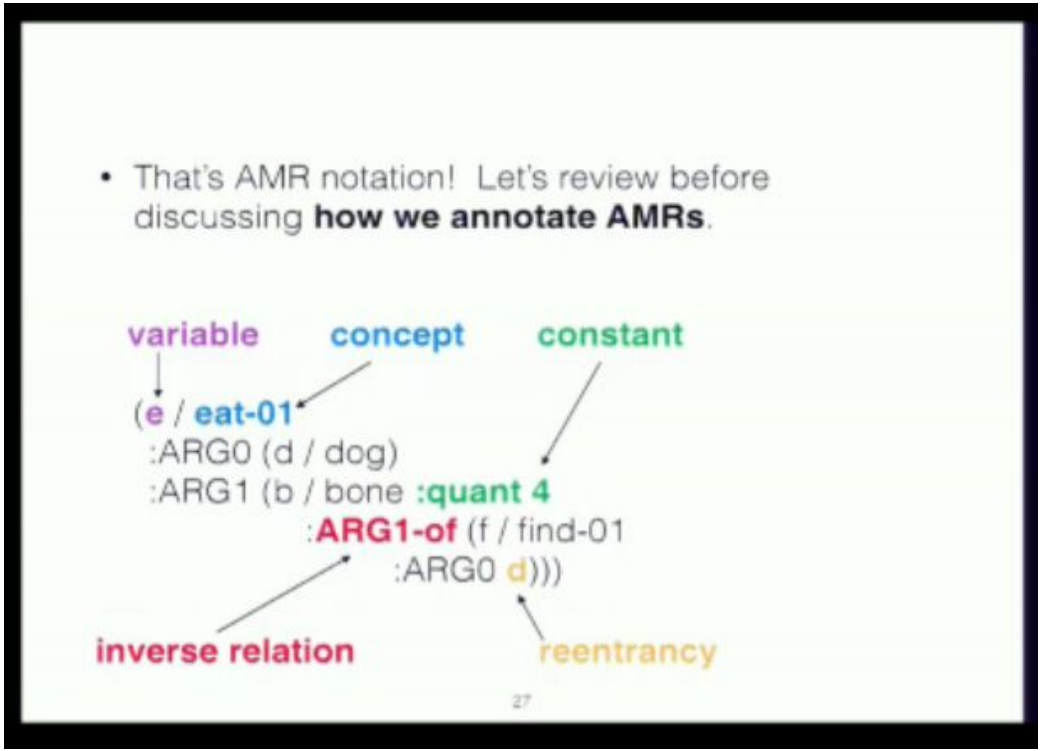
Symbolic reasoning

Many choices: First order logic, higher order logic, propositional logic, OWL, AMR, KIF, SUO-KIF, natural language etc.

My choice: SUB - PRED *(OBJ)

On the side of natural language, and natural logic (NatLog) where necessary; but

It's 'common sense' it's not hardocre logic.



Getting ‘common sense’ frames

In tricky place between definitions and axioms.

“Common experience assertions” or “middle level axioms” (ie. not too abstract, and not too technical).

- “Getting stuck in traffic can make you angry”

Combination of:

- Automatically extracting them from text (“IE”)
  - Spacy heuristics (subj., pred., obj.)
  - University of Washington, Stanford tools
- Parse ontologies
  - My SUO-KIF parser (convert type of logic language to natural language)

The Frames v 0.1

~ 250,000 high quality frames ! [Ngrok URL]

Not 1 manually added - all from automatic concept extraction or parsing ontologies

Provides enough background info to solve ~30% of the Winograd Schema corpus, based on sampling; ~70% of WS corpus “come very close but slightly off”

Some interesting ones:

Something that might happen while bringing home some fish is cats follow you along the street

Modern economics is shrouded in idiosyncratic self-serving definitions

Frames disambiguating

Most basic heuristic: The WS has 2 explicit references and a pronoun, so the right frame must capture the pronoun and at least 1 of the explicit references.

WS: The trophy doesn't fit into the brown suitcase because it's too large.

Frame: A larger thing can not fit inside a smaller thing

WS: Joan made sure to thank Susan for all the help she had given.

Frame: Something that might happen when you give assistance is you may be thanked

WS: The lawyer asked the witness a question, but he was reluctant to answer it.

Frames: A lawyer can question a witness; You would answer questions because you were asked a question

Word matching; SRL - Semantic Role Labeling a.k.a. coordinate predicate references; NLP “alignment” methods → pretty automatic.

Helping in this whole process… not a blank slate

The ‘substrate’/ lowest level:

- ‘Knows’ all common predicates, the arguments/roles that are part of the predicates, the range of syntactic realizations for the predicates. It knows broader semantic situations and things and people typically involved. It understands the meaning of hundreds of thousands of collocations. It knows the definition of almost all words, their synonyms, and their hypernymy.

