Natural Language Inference

Christopher Potts

Stanford Linguistics

CS 224U: Natural language understanding
April 27 and 29
Overview

1. Overview
2. SNLI, MultiNLI, and Adversarial NLI
3. Hand-built features
4. nli.experiment
5. Sentence-encoding models
6. Chained models
7. Attention
8. Error analysis
**Overview**

SNLI, MultiNLI, ANLI

Hand-built features

nli.experiment

Sentence-encoding

Chained

Attention

Error analyses

---

**Associated materials**

1. Code
   a. nli.py
   b. nli_01_task_and_data.ipynb
   c. nli_02_models.ipynb

2. Homework and bake-off: hw_wordentail.ipynb

3. Core readings: Bowman et al. 2015; Williams et al. 2018; Nie et al. 2019b; Rocktäschel et al. 2016

4. Auxiliary readings: Goldberg 2015; Dagan et al. 2006; MacCartney and Manning 2008; Gururangan et al. 2018
## Simple examples

<table>
<thead>
<tr>
<th>Premise</th>
<th>Relation</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>A turtle danced.</td>
<td>entails</td>
<td>A turtle moved.</td>
</tr>
<tr>
<td>turtle</td>
<td>contradicts</td>
<td>linguist</td>
</tr>
<tr>
<td>Every reptile danced.</td>
<td>neutral</td>
<td>A turtle ate.</td>
</tr>
<tr>
<td>Some turtles walk.</td>
<td>contradicts</td>
<td>No turtles move.</td>
</tr>
<tr>
<td>James Byron Dean refused to move without blue jeans.</td>
<td>entails</td>
<td>James Dean didn’t dance without pants.</td>
</tr>
<tr>
<td>Mitsubishi Motors Corp’s new vehicle sales in the US fell 46 percent in June.</td>
<td>contradicts</td>
<td>Mitsubishi’s sales rose 46 percent.</td>
</tr>
<tr>
<td>Acme Corporation reported that its CEO resigned.</td>
<td>entails</td>
<td>Acme’s CEO resigned.</td>
</tr>
</tbody>
</table>
NLI task formulation

Does the premise justify an inference to the hypothesis?

- Commonsense reasoning, rather than strict logic.
- Focus on local inference steps, rather than long deductive chains.
- Emphasis on variability of linguistic expression.

Perspectives

- Zaenen et al. (2005): Local textual inference: can it be defined or circumscribed?
- Manning (2006): Local textual inference: it’s hard to circumscribe, but you know it when you see it – and NLP needs it.
- Crouch et al. (2006): Circumscribing is not excluding: a reply to Manning.
Connections to other tasks

Dagan et al. (2006)

It seems that major inferences, as needed by multiple applications, can indeed be cast in terms of textual entailment.

[...] Consequently, we hypothesize that textual entailment recognition is a suitable generic task for evaluating and comparing applied semantic inference models. Eventually, such efforts can promote the development of entailment recognition “engines” which may provide useful generic modules across applications.
## Connections to other tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>NLI framing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paraphrase</td>
<td>text ≡ paraphrase</td>
</tr>
<tr>
<td>Summarization</td>
<td>text ⊏ summary</td>
</tr>
<tr>
<td>Information retrieval</td>
<td>query ⊑ document</td>
</tr>
<tr>
<td>Question answering</td>
<td>question ⊑ answer</td>
</tr>
<tr>
<td></td>
<td><em>Who left? ⇒ Someone left</em></td>
</tr>
<tr>
<td></td>
<td><em>Someone left ⊑ Sandy left</em></td>
</tr>
</tbody>
</table>
Models for NLI

- Hand-built features
- nli.experiment
- Sentence-encoding
- Chained
- Attention
- Error analyses

- SNLI, MultiNLI, ANLI

- Overview

Deep, brittle

- Logic and theorem proving
- Natural Logic
- Semantic graphs
- Clever hand-built features
- Deep learning (2015)
- N-gram variations

Robust, shallow

See the Excitement Open Platform (Pado et al. 2012)

A standard baseline, often very robust!

- Bos & Markert 2005
- MacCartney 2009
- Hickl et al. 2006; de Marneffe et al. 2006

Effectiveness

Depth of representations
Models for NLI

- SNLI, MultiNLI, ANLI
- Hand-built features
- nli.experiment
- Sentence-encoding
- Chained
- Attention
- Error analyses

Bos & Markert 2005
MacCartney 2009
Hickl et al. 2006; de Marneffe et al. 2006

Logic and theorem proving
Natural Logic
Semantic graphs
N-gram variations
Clever hand-built features
Deep learning (2017–)

effectiveness

depth of representations

- Robust, shallow
- Deep, brittle

See the Excitement Open Platform (Pado et al. 2012)

Standard baseline, often very robust!
Other NLI datasets
Other NLI datasets

Recent

- The GLUE benchmark (diverse tasks including NLI)
  https://gluebenchmark.com
- NLI Style FEVER
  https://github.com/easonnie/combine-FEVER-NSMN/blob/master/other_resources/nli_fever.md
- MedNLI (derived from MIMIC III)
  https://physionet.org/physio-tools/mimic-code/mednli/
- XNLI is a multilingual NLI dataset derived from MultiNLI
  https://github.com/facebookresearch/XNLI
- Diverse Natural Language Inference Collection (DNC)
  http://decomp.io/projects/diverse-natural-language-inference/
- SciTail (derived from science exam questions and Web text)
  http://data.allenai.org/scitail/
Other NLI datasets

Older

- SemEval 2013
  [https://www.cs.york.ac.uk/semeval-2013/](https://www.cs.york.ac.uk/semeval-2013/)

- SemEval 2014: Sentences Involving Compositional Knowledge (SICK)

- The FraCaS textual inference test suite
  [https://nlp.stanford.edu/~wcmac/downloads/](https://nlp.stanford.edu/~wcmac/downloads/)
Other NLI datasets

Related

- 30M Factoid Question-Answer Corpus
  http://agarciaduran.org/
- The Penn Paraphrase Database
  http://paraphrase.org/
## Label sets

<table>
<thead>
<tr>
<th>2-way RTE 1,2,3</th>
<th>3-way RTE4, FraCaS, *NLI</th>
<th>4-way Sánchez-Valencia</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>couch</strong> sofa</td>
<td><strong>crow</strong> bird</td>
<td><strong>bird</strong> crow</td>
</tr>
<tr>
<td><strong>hippo</strong> hungry</td>
<td></td>
<td><strong>turtle</strong> linguist</td>
</tr>
<tr>
<td>Yes entailment</td>
<td>Yes entailment</td>
<td><strong>P ≡ Q</strong> equivalence</td>
</tr>
<tr>
<td></td>
<td>Unknown</td>
<td><strong>P ⊏ Q</strong> forward</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>P ⊐ Q</strong> reverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>P # Q</strong> non-entailment</td>
</tr>
<tr>
<td>No non-entailment</td>
<td></td>
<td><strong>P # Q</strong> non-entailment</td>
</tr>
</tbody>
</table>
NLI dataset artifacts

1. **Artifact**: A dataset bias that would make a system susceptible to adversarial attack even if the bias is linguistically motivated.

2. Tricky example: negated hypotheses signal contradiction
   - Linguistically motivated: negation is our best way of establishing relevant contradictions.
   - An artifact because we would curate a dataset in which negation correlated with the other labels but led to no human confusion.
Hypothesis-only baselines

- In his project for this course (2016), Leonid Keselman observed that hypothesis-only models are strong.

- Other groups have since further supported this (Poliak et al. 2018; Gururangan et al. 2018; Tsuchiya 2018; Belinkov et al. 2019)

- Likely due to artifacts:
  - Specific claims are likely to be premises in entailment cases.
  - General claims are likely to be hypotheses in entailment pairs.
  - Specific claims are more likely to lead to contradiction.
Known artifacts in SNLI and MultiNLI

- These datasets contain words whose appearance nearly perfectly correlates with specific labels [1, 2].
- Entailment hypotheses over-represent general and approximating words [2].
- Neutral hypotheses often introduce modifiers [2].
- Contradiction hypotheses over-represent negation [1, 2].
- Neutral hypotheses tend to be longer [2].