Propbank, FrameNet, Verbnet, WordNet, some custom resources too (collocations, polysemy).

```python
class OxCollocations():
    def __init__(self, pdf=None, current_page=None):
        if not pdf:
            pdf = pyPdf.PdfFileReader(open("/Users/richfisher/Downloads/oxford_collocations_dictionary.pdf", "rb"))
        self.pdf = pdf
        #self.set_header_regex = re.compile("[A-Z]{2}[A-Z\.\s\+]{1,26}") #QUANT. / ADJ. / CHAIN + NOUN etc.
        self.set_header_regex = re.compile("[A-Z]{1}[A-Z\.\s\+]{2,26}")

        self.all_coll_chunks = []
        self.skipped_multi_page = []
        self.exceptions_in_iter = []
        self.all_templates = []
        self.current_template = None
        self.current_page = current_page

        self.first_entry = self.pdf.pages[self.current_page].extractText()
        self.base_word = self.get_base_word()
        self.base_word_pos = self.get_base_word_pos()
        self.base_word_with_pos = self.base_word + "_" + self.base_word_pos
        self.multi_page = self.determine_if_multi_page()
        self.num_pages_for_word = self.page_tracker()[1]
```

```python
class Propbank():
    def __init__(self, parser=None, frames_path=None):
        if not frames_path:
            frames_path = "/Users/richfisher/projects/cortazar_data/propbank-frames/frames"
        if not parser:
            parser = None#English()
        self.parser = parser
        self.file_name_list = [f for f in listdir(frames_path) if isfile(join(frames_path, f))]
        self.all_files_parsed = self.parse_all_files() #7,308 xml sets
        #self.current_frame_xml = None
        #self.current_template = None
        self.all_examples = []
        self.all_templates = self.build_all_templates()
        self.all_rolesets = self.get_arr_of_all_rolesets()

    def pp_template(self, template):
        pp(json.loads(json.dumps(template)))
        #pp(json.loads(json.dumps(rs[29])))

    def get_base_template(self):
        return defaultdict(str, {
            "lemma": "",
            "num_rolesets": 1,
            "rolesets": defaultdict(str,{
                "roleset1": defaultdict(str,{
                    "meta": defaultdict(str, {
                        "alias_txts": [],
                        "framenet_names": [],
                        "verbnet_names": [],
                        "num_roles": 0,
                        "num_examples": 0
                    }),
                    "roles": [], #[n_val, f_val, descrip], [n_val, f_val, descrip]... #different than argtype below
                    "examples": defaultdict(str,{
                        "example1": defaultdict(str, {
                            "name": "", #name tag in <example>
                            "text": "",
                            "args": [] #[argtype, f_val, text], [argtype, f_val, text]... argtype can be arg0, arg1, rel, argm
                        })
                    })
                })
            })
        })

    def parse_all_files(self):
```

```python
class Verbnet():
    def __init__(self, parser=None):
        self.files_path = "/Users/richfisher/projects/cortazar_data/verbnet/"
        self.parser = parser #for mining examples
        self.file_name_list = [f for f in listdir(self.files_path) if isfile(join(self.files_path, f)) and f[-3:] == "xml"]
        self.all_files_parsed = self.parse_all_files() #277
        self.all_templates = self.build_all_templates()
        self.all_words_per_group = self.evocation_words_per_main_class()

    def pp_template(self, template):
        pp(json.loads(json.dumps(template)))

    def parse_all_files(self):
        #self.file_name_list.remove('frameset.dtd')
        parsed = []
        for fname in self.file_name_list:
            parsed.append(ET.parse(self.files_path + fname))
        return parsed

    def build_all_templates(self):
        complete_templates = []
```

```python
class Framenet():
    def __init__(self, parser=None, frames_path=None):
        if not frames_path:
            frames_path = "/Users/richfisher/projects/cortazar_data/framenet-frames/"
        if not parser:
            parser = None#English()
        self.frames_path = frames_path
        self.parser = parser
        self.lu_pat = re.compile("([a-zA-Z\s\d\-\(\)]+)\.")
        self.file_name_list = [f for f in listdir(self.frames_path) if isfile(join(self.frames_path, f)) and f[-3:] == "xml"]
        self.all_files_parsed = self.parse_all_files() # 1221
        self.all_templates = self.build_all_templates()


    def pp_template(self, template):
        pp(json.loads(json.dumps(template)))

    def parse_all_files(self):
        parsed = []
        for fname in self.file_name_list:
            parsed.append(ET.parse(self.frames_path + fname))
        return parsed

    def build_all_templates(self):
        completed_templates = []
        for parsedf in self.all_files_parsed:
            completed_templates.append(self.fill_base_template(parsedf))
        return completed_templates

    def fill_base_template(self, parsedf):
        root = parsedf.getroot()
        frame_name = root.attrib['name'].replace("_", " ").lower()
        all_els = root.getchildren()
        def_raw = filter(lambda x: 'definition' in str(x), all_els)[0].text
        all_lex = filter(lambda x: 'lexunit' in str(x), all_els)
        all_lex_word_pos = []
        for lex in all_lex:
            pos = lex.attrib['pos']
            name = lex.attrib['name'].replace("]", '').replace("[",'')
            find = self.lu_pat.findall(name)
            if find:
                find = find[0].lower()
                all_lex_word_pos.append([find, pos])
        base_t = defaultdict(str,{
            'f_name': frame_name,
            'def_raw': def_raw,
            'lus': all_lex_word_pos
```

An “interpretation”/ “association layer” using ML

Can be hacky otherwise, no “weighted” sense for frames for certain situations; slightly off frames, should use a 2nd frame, needs a more general frame, etc.

Tried to imagine an ‘interpretation’ layer that processes each prompt.

To mimic the way we naturally associate.

It can be done!

Seq2Seq / Encoder - Decoder

- Deep learning multi models

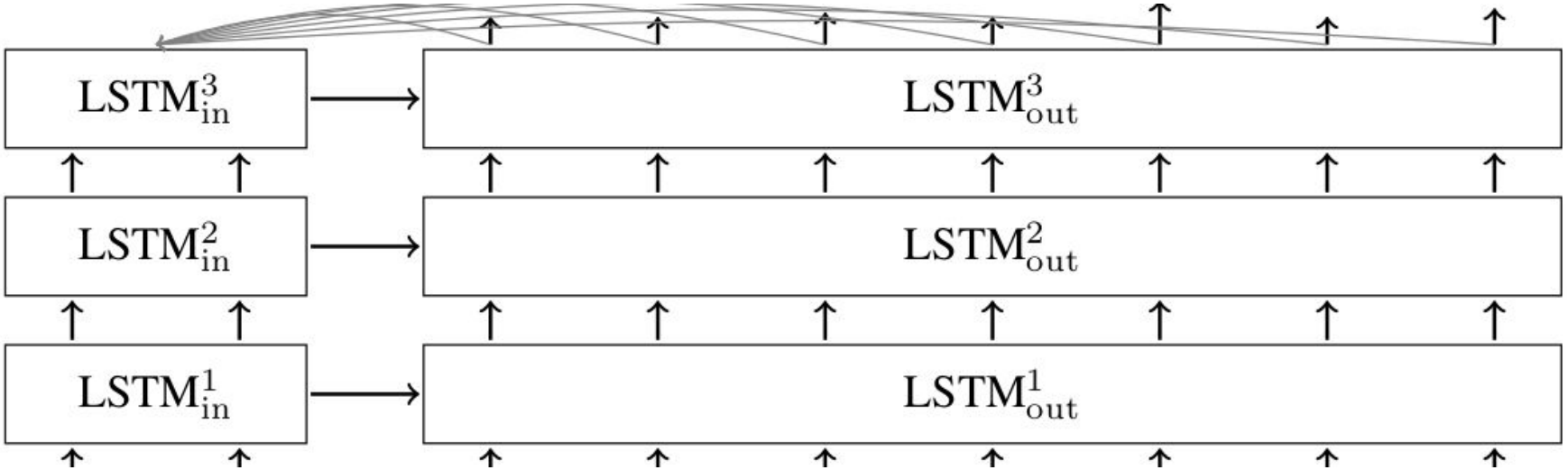
Take state-of-art technology used for translating English to German, and translate from English to English.

Recent research --> 80% + accuracy

Zero examples of this implemented online

Hacked it together in Tensorflow

Seq2seq, machine translation deep learning



Seq2Seq

- “It was a summer day, and the dog was sitting in the middle of the lawn. After a while, it got up and moved to a spot under the tree, because it was hot.”
- Spits out:
- “The animal was hot so it moved”

Useful for evoking more general, explanatory power frames, or mediating between frames, or even for sentence generation.