1 = Poliak et al. 2018, 2 = Gururangan et al. 2018
Artifacts in other tasks

- Visual Question Answering: Kafle and Kanan 2017; Chen et al. 2020
- Story Completion: Schwartz et al. 2017
- Reading Comprehension/Question Answering: Kaushik and Lipton 2018
- Stance Detection: Schiller et al. 2020
- Fact Verification: Schuster et al. 2019
## SNLI, MultiNLI, and Adversarial NLI

1. Overview
2. SNLI, MultiNLI, and Adversarial NLI
3. Hand-built features
4. nli.experiment
5. Sentence-encoding models
6. Chained models
7. Attention
8. Error analysis
SNLI

1. Bowman et al. 2015
2. All the premises are image captions from the Flickr30K corpus (Young et al. 2014).
3. All the hypotheses were written by crowdworkers.
5. 550,152 train examples; 10K dev; 10K test
6. Mean length in tokens:
   - Premise: 14.1
   - Hypothesis: 8.3
7. Clause-types:
   - Premise S-rooted: 74%
   - Hypothesis S-rooted: 88.9%
8. Vocab size: 37,026
9. 56,951 examples validated by four additional annotators.
   - 58.3% examples with unanimous gold label
   - 91.2% of gold labels match the author’s label
   - 0.70 overall Fleiss kappa
Crowdsourcing methods

Instructions

The Stanford University NLP Group is collecting data for use in research on computer understanding of English. We appreciate your help!

We will show you the caption for a photo. We will not show you the photo. Using only the caption and what you know about the world:

- Write one alternate caption that is **definitely a true** description of the photo.
- Write one alternate caption that **might be a true** description of the photo.
- Write one alternate caption that is **definitely an false** description of the photo.

**Photo caption** A little boy in an apron helps his mother cook.

**Definitely correct**   Example: For the caption "Two dogs are running through a field." you could write "There are animals outdoors."

Write a sentence that follows from the given caption.

**Maybe correct**     Example: For the caption "Two dogs are running through a field." you could write "Some puppies are running to catch a stick."

Write a sentence which may be true given the caption, and may not be.

**Definitely incorrect**   Example: For the caption "Two dogs are running through a field." you could write "The pets are sitting on a couch."

Write a sentence which contradicts the caption.

**Problems (optional)**   *If something is wrong with the caption that makes it difficult to understand, do your best above and let us know here.*
## Examples

<table>
<thead>
<tr>
<th>Premise</th>
<th>Relation</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>A man inspects the uniform of a figure in some East Asian country.</td>
<td>contradiction</td>
<td>The man is sleeping</td>
</tr>
<tr>
<td>An older and younger man smiling.</td>
<td>neutral</td>
<td>Two men are smiling and laughing at the cats playing on the floor.</td>
</tr>
<tr>
<td>A black race car starts up in front of a crowd of people.</td>
<td>contradiction</td>
<td>A man is driving down a lonely road.</td>
</tr>
<tr>
<td>A soccer game with multiple males playing.</td>
<td>entailment</td>
<td>Some men are playing a sport.</td>
</tr>
<tr>
<td>A smiling costumed woman is holding an umbrella.</td>
<td>neutral</td>
<td>A happy woman in a fairy costume holds an umbrella.</td>
</tr>
</tbody>
</table>
## Event coreference

<table>
<thead>
<tr>
<th>Premise</th>
<th>Relation</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>A boat sank in the Pacific Ocean.</td>
<td>contradiction</td>
<td>A boat sank in the Atlantic Ocean.</td>
</tr>
<tr>
<td>Ruth Bader Ginsburg was appointed to the Supreme Court.</td>
<td>contradiction</td>
<td>I had a sandwich for lunch today</td>
</tr>
</tbody>
</table>

If premise and hypothesis *probably* describe a different photo, then the label is contradiction.
Progress on SNLI

SNLI leaderboard: Systems over time

F1 score


Human
MultiNLI

1. Williams et al. 2018

2. Train premises drawn from five genres:
   - Fiction: works from 1912–2010 spanning many genres
   - Government: reports, letters, speeches, etc., from government websites
   - The *Slate* website
   - Telephone: the Switchboard corpus
   - Travel: Berlitz travel guides

3. Additional genres just for dev and test (the mismatched condition):
   - The 9/11 report
   - Face-to-face: The Charlotte Narrative and Conversation Collection
   - Fundraising letters
   - Non-fiction from Oxford University Press
   - *Verbatim*: articles about linguistics

4. 392,702 train examples; 20K dev; 20K test

5. 19,647 examples validated by four additional annotators
   - 58.2% examples with unanimous gold label
   - 92.6% of gold labels match the author’s label

6. Test-set labels available as a Kaggle competition.

7. Project page: [https://www.nyu.edu/projects/bowman/multinli/](https://www.nyu.edu/projects/bowman/multinli/)
## MultiNLI annotations

<table>
<thead>
<tr>
<th>Category</th>
<th>Matched</th>
<th>Mismatched</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVE/PASSIVE</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>ANTO</td>
<td>17</td>
<td>20</td>
</tr>
<tr>
<td>BELIEF</td>
<td>66</td>
<td>58</td>
</tr>
<tr>
<td>CONDITIONAL</td>
<td>23</td>
<td>26</td>
</tr>
<tr>
<td>COREF</td>
<td>30</td>
<td>29</td>
</tr>
<tr>
<td>LONG_SENTENCE</td>
<td>99</td>
<td>109</td>
</tr>
<tr>
<td>MODAL</td>
<td>144</td>
<td>126</td>
</tr>
<tr>
<td>NEGATION</td>
<td>129</td>
<td>104</td>
</tr>
<tr>
<td>PARAPHRASE</td>
<td>25</td>
<td>37</td>
</tr>
<tr>
<td>QUANTIFIER</td>
<td>125</td>
<td>140</td>
</tr>
<tr>
<td>QUANTITY/TIME_REASONING</td>
<td>15</td>
<td>39</td>
</tr>
<tr>
<td>TENSE_DIFFERENCE</td>
<td>51</td>
<td>18</td>
</tr>
<tr>
<td>WORD_OVERLAP</td>
<td>28</td>
<td>37</td>
</tr>
</tbody>
</table>

| Total                           | 767     | 753        |
Progress on MultiNLI

MultiNLI leaderboard: Systems over time

Human: 92.6
<table>
<thead>
<tr>
<th>Premise</th>
<th>Relation</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>A turtle danced.</td>
<td>entails</td>
<td>A turtle moved.</td>
</tr>
<tr>
<td>Every reptile danced.</td>
<td>neutral</td>
<td>A turtle ate.</td>
</tr>
<tr>
<td>Some turtles walk.</td>
<td>contradicts</td>
<td>No turtles move.</td>
</tr>
</tbody>
</table>
NLI adversarial testing

<table>
<thead>
<tr>
<th>Premise</th>
<th>Relation</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train A little girl kneeling in the dirt crying.</td>
<td>entails</td>
<td>A little girl is very sad.</td>
</tr>
<tr>
<td>Adversarial</td>
<td>entails</td>
<td>A little girl is very unhappy.</td>
</tr>
</tbody>
</table>

Glockner et al. 2018
NLI adversarial testing

<table>
<thead>
<tr>
<th>Premise</th>
<th>Relation</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>A woman is pulling a child on a sled in the snow.</td>
<td>entails</td>
<td>A child is sitting on a sled in the snow.</td>
</tr>
<tr>
<td>A child is pulling a woman on a sled in the snow.</td>
<td>neutral</td>
<td></td>
</tr>
</tbody>
</table>

Train

Adversarial

Nie et al. 2019a
Adversarial NLI dataset (ANLI)

1. Nie et al. 2019b

2. 162,865 labeled examples

3. The premises come from diverse sources.

4. The hypotheses are written by crowdworkers with the explicit goal of fooling state-of-the-art models.

5. This effort is a direct response to the results and findings for SNLI and MultiNLI that we just reviewed.
ANLI dataset creation

1. The annotator is presented with a premise sentence and a condition (entailment, contradiction, neutral).

2. The annotator writes a hypothesis.

3. A state-of-the-art model makes a prediction about the premise–hypothesis pair.

4. If the model’s prediction matches the condition, the annotator returns to step 2 to try again.

5. If the model was fooled, the premise–hypothesis pair is independently validated by other annotators.
## Additional ANLI details

<table>
<thead>
<tr>
<th>Round</th>
<th>Model</th>
<th>Training data</th>
<th>Context sources</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>BERT-large</td>
<td>SNLI + MultiNLI</td>
<td>Wikipedia</td>
<td>16,946</td>
</tr>
<tr>
<td></td>
<td>(Devlin et al. 2019)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>ROBERTa</td>
<td>SNLI + MultiNLI + NLI-FEVER + R1</td>
<td>Wikipedia</td>
<td>45,460</td>
</tr>
<tr>
<td></td>
<td>(Liu et al. 2019)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>ROBERTa</td>
<td>SNLI + MultiNLI + NLI-FEVER + R2</td>
<td>Various</td>
<td>100,459</td>
</tr>
<tr>
<td></td>
<td>(Liu et al. 2019)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>162,865</strong></td>
</tr>
</tbody>
</table>

- The train sets mix cases where the model’s predictions were correct and incorrect. The majority of the model predictions are correct, though.
- The dev and test sets contain only cases where the model’s prediction was incorrect.
### Code snippets: Readers and Example objects

[1]:
```python
import nli, os
```

[2]:
```python
SNLI_HOME = os.path.join("data", "nli_data", "snli_1.0")
MULTINLI_HOME = os.path.join("data", "nli_data", "multinli_1.0")
ANLI_HOME = os.path.join("data", "nli_data", "anli_v1.0")
```

[3]:
```python
snli_train_reader = nli.SNLITrainReader(SNLI_HOME, samp_percentage=0.10)
```

[4]:
```python
snli_dev_reader = nli.SNLIDevReader(SNLI_HOME, samp_percentage=0.10)
```

[5]:
```python
multi_train_reader = nli.MultiNLITrainReader(SNLI_HOME, samp_percentage=0.10)
```

[6]:
```python
multi_matched_dev_reader = nli.MultiNLIMatchedDevReader(SNLI_HOME)
```

[7]:
```python
multi_mismatched_dev_reader = nli.MultiNLIMismatchedDevReader(SNLI_HOME)
```

[8]:
```python
anli_train_reader = nli.ANLITrainReader(ANLI_HOME, rounds=(1, 2, 3))
```

[9]:
```python
anli_dev_reader = nli.ANLIDevReader(ANLI_HOME, rounds=(1, 2, 3))
```
Code snippets: Examples

[10]: `snli_iterator = iter(nli.SNLITrainReader(SNLI_HOME).read())`

[11]: `snli_ex = next(snli_iterator)`

[12]: `snli_ex.sentence1`

[12]: 'A person on a horse jumps over a broken down airplane.'

[13]: `snli_ex.sentence2`

[13]: 'A person is training his horse for a competition.'

[14]: `snli_ex.gold_label`

[14]: 'neutral'
Code snippets: Examples

[15]: snli_ex.sentence1_binary_parse
Code snippets: Examples

[16]: `snli_ex.sentencl_parse`

[16]:

```
(Re), bMHBn2tXb2Mi2M+2RnT`b2
(Re),
( ),
( ),
j
```

```
(ROOT
  (S
    (NP
      (NP
        (DT A)
        (NN person)
      )
      (IN on)
      (NP
        (DT DT)
        (NN NN)
        (NN a horse)
      )
    )
    (VP
      (VBZ jumps)
      (IN over)
      (NP
        (DT DT)
        (JJ JJ)
        (JJ JJ)
        (NN NN)
        (NN a broken down airplane)
      )
    )
  )
)
```
Code snippets: MultiNLI annotations

```python
[1]: import nli, os

[2]: ANN_HOME = os.path.join("data", "nlidata", "multinli_1.0_annotations")
MULTINLI_HOME = os.path.join("data", "nlidata", "multinli_1.0")

[3]: matched_filename = os.path.join(
    ANN_HOME, "multinli_1.0_matched_annotations.txt")
mismatched_filename = os.path.join(
    ANN_HOME, "multinli_1.0_mismatched_annotations.txt")

[4]: matched_ann = nli.read_annotated_subset(matched_filename, MULTINLI_HOME)

[5]: pair_id = '116176e'
    ann_ex = matched_ann[pair_id]
    print("pairID: {}".format(pair_id))
    print(ann_ex['annotations'])
    ex = ann_ex['example']
    print(ex['sentence1'])
    print(ex['gold_label'])
    print(ex['sentence2'])

pairID: 116176e
['#MODAL', '#COREF']
Students of human misery can savor its underlying sadness and futility.
entailment
Those who study human misery will savor the sadness and futility.
```
Hand-built features

1. Overview
2. SNLI, MultiNLI, and Adversarial NLI
3. Hand-built features
4. nli.experiment
5. Sentence-encoding models
6. Chained models
7. Attention
8. Error analysis
Word overlap and word-cross product

[1]:
```python
from collections import Counter
from itertools import product
import nli
from nltk.tree import Tree
import os
```

[2]:
```python
def word_overlap_phi(t1, t2):
    overlap = set([w1 for w1 in t1.leaves() if w1 in t2.leaves()])
    return Counter(overlap)
```

[3]:
```python
def word_cross_product_phi(t1, t2):
    return Counter([(w1, w2) for w1, w2 in product(t1.leaves(), t2.leaves())])
```

[4]:
```python
t1 = Tree.fromstring("""(S (NP Tobi) (VP (V is) (NP (D a) (N dog))))""")
```

[5]:
```python
t2 = Tree.fromstring("""(S (NP Tobi) (VP (V is) (NP (D a) (NP (A big) (N dog))))))"""")
```
Word overlap and word-cross product

In [6]: display(t1, t2)

Out[7]: Counter({'Tobi': 1, 'dog': 1, 'is': 1, 'a': 1})

In [8]: word_cross_product_phi(t1, t2)

Out[8]: Counter({('Tobi', 'Tobi'): 1, ('Tobi', 'is'): 1, ('Tobi', 'a'): 1, ('Tobi', 'big'): 1, ('Tobi', 'dog'): 1, ('is', 'Tobi'): 1, ('is', 'is'): 1, ('is', 'a'): 1, ('is', 'big'): 1, ('is', 'dog'): 1, ('a', 'Tobi'): 1, ('a', 'is'): 1, ('a', 'a'): 1, ('a', 'big'): 1, ('a', 'dog'): 1, ('dog', 'Tobi'): 1, ('dog', 'is'): 1, ('dog', 'a'): 1, ('dog', 'big'): 1, ('dog', 'dog'): 1})
WordNet features

[1]:
```
from collections import Counter
from itertools import product
from nltk.corpus import wordnet as wn
from nltk.tree import Tree
```

[2]:
```
puppies = wn.synsets('puppy')
[h for ss in puppies for h in ss.hypernyms()]
```

[2]:
```
[Synset('dog.n.01'), Synset('pup.n.01'), Synset('young_person.n.01')]
```

[3]:
```
# A more conservative approach uses just the first-listed
# Synset, which should be the most frequent sense:
wn.synsets('puppy')[0].hypernyms()
```

[3]:
```
[Synset('dog.n.01'), Synset('pup.n.01')]
```

[4]:
```
def wordnet_features(t1, t2, methodname):
    pairs = []
    words1 = t1.leaves()
    words2 = t2.leaves()
    for w1, w2 in product(words1, words2):
        hyps = [h for ss in wn.synsets(w1) for h in getattr(ss, methodname)()]  
        syns = wn.synsets(w2)
        if set(hyps) & set(syns):
            pairs.append((w1, w2))
    return Counter(pairs)
```

[5]:
```
def hypernym_features(t1, t2):
    return wordnet_features(t1, t2, 'hypernyms')
```

[6]:
```
def hyponym_features(t1, t2):
    return wordnet_features(t1, t2, 'hyponyms')
```
WordNet features

```
In [7]: t1 = Tree.fromstring("(S (NP (D the) (N puppy)) (VP moved))")
In [8]: t2 = Tree.fromstring("(S (NP (D the) (N dog)) (VP danced))")

In [9]: display(t1, t2)
```

```
In [10]: hypernym_features(t1, t2)
Out[10]: Counter({('puppy', 'dog'): 1})

In [11]: hyponym_features(t1, t2)
Out[11]: Counter({('moved', 'danced'): 1})
```
Other hand-built features

1. Additional WordNet relations

2. Edit distance

3. Word differences (cf. word overlap)

4. Alignment-based features

5. Negation

6. Quantifier relations (e.g., every $\sqsubseteq$ some; see MacCartney and Manning 2009)

7. Named entity features
1. Overview
2. SNLI, MultiNLI, and Adversarial NLI
3. Hand-built features
4. nli.experiment
5. Sentence-encoding models
6. Chained models
7. Attention
8. Error analysis
Complete experiment with nli.experiment

```python
[1]: from collections import Counter
    import nli
    import os
    from sklearn.linear_model import LogisticRegression
    import utils

[2]:  SNLI_HOME = os.path.join("data", "nli-data", "snli_1.0")

[3]:  def word_overlap_phi(t1, t2):
      overlap = set([w1 for w1 in t1.leaves() if w1 in t2.leaves()])
      return Counter(overlap)

[4]:  def fit_softmax(X, y):
      mod = LogisticRegression(solver='liblinear', multi_class='auto')
      mod.fit(X, y)
      return mod

[5]:  train_reader_10 = nli.SNLITrainReader(SNLI_HOME, samp_percentage=0.10)

[6]:  basic_experiment = nli.experiment(
    train_reader_10,
    word_overlap_phi,
    fit_softmax,
    assess_reader=None, # Default
    train_size=0.7, # Default
    score_func=utils.safe_macro_f1, # Default
    vectorize=True, # Default
    verbose=True, # Default
    random_state=None) # Default
```
Hyperparameter selection on train subsets

[1]:
```python
from collections import Counter
import nli
import os
from sklearn.linear_model import LogisticRegression
import utils
```

[2]:
```python
SNLI_HOME = os.path.join("data", "nldata", "snli_1.0")
```

[3]:
```python
def word_overlap_phi(t1, t2):
    overlap = set([w1 for w1 in t1.leaves() if w1 in t2.leaves()])
    return Counter(overlap)
```
Hyperparameter selection on train subsets

[1]:
```python
from collections import Counter
import nli
import os
from sklearn.linear_model import LogisticRegression
import utils
```

[2]:
```python
SNLI_HOME = os.path.join("data", "nlidata", "snli_1.0")
```

[3]:
```python
def word_overlap_phi(t1, t2):
    overlap = set([w1 for w1 in t1.leaves() if w1 in t2.leaves()])
    return Counter(overlap)
```

[4]:
```python
def fit_softmax_with_crossvalidation(X, y):
    basemod = LogisticRegression(
        fit_intercept=True, solver='liblinear', multi_class='auto')
    param_grid = {'C': [0.6, 0.7, 0.8, 1.0, 1.1], 'penalty': ['l1', 'l2']}
    best_mod = utils.fit_classifier_with_crossvalidation(
        X, y, basemod, cv=3, param_grid=param_grid)
    return best_mod
```

[5]:
```python
# Select hyperparameters based on a subset of the data:
  tuning_experiment_sample = nli_experiment(
      nli.SNLITrainReader(SNLI_HOME, samp_percentage=0.10),
      word_overlap_phi,
      fit_softmax_with_crossvalidation)
```

Best params: {'C': 1.0, 'penalty': 'l2'}
Best score: 0.413
Hyperparameter selection on train subsets

```
[1]: from collections import Counter
    import nli
    import os
    from sklearn.linear_model import LogisticRegression
    import utils

[2]: SNLI_HOME = os.path.join("data", "nlidata", "snli_1.0")

[3]: def word_overlap_phi(t1, t2):
    overlap = set([w1 for w1 in t1.leaves() if w1 in t2.leaves()])
    return Counter(overlap)

[6]: def fit_softmax_classifier_with_preselected_params(X, y):
    mod = LogisticRegression(
        fit_intercept=True, solver='liblinear', multi_class='auto',
        C=1.0, penalty='l2')
    mod.fit(X, y)
    return mod

[7]: # Use the selected hyperparamters in a (costly) full dataset training run:
    full_experiment = nli.experiment(
        nli.SNLITrainReader(SNLI_HOME),
        word_overlap_phi,
        fit_softmax_classifier_with_preselected_params,
        assess_reader=nli.SNLIDevReader(SNLI_HOME))
```
Hyperparameter selection with a few iterations

```python
[8]: def fit_softmax_with_crossvalidation_small_iter(X, y):
    basemod = LogisticRegression(
        fit_intercept=True, solver='liblinear', multi_class='auto',
        max_iter=3)
    param_grid = {'C': [0.6, 0.7, 0.8, 1.0, 1.1], 'penalty': ['l1', 'l2']}
    best_mod = utils.fit_classifier_with_crossvalidation(
        X, y, basemod, cv=3, param_grid=param_grid)
    return best_mod

[9]: # Select hyperparameters based on a few iterations:
    tuning_experiment_small_iter = nli.experiment(
        nli.SNLITrainReader(SNLI_HOME),
        word_overlap_phi,
        fit_softmax_with_crossvalidation_small_iter)

.../base.py:922: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.

Best params: {'C': 1.0, 'penalty': 'l1'}
Best score: 0.425
A hypothesis-only experiment

```python
[1]: from collections import Counter
    import os
    from sklearn.linear_model import LogisticRegression
    import nli, utils

[2]: SNLI_HOME = os.path.join("data", "nldata", "snli_1.0")

[3]: def hypothesis_only_unigrams_phi(t1, t2):
    return Counter(t2.leaves())

[4]: def fit_softmax(X, y):
    mod = LogisticRegression(solver='liblinear', multi_class='auto')
    mod.fit(X, y)
    return mod

[5]: hypothesis_only_experiment = nli.experiment(
    nli.SNLITrainReader(SNLI_HOME),
    hypothesis_only_unigrams_phi,
    fit_softmax,
    assess_reader=nli.SNLIDevReader(SNLI_HOME))

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>contradiction</td>
<td>0.654</td>
<td>0.631</td>
<td>0.642</td>
<td>3278</td>
</tr>
<tr>
<td>entailment</td>
<td>0.639</td>
<td>0.715</td>
<td>0.675</td>
<td>3329</td>
</tr>
<tr>
<td>neutral</td>
<td>0.670</td>
<td>0.613</td>
<td>0.640</td>
<td>3235</td>
</tr>
<tr>
<td>accuracy</td>
<td></td>
<td></td>
<td>0.653</td>
<td>9842</td>
</tr>
<tr>
<td>macro avg</td>
<td>0.655</td>
<td>0.653</td>
<td>0.653</td>
<td>9842</td>
</tr>
<tr>
<td>weighted avg</td>
<td>0.654</td>
<td>0.653</td>
<td>0.653</td>
<td>9842</td>
</tr>
</tbody>
</table>
```
A hypothesis-only experiment

```python
[6]: from sklearn.dummy import DummyClassifier

[7]: def fit_dummy_classifier(X, y):
    mod = DummyClassifier(strategy='stratified')
    mod.fit(X, y)
    return mod

[8]: random_experiment = nli.experiment(
    nli.SNLITrainReader(SNLI_HOME),
    lambda t1, t2: {'constant': 1},  # `DummyClassifier` ignores this!
    fit_dummy_classifier,
    assess_reader=nli.SNLIDevReader(SNLI_HOME))

<table>
<thead>
<tr>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>contradiction</td>
<td>0.333</td>
<td>0.333</td>
<td>0.333</td>
</tr>
<tr>
<td>entailment</td>
<td>0.343</td>
<td>0.339</td>
<td>0.341</td>
</tr>
<tr>
<td>neutral</td>
<td>0.332</td>
<td>0.335</td>
<td>0.333</td>
</tr>
</tbody>
</table>

| accuracy | 0.336 | 9842 |
| macro avg | 0.336 | 9842 |
| weighted avg | 0.336 | 9842 |
Sentence-encoding models

1. Overview
2. SNLI, MultiNLI, and Adversarial NLI
3. Hand-built features
4. nli.experiment
5. Sentence-encoding models
6. Chained models
7. Attention
8. Error analysis
Distributed representations as features

A classifier of some kind (learned)

e.g., concatenation, difference (not learned)

e.g., sum, average, etc. (not learned)

Embedding look-up

\[ y \]

\[ x \]

\[ x_p \]

\[ x_h \]

\[ x_3 \]

\[ x_2 \]

\[ x_1 \]

\[ x_3 \]

\[ x_5 \]

\[ x_4 \]

\[ \text{every} \]

\[ \text{dog} \]

\[ \text{danced} \]

\[ \text{every} \]

\[ \text{poodle} \]

\[ \text{moved} \]
**Code: Distributed representations as features**

```python
[1]: import numpy as np
    import os
    from sklearn.linear_model import LogisticRegression
    import nli, utils

[2]: SNLI_HOME = os.path.join("data", "nlidata", "snli_1.0")
    GLOVE_HOME = os.path.join('data', 'glove.6B')

[3]: glove_lookup = utils.glove2dict(
    os.path.join(GLOVE_HOME, 'glove.6B.50d.txt'))

[4]: def _get_tree_vecs(tree, lookup, np_func):
    allvecs = np.array([lookup[w] for w in tree.leaves() if w in lookup])
    if len(allvecs) == 0:
        dim = len(next(iter(lookup.values())))
        feats = np.zeros(dim)
    else:
        feats = np_func(allvecs, axis=0)
    return feats

[5]: def glove_leaves_phi(t1, t2, np_func=np.sum):
    prem_vecs = _get_tree_vecs(t1, glove_lookup, np_func)
    hyp_vecs = _get_tree_vecs(t2, glove_lookup, np_func)
    return np.concatenate((prem_vecs, hyp_vecs))

[6]: def glove_leaves_sum_phi(t1, t2):
    return glove_leaves_phi(t1, t2, np_func=np.sum)
```
Code: Distributed representations as features

```python
[7]: def fit_softmax(X, y):
    mod = LogisticRegression(
        fit_intercept=True, solver='liblinear', multi_class='auto')
    mod.fit(X, y)
    return mod

[8]: glove_sum_experiment = nli.experiment(
    nli.SNLITrainReader(SNLI_HOME),
    glove_leaves_sum_phi,
    fit_softmax,
    assess_reader=nli.SNLIDevReader(SNLI_HOME),
    vectorize=False) # We already have vectors!
```
Rationale for sentence-encoding models

1. Encoding the premise and hypothesis separately might give the model a chance to find rich abstract relationships between them.

2. Sentence-level encoding could facilitate transfer to other tasks (Dagan et al.’s (2006) vision).
Sentence-encoding RNNs
PyTorch strategy: Sentence-encoding RNNs

The full implementation is in nli_02_models.ipynb.

TorchRNNSentenceEncoderDataset
This is conceptually a list of pairs of sequences, each with their lengths, and a label vector:

\[
\left( \text{[every, dog, danced]}, \text{[every, poodle, moved]} \right), (3, 3), \text{entailment}
\]

TorchRNNSentenceEncoderClassifierModel
This is conceptually a premise RNN and a hypothesis RNN. The forward method uses them to process the two parts of the example, concatenate the outputs of those passes, and feed them into a classifier.

TorchRNNSentenceEncoderClassifier
This is basically unchanged from its super class TorchRNNClassifier, except the predict_proba method needs to deal with the new example format.
Sentence-encoding TreeNNs

\[ y \leftarrow \text{combo}(p_B, p_D) \]

\[ p_B = f([p_A; x_1]W + b) \]

\[ p_A = f([x_3; x_4]W + b) \]

\[ x_1 \]

\[ x_2 \]

\[ \text{danced} \]

\[ x_3 \]

\[ \text{every} \]

\[ x_4 \]

\[ x_5 \]

\[ \text{poodle} \]

\[ p_D = f([p_C; x_4]W + b) \]

\[ p_C = f([x_3; x_5]W + b) \]

Leaf nodes are looked up in the embedding.

Likely to be concatenation
Chained models

1. Overview
2. SNLI, MultiNLI, and Adversarial NLI
3. Hand-built features
4. nli.experiment
5. Sentence-encoding models
6. Chained models
7. Attention
8. Error analysis
Simple RNN

Recurrent architectures: simple classifiers

\[
\begin{align*}
W_{hh} & \quad W_{hh} & \quad W_{hh} & \quad W_{hh} & \quad W_{hh} & \quad W_{hh} \\
W_{xh} & \quad W_{xh} & \quad W_{xh} & \quad W_{xh} & \quad W_{xh} & \quad W_{xh} \\
\end{align*}
\]

\[
\begin{align*}
h_0 & \quad h_1 & \quad h_2 & \quad h_3 & \quad h_4 & \quad h_5 & \quad h_6 \\
x_3 & \quad x_2 & \quad x_1 & \quad x_3 & \quad x_5 & \quad x_4 \\
every & \quad dog & \quad danced & \quad every & \quad poodle & \quad moved \\
\end{align*}
\]
Rationale for sentence-encoding models

1. The premise truly establishes the context for the hypothesis.

2. Might be seen as corresponding to a real processing model.
Code snippet: Simple RNN

```python
[1]: import os
from torch_rnn_classifier import TorchRNNClassifier
import nli, utils

[2]: SNLI_HOME = os.path.join("data", "nlidata", "snli_1.0")

[3]: def simple_chained_rep_rnn_phi(t1, t2):
    return t1.leaves() + ["[SEP]"] + t2.leaves()

[4]: def fit_simple_chained_rnn(X, y):
    vocab = utils.get_vocab(X, n_words=10000)
    vocab.append("[SEP]")
    mod = TorchRNNClassifier(vocab, hidden_dim=50, max_iter=50)
    mod.fit(X, y)
    return mod

[5]: simple_chained_rnn_experiment = nli.experiment(
    nli.SNLITrainReader(SNLI_HOME, samp_percentage=0.10),
    simple_chained_rep_rnn_phi,
    fit_simple_chained_rnn,
    vectorize=False)
```
Premise and hypothesis RNNs

The PyTorch implementation strategy is similar to the one outlined earlier for sentence-encoding RNNs, except the final hidden state of the premise RNN becomes the initial hidden state for the hypothesis RNN.
Other strategies

TorchRNNClassifier

- TorchRNNClassifier feeds its final hidden state directly to the classifier layer.
- If bidirectional=True, then the two final states are concatenated and fed directly to the classifier layer.

Other ideas

- *Pool* all the hidden states with max or mean.
- Different pooling options can be combined.
- Additional layers between the hidden representation (however defined) and the classifier layer.
Attention

1. Overview
2. SNLI, MultiNLI, and Adversarial NLI
3. Hand-built features
4. nli.experiment
5. Sentence-encoding models
6. Chained models
7. **Attention**
8. Error analysis
Guiding ideas

1. We need more connections between premise and hypothesis.

2. In processing the hypothesis, the model needs “reminders” of what the premise contained; the final premise hidden state isn’t enough.

3. Soft alignment between premise and hypothesis – a neural interpretation of an old idea in NLI.
Global attention

\[
\begin{align*}
&h_1 \quad h_2 \quad h_3 \\
&x_3 \quad x_2 \quad x_1 \\
&\text{every} \quad \text{dog} \quad \text{danced}
\end{align*}
\]
Global attention

\[ \tilde{\alpha} = \begin{bmatrix} h_C^T h_1 & h_C^T h_2 & h_C^T h_3 \end{bmatrix} \]

scores

\[ h_1 \quad h_2 \quad h_3 \quad h_A \quad h_B \quad h_C \]

\[ x_3 \quad x_2 \quad x_1 \quad x_{27} \quad x_{21} \quad x_{11} \]

every
dog
danced
some
poodle
danced
Global attention

attention weights \( \alpha = \text{softmax}(\tilde{\alpha}) \)

scores \( \tilde{\alpha} = \begin{bmatrix} h^T_C h_1 & h^T_C h_2 & h^T_C h_3 \end{bmatrix} \)

\( x_1 \) danced every dog danced some poodle

\( x_2 \) dog

\( x_3 \) every

\( x_{11} \) danced

\( x_{21} \) poodle

\( x_{27} \) some

\( h_1 \)

\( h_2 \)

\( h_3 \)

\( h_A \)

\( h_B \)

\( h_C \)
Global attention

context \( \kappa = \text{mean}(\alpha_1h_1, \alpha_2h_2, \alpha_3h_3) \)

attention weights \( \alpha = \text{softmax}(\tilde{\alpha}) \)

scores \( \tilde{\alpha} = \begin{bmatrix} h^C_1 \ h^T_1 \ h^T_2 \ h^T_3 \end{bmatrix} \)
Global attention

attention combo
\[ \tilde{h} = \tanh([\kappa; h_C]W_\kappa) \]

context
\[ \kappa = \text{mean}(\alpha_1 h_1, \alpha_2 h_2, \alpha_3 h_3) \]

attention weights
\[ \alpha = \text{softmax}(\tilde{\alpha}) \]

scores
\[ \tilde{\alpha} = \begin{bmatrix} h_C^T h_1 & h_C^T h_2 & h_C^T h_3 \end{bmatrix} \]
Global attention

attention combo \( \tilde{h} = \tanh([\kappa; h_C]W_k) \) or \( \tilde{h} = \tanh(\kappa W_k + h_C W_h) \)

context \( \kappa = \text{mean}(\alpha_1 h_1, \alpha_2 h_2, \alpha_3 h_3) \)

attention weights \( \alpha = \text{softmax}(\tilde{\alpha}) \)

scores \( \tilde{\alpha} = \begin{bmatrix} h_C^T h_1 & h_C^T h_2 & h_C^T h_3 \end{bmatrix} \)
Global attention

classifier  \[ y = \text{softmax}(\tilde{h}W + b) \]

attention combo  \[ \tilde{h} = \tanh(\kappa; h_C)W_k \]

context  \[ \kappa = \text{mean}(\alpha_1 h_1, \alpha_2 h_2, \alpha_3 h_3) \]

attention weights  \[ \alpha = \text{softmax}(\tilde{\alpha}) \]

scores  \[ \tilde{\alpha} = \begin{bmatrix} h_C^T h_1 & h_C^T h_2 & h_C^T h_3 \end{bmatrix} \]
Global attention

\[
y = \text{softmax}(\tilde{h}W + b)
\]

\[
\tilde{h} = \text{tanh}(\begin{bmatrix} 0.07, 0.11, 0.1, 0.2 \end{bmatrix} W_p)
\]

\[
p = \text{mean}(\begin{bmatrix} 0.35 \cdot [0.4, 0.6], 0.33 \cdot [0.2, 0.4], 0.31 \cdot [0.1, 0.1] \end{bmatrix})
\]

\[
g = \begin{bmatrix} 0.35, 0.33, 0.31 \end{bmatrix}
\]

\[
\tilde{g} = \begin{bmatrix} 0.16, 0.10, 0.03 \end{bmatrix}
\]

\[
\text{danced every dog danced some poodle}
\]

\[
\begin{bmatrix} 0.4, 0.6 \end{bmatrix}, \begin{bmatrix} 0.2, 0.4 \end{bmatrix}, \begin{bmatrix} 0.1, 0.1 \end{bmatrix}, \begin{bmatrix} 0.1, 0.2 \end{bmatrix}
\]
Global attention

\[ \alpha = [0.16, 0.10, 0.03] \]
Global attention

attention weights \( \alpha = [0.35, 0.33, 0.31] \)

scores \( \tilde{\alpha} = [0.16, 0.10, 0.03] \)
Global attention

context $\kappa = \text{mean}(0.35 \cdot [0.4, 0.6], 0.33 \cdot [0.2, 0.4], 0.31 \cdot [0.1, 0.1])$

attention weights $\alpha = [0.35, 0.33, 0.31]$

scores $\tilde{\alpha} = [0.16, 0.10, 0.03]$
Global attention

attention combo \( \tilde{h} = \tanh([0.07, 0.11, 0.1, 0.2]W_k) \)

dance combo

context \( \kappa = \text{mean}(0.35 \cdot [0.4, 0.6], 0.33 \cdot [0.2, 0.4], 0.31 \cdot [0.1, 0.1]) \)

dance context

attention weights \( \alpha = [0.35, 0.33, 0.31] \)

dance attention weights

scores \( \tilde{\alpha} = [0.16, 0.10, 0.03] \)

dance scores

\[ [0.4, 0.6] \quad [0.2, 0.4] \quad [0.1, 0.1] \quad [0.4, 0.6] \quad [0.2, 0.4] \quad [0.1, 0.1] \]
Global attention

classifier \[ y = \text{softmax}(\tilde{h}W + b) \]

attention combo \[ \tilde{h} = \tanh([0.07, 0.11, 0.1, 0.2]W_k) \]

context \[ \kappa = \text{mean}(.35 \cdot [.4, .6], .33 \cdot [.2, .4], .31 \cdot [.1, .1]) \]

attention weights \[ \alpha = [0.35, 0.33, 0.31] \]

scores \[ \tilde{\alpha} = [0.16, 0.10, 0.03] \]
Other scoring functions (Luong et al. 2015)

\[
\text{score}(h_C, h_i) = \begin{cases} 
  h_C^T h_i & \text{dot} \\
  h_C^T W_\alpha h_i & \text{general} \\
  W_\alpha [h_C; h_i] & \text{concat}
\end{cases}
\]
Word-by-word attention
Word-by-word attention

\[
M = \tanh \left( \begin{bmatrix}
0.4 & 0.6 \\
0.2 & 0.4 \\
0.1 & 0.1 \\
\end{bmatrix} + \begin{bmatrix}
0.2 & 0.3 \\
0.2 & 0.3 \\
0.2 & 0.3 \\
\end{bmatrix} \begin{bmatrix}
K_A \\
K_A \\
K_A \\
\end{bmatrix} W_h \right)
\]
Word-by-word attention

weights at $B$ \[ \alpha_B = \text{softmax}(MW) \]

\[ M = \text{tanh} \begin{bmatrix} 0.4 & 0.6 \\ 0.2 & 0.4 \\ 0.1 & 0.1 \end{bmatrix} + \begin{bmatrix} 0.2 & 0.3 \\ 0.2 & 0.3 \\ 0.2 & 0.3 \end{bmatrix} W_h \]
Word-by-word attention

context at $B$ \quad $\kappa_B = \begin{bmatrix} 0.4 & 0.6 \\ 0.2 & 0.4 \\ 0.1 & 0.1 \end{bmatrix} \alpha_B + \tanh(\kappa_A W_\alpha)$

weights at $B$ \quad $\alpha_B = \text{softmax}(Mw)$

$M = \tanh\left( \begin{bmatrix} 0.4 & 0.6 \\ 0.2 & 0.4 \\ 0.1 & 0.1 \end{bmatrix} + \begin{bmatrix} 0.2 & 0.3 \\ 0.2 & 0.3 \\ 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} \kappa_A \\ \kappa_A \\ \kappa_A \end{bmatrix} W_h \right)$

$[0.4, 0.6]$  \hspace{1cm} $[0.2, 0.4]$  \hspace{1cm} $[0.1, 0.1]$  \hspace{1cm} $A$  \hspace{1cm} $B$  \hspace{1cm} $C$  

$\begin{array}{l} x_3 \\ x_2 \\ x_1 \end{array}$  \hspace{1cm}  \hspace{1cm}  \hspace{1cm}  \hspace{1cm}  \hspace{1cm}  \hspace{1cm}  \hspace{1cm}  \hspace{1cm} $x_{27}$  \hspace{1cm}  \hspace{1cm}  \hspace{1cm} $x_{21}$  \hspace{1cm}  \hspace{1cm}  \hspace{1cm} $x_{11}$

\begin{array}{l} \text{every} \\ \text{dog} \\ \text{danced} \end{array}$  \hspace{1cm}  \hspace{1cm}  \hspace{1cm}  \hspace{1cm}  \hspace{1cm}  \hspace{1cm}  \hspace{1cm}  \hspace{1cm} $\text{some}$  \hspace{1cm} $\text{poodle}$  \hspace{1cm} $\text{danced}$
**Word-by-word attention**

**classifier input**

\[ \tilde{h} = \tanh([\kappa_C; h_C]W_K) \]

**context at B**

\[ \kappa_B = \begin{bmatrix} 0.4 & 0.6 \\ 0.2 & 0.4 \\ 0.1 & 0.1 \end{bmatrix} \alpha_B + \tanh(\kappa_A W_\alpha) \]

**weights at B**

\[ \alpha_B = \text{softmax}(Mw) \]

\[ M = \tanh \left( \begin{bmatrix} 0.4 & 0.6 \\ 0.2 & 0.4 \\ 0.1 & 0.1 \end{bmatrix} + \begin{bmatrix} 0.2 & 0.3 \\ 0.2 & 0.3 \\ 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} K_A \\ K_A \end{bmatrix} W_h \right) \]
Connection with the Transformer

\[ a_{\text{attn}} + a_{\text{input}} \]
\[ b_{\text{attn}} + b_{\text{input}} \]
\[ c_{\text{attn}} + c_{\text{input}} \]

\[ x_{47} \quad \frac{1}{p_1} \quad \text{The} \]
\[ x_{30} \quad \frac{2}{p_2} \quad \text{Rock} \]
\[ x_{34} \quad \frac{3}{p_3} \quad \text{rules} \]

\[ c_{\text{attn}} = \text{sum}(\alpha_{1} a_{\text{input}}, \alpha_{2} b_{\text{input}}) \]
\[ \alpha = \text{softmax}(\tilde{\alpha}) \]
\[ \tilde{\alpha} = \left[ \frac{c_{\text{input}}^T a_{\text{input}}}{\sqrt{d_k}}, \frac{c_{\text{input}}^T b_{\text{input}}}{\sqrt{d_k}} \right] \]

\[ c_{\text{input}} = x_{34} + p_3 \]

Vaswani et al. 2017
Other variants

- Local attention (Luong et al. 2015) builds connections between selected points in the premise and hypothesis.

- Word-by-word attention can be set up in many ways, with many more learned parameters than my simple example. A pioneering instance for NLI is Rocktäschel et al. 2016.

- The attention representation at time $t$ could be appended to the hidden representation at $t + 1$ (Luong et al. 2015).

- Memory networks (Weston et al. 2015) can be used to address similar issues related to properly recalling past experiences.
Error analyses

1. Overview
2. SNLI, MultiNLI, and Adversarial NLI
3. Hand-built features
4. nli.experiment
5. Sentence-encoding models
6. Chained models
7. Attention
8. Error analysis
## Systems compared

<table>
<thead>
<tr>
<th>Model</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Regression</td>
<td>cross-product</td>
</tr>
<tr>
<td>Chained LSTM with deep classifier</td>
<td>GloVe</td>
</tr>
<tr>
<td>Fine-tuned ROBERTa</td>
<td>ROBERTa-large</td>
</tr>
</tbody>
</table>
The Logistic Regression implementation

```python
[1]: from collections import Counter
    from itertools import product
    import os
    from sklearn.linear_model import LogisticRegression
    import nli, utils

[2]: MULTINLI_HOME = os.path.join("data", "nlidata", "multinli_1.0")

[3]: def word_cross_product_phi(t1, t2):
    return Counter([(w1, w2) for w1, w2 in product(t1.leaves(), t2.leaves())])

[4]: def fit_softmax(X, y):
    mod = LogisticRegression(solver='liblinear', multi_class='auto')
    mod.fit(X, y)
    return mod

[5]: train_reader = nli.MultiNLITrainReader(MULTINLI_HOME)

[6]: dev_reader = nli.MultiNLIMatchedDevReader(MULTINLI_HOME)

[7]: experiment = nli.experiment(
    train_reader,
    word_cross_product_phi,
    fit_softmax,
    assess_reader=dev_reader,
    verbose=True)
```
The Chained LSTM implementation

```
[1]: import os
    import torch.nn as nn
    from torch_rnn_classifier import TorchRNNClassifier, TorchRNNClassifierModel
    import nli, utils

[2]: class DeepRNNClassifierModel(TorchRNNClassifierModel):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        drop_prob = 0.1
        self.dropout = nn.Dropout(p=drop_prob)
        self.relu = nn.ReLU()
        self.bidirectional = kwargs['bidirectional']
        self.hidden_dim = kwargs['hidden_dim']
        if self.bidirectional:
            classifier_dim = self.hidden_dim * 2
        else:
            classifier_dim = self.hidden_dim
        self.mlp_layer = nn.Linear(classifier_dim, classifier_dim)

    def forward(self, X, seq_lengths):
        state = self.rnn_forward(X, seq_lengths, self.rnn)
        h = self.relu(self.mlp_layer(state))
        h = self.dropout(h)
        logits = self.classifier_layer(h)
        return logits

class DeepRNNClassifier(TorchRNNClassifier):
    def build_graph(self):
        return DeepRNNClassifierModel(
            vocab_size=len(self.vocab),
            embedding=self.embedding,
            use_embedding=self.use_embedding,
            embed_dim=self.embed_dim,
            hidden_dim=self.hidden_dim,
            output_dim=self.n_classes_,
            bidirectional=self.bidirectional,
            device=self.device)
```

Inspired by the BiLSTM of Williams et al. 2018
The Chained LSTM implementation

```python
[3]:
    def fix_random_seeds(

[4]:
    GLOVE_HOME = os.path.join("data", 'glove.6B')
    MULTINLI_HOME = os.path.join("data", "nliData", "multinli_1.0")

[5]:
    SEP = "[SEP]"

[6]:
    def chained_rnn_phi(t1, t2):
        return t1.leaves() + [SEP] + t2.leaves()

[7]:
    glove_lookup = utils.glove2dict(os.path.join(GLOVE_HOME, 'glove.840B.300d.txt'))

[8]:
    def fit_deep_rnn(X, y):
        vocab = utils.get_vocab(X)
        glove_embedding, glove_vocabulary = utils.create_pretrained_embedding(
            glove_lookup, vocab, required_tokens=('UNK', SEP))
        mod = DeepRNNClassifier(
            glove_vocabulary,
            embedding=glove_embedding,
            embed_dim=300,
            hidden_dim=300,
            bidirectional=True,
            max_iter=8,
            eta=0.0004,
            l2_strength=0.00001,
            batch_size=16,
            warm_start=True)
        mod.fit(X, y)
        return mod

[9]:
    train_reader = nli.MultiNLITrainReader(MULTINLI_HOME)

[10]:
    dev_reader = nli.MultiNLIMatchedDevReader(MULTINLI_HOME)

[11]:
    basic_experiment = nli.experiment(
        train_reader,
        chained_rnn_phi,
        fit_deep_rnn,
        assess_reader=dev_reader,
        vectorize=False,
        verbose=True)
```

Inspired by the BiLSTM of Williams et al. 2018
The ROBERTa implementation

```
[1]: import nli, os
    import torch
    from sklearn.metrics import classification_report

[2]: MULTINLI_HOME = os.path.join("data", "nlidata", "multinli_1.0")

[3]: model = torch.hub.load('pytorch/fairseq', 'roberta.large.mnli')
    _ = model.eval()
    Using cache found in /Users/cgpotts/.cache/torch/hub/pytorch_fairseq_master

[4]: dev = [((ex.sentence1, ex.sentence2), ex.gold_label)
        for ex in nli.MultiNLIMatchedDevReader(MULTINLI_HOME).read()]

[5]: X_dev_str, y_dev = zip(*dev)

[6]: X_dev = [model.encode(*ex) for ex in X_dev_str]

[7]: %time pred_indices = [model.predict('mnli', ex).argmax() for ex in X_dev]
    CPU times: user 1h 45min 44s, sys: 3min 44s, total: 1h 49min 28s
    Wall time: 27min 23s

[8]: to_str = {0: 'contradiction', 1: 'neutral', 2: 'entailment'}

[9]: preds = [to_str[c.item()] for c in pred_indices]

[10]: print(classification_report(y_dev, preds))

https://github.com/pytorch/fairseq/tree/master/examples/roberta
```
Dev-set score comparisons

<table>
<thead>
<tr>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>Macro-F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic regression</td>
<td>59.2</td>
<td>59.0</td>
<td>59.1</td>
</tr>
<tr>
<td>Chained LSTM with deep classifier</td>
<td>68.1</td>
<td>67.1</td>
<td>67.3</td>
</tr>
<tr>
<td>Fine-tuned ROBERTa</td>
<td>90.5</td>
<td>90.5</td>
<td>90.5</td>
</tr>
</tbody>
</table>
## MultiNLI annotations

<table>
<thead>
<tr>
<th>Annotations</th>
<th>Premise</th>
<th>Relation</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>#MODAL, #COREF</td>
<td>Students of human misery can savor its underlying sadness and futility.</td>
<td>entailment</td>
<td>Those who study human misery will savor the sadness and futility.</td>
</tr>
<tr>
<td>#NEGATION, #TENSE_DIFFERENCE, #CONDITIONAL</td>
<td>oh really it wouldn’t matter if we plant them when it was starting to get warmer</td>
<td>contradiction</td>
<td>It is better to plant when it is colder.</td>
</tr>
<tr>
<td>#QUANTIFIER, #ACTIVE/PASSIVE</td>
<td>They consolidated programs to increase efficiency and deploy resources more effectively</td>
<td>entailment</td>
<td>Programs to increase efficiency were consolidated.</td>
</tr>
</tbody>
</table>
Annotation-set score comparisons

<table>
<thead>
<tr>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>Macro-F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic regression</td>
<td>58.5</td>
<td>58.0</td>
<td>58.0</td>
</tr>
<tr>
<td>Chained LSTM with deep classifier</td>
<td>69.3</td>
<td>68.3</td>
<td>68.4</td>
</tr>
<tr>
<td>Fine-tuned ROBERTa</td>
<td>91.9</td>
<td>91.9</td>
<td>91.9</td>
</tr>
</tbody>
</table>
MultiNLI annotations: Results by category

All results
MultiNLI annotations: Results by category

Most challenging categories

#CONDITIONAL

#QUANTITY/TIME_REASONING

#WORD_OVERLAP

Examples

Examples

Examples
Testing for specific patterns

Does your model know that negation is downward monotone?

Fido moved.       Fido didn’t move.
↑  ↓
Fido ran.         Fido didn’t run.

Does your model know that every is downward monotone on its first argument and upward monotone on its second?

Every dog moved.

Every puppy moved. Every dog ran.

Does your model systematically capture such patterns?
Testing with adversarial test sets
Testing with adversarial test sets

One-word changes to SNLI hypotheses using structured resources; labels separately validated by crowdworkers.

<table>
<thead>
<tr>
<th>Premise</th>
<th>Relation</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>A little girl kneeling in the dirt crying.</td>
<td>entails</td>
<td>A little girl is very sad.</td>
</tr>
<tr>
<td>A little girl is very sad.</td>
<td>entails</td>
<td>A little girl is very unhappy.</td>
</tr>
</tbody>
</table>

Glockner et al. 2018
Testing with adversarial test sets

One-word changes to SNLI hypotheses using structured resources; labels separately validated by crowdworkers.

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>antonyms</td>
<td>1147</td>
</tr>
<tr>
<td>synonyms</td>
<td>894</td>
</tr>
<tr>
<td>cardinals</td>
<td>759</td>
</tr>
<tr>
<td>nationalities</td>
<td>755</td>
</tr>
<tr>
<td>drinks</td>
<td>731</td>
</tr>
<tr>
<td>antonyms_wordnet</td>
<td>706</td>
</tr>
<tr>
<td>colors</td>
<td>699</td>
</tr>
<tr>
<td>ordinals</td>
<td>663</td>
</tr>
<tr>
<td>countries</td>
<td>613</td>
</tr>
<tr>
<td>rooms</td>
<td>595</td>
</tr>
<tr>
<td>materials</td>
<td>397</td>
</tr>
<tr>
<td>vegetables</td>
<td>109</td>
</tr>
<tr>
<td>instruments</td>
<td>65</td>
</tr>
<tr>
<td>planets</td>
<td>60</td>
</tr>
</tbody>
</table>

Glockner et al. 2018
Testing with adversarial test sets

One-word changes to SNLI hypotheses using structured resources; labels separately validated by crowdworkers.

<table>
<thead>
<tr>
<th>Model</th>
<th>Train set</th>
<th>SNLI test set</th>
<th>New test set</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposable Attention</td>
<td>SNLI</td>
<td>84.7%</td>
<td>51.9%</td>
<td>-32.8</td>
</tr>
<tr>
<td>(Parikh et al., 2016)</td>
<td>MultiNLI + SNLI</td>
<td>84.9%</td>
<td>65.8%</td>
<td>-19.1</td>
</tr>
<tr>
<td></td>
<td>SciTail + SNLI</td>
<td>85.0%</td>
<td>49.0%</td>
<td>-36.0</td>
</tr>
<tr>
<td>ESIM (Chen et al., 2017)</td>
<td>SNLI</td>
<td>87.9%</td>
<td>65.6%</td>
<td>-22.3</td>
</tr>
<tr>
<td></td>
<td>MultiNLI + SNLI</td>
<td>86.3%</td>
<td>74.9%</td>
<td>-11.4</td>
</tr>
<tr>
<td></td>
<td>SciTail + SNLI</td>
<td>88.3%</td>
<td>67.7%</td>
<td>-20.6</td>
</tr>
<tr>
<td>Residual-Stacked-Encoder</td>
<td>SNLI</td>
<td>86.0%</td>
<td>62.2%</td>
<td>-23.8</td>
</tr>
<tr>
<td>(Nie and Bansal, 2017)</td>
<td>MultiNLI + SNLI</td>
<td>84.6%</td>
<td>68.2%</td>
<td>-16.8</td>
</tr>
<tr>
<td></td>
<td>SciTail + SNLI</td>
<td>85.0%</td>
<td>60.1%</td>
<td>-24.9</td>
</tr>
<tr>
<td>WordNet Baseline</td>
<td>-</td>
<td>-</td>
<td>85.8%</td>
<td>-</td>
</tr>
<tr>
<td>KIM (Chen et al., 2018)</td>
<td>SNLI</td>
<td>88.6%</td>
<td>83.5%</td>
<td>-5.1</td>
</tr>
</tbody>
</table>

Table 3: Accuracy of various models trained on SNLI or a union of SNLI with another dataset (MultiNLI, SciTail), and tested on the original SNLI test set and the new test set.

Glockner et al. 2018
Testing with adversarial test sets

```python
[1]: import nli, os, torch
    from sklearn.metrics import classification_report

[2]: # Available from https://github.com/BIU-NLP/Breaking_NLI:
    breaking_nli_src_filename = os.path.join("../new-data/data/dataset.jsonl")
    reader = nli.NLIRender(breaking_nli_src_filename)

[3]: exs = [((ex.sentence1, ex.sentence2), ex.gold_label) for ex in reader.read()]

[4]: X_test_str, y_test = zip(*exs)

[5]: model = torch.hub.load('pytorch/fairseq', 'roberta.large.mnli')
    _ = model.eval()

    Using cache found in /Users/cgpotts/.cache/torch/hub/pytorch_fairseq_master

[6]: X_test = [model.encode(*ex) for ex in X_test_str]

[7]: pred_indices = [model.predict('mnli', ex).argmax() for ex in X_test]

[8]: to_str = {0: 'contradiction', 1: 'neutral', 2: 'entailment'}

[9]: preds = [to_str[c.item()] for c in pred_indices]

https://github.com/pytorch/fairseq/tree/master/examples/roberta
Testing with adversarial test sets

```
[10]: print(classification_report(y_test, preds))

    precision    recall  f1-score   support

contradiction    0.99    0.97    0.98      7164
entailment      0.86    1.00    0.92      982
neutral         0.15    0.15    0.15       47

accuracy        0.97
macro avg       0.67    0.71    0.68      8193
weighted avg   0.97    0.97    0.97      8193
```

https://github.com/pytorch/fairseq/tree/master/examples/roberta
References


Yoav Goldberg. 2015. A primer on neural network models for natural