LSTM w/ Attention pairs; data - results

Various summarization and entailment corpora - 200k samples

Metric is “perplexity”

- How confident it feels about probability of next words in a sequence

Uses buckets

Hidden layer nodes: 1024; batch size 40

Validation vs. train

Automatically adjusts learning rate (first adjustment was at 24 hour mark)

Saves checkpoints, ability to reload, because many-day training process on my CPU

```
global step 50 learning rate 0.5000 step-time 2.04 perplexity 36934.31
  eval: bucket 0 perplexity 5348.12
  eval: bucket 1 perplexity 5574.72
  eval: bucket 2 perplexity 5822.01
  eval: bucket 3 perplexity 9613.48
global step 100 learning rate 0.5000 step-time 2.13 perplexity 2176.79
  eval: bucket 0 perplexity 914.64
  eval: bucket 1 perplexity 1143.85
  eval: bucket 2 perplexity 889.72
  eval: bucket 3 perplexity 679.30
global step 150 learning rate 0.5000 step-time 1.92 perplexity 806.21
  eval: bucket 0 perplexity 157.41
  eval: bucket 1 perplexity 327.96
  eval: bucket 2 perplexity 368.92
  eval: bucket 3 perplexity 445.66
global step 200 learning rate 0.5000 step-time 1.79 perplexity 488.25
  eval: bucket 0 perplexity 184.32
  eval: bucket 1 perplexity 351.03
  eval: bucket 2 perplexity 380.10
  eval: bucket 3 perplexity 375.09
global step 250 learning rate 0.5000 step-time 2.15 perplexity 433.34
  eval: bucket 0 perplexity 173.22
  eval: bucket 1 perplexity 308.06
  eval: bucket 2 perplexity 328.33
  eval: bucket 3 perplexity 322.28
global step 300 learning rate 0.5000 step-time 2.25 perplexity 375.13
  eval: bucket 0 perplexity 134.09
  eval: bucket 1 perplexity 255.20
  eval: bucket 2 perplexity 292.05
  eval: bucket 3 perplexity 339.11
global step 350 learning rate 0.5000 step-time 2.03 perplexity 347.47
  eval: bucket 0 perplexity 215.51
  eval: bucket 1 perplexity 312.70
  eval: bucket 2 perplexity 304.75
  eval: bucket 3 perplexity 339.18
```

```
  eval: bucket 0 perplexity 13.25
  eval: bucket 1 perplexity 8.40
  eval: bucket 2 perplexity 11.96
  eval: bucket 3 perplexity 10.27
global step 15000 learning rate 0.4432 step-time 9.82 perplexity 8.86
  eval: bucket 0 perplexity 9.87
  eval: bucket 1 perplexity 11.34
  eval: bucket 2 perplexity 12.34
  eval: bucket 3 perplexity 15.46
global step 15100 learning rate 0.4432 step-time 10.82 perplexity 9.03
  eval: bucket 0 perplexity 11.95
  eval: bucket 1 perplexity 15.51
  eval: bucket 2 perplexity 16.90
  eval: bucket 3 perplexity 14.57
global step 15200 learning rate 0.4388 step-time 9.69 perplexity 8.67
  eval: bucket 0 perplexity 10.54
  eval: bucket 1 perplexity 12.74
  eval: bucket 2 perplexity 16.29
  eval: bucket 3 perplexity 13.35
global step 15300 learning rate 0.4388 step-time 9.59 perplexity 8.63
  eval: bucket 0 perplexity 11.14
  eval: bucket 1 perplexity 11.21
  eval: bucket 2 perplexity 10.38
  eval: bucket 3 perplexity 16.85
global step 15400 learning rate 0.4388 step-time 9.39 perplexity 8.69
  eval: bucket 0 perplexity 10.92
  eval: bucket 1 perplexity 15.09
  eval: bucket 2 perplexity 12.86
  eval: bucket 3 perplexity 13.02
global step 15500 learning rate 0.4388 step-time 10.67 perplexity 8.82
  eval: bucket 0 perplexity 9.93
  eval: bucket 1 perplexity 15.21
  eval: bucket 2 perplexity 13.13
  eval: bucket 3 perplexity 11.59
global step 15600 learning rate 0.4344 step-time 8.92 perplexity 8.68
  eval: bucket 0 perplexity 15.08
  eval: bucket 1 perplexity 10.52
  eval: bucket 2 perplexity 15.89
  eval: bucket 3 perplexity 13.17
```

Googler on Seq2Seq on Tensorflow help forum

“I think the perplexity needs to go to around 4 for the results to be good. On a single GPU, this can take about a month of training..."

-For using Seq2Seq for language translation on millions of examples

AI approach

What’s next?

- Towards a massive graph where nodes are concepts and edges are learned weights between concepts, managed and manipulated by objective functions that mimic cognition --> philosophy goes here !
- Multi agent, online learning setup
  - Prepare for dialogue, conversation tasks
  - Accelerate process of strengthening weights between frames/ concepts

Ngrok demo

Ngrok URL forthcoming

Pick words/ phrases to see what 250k frames come up

Pick sentences to see what the ‘interpretation layer’ throws back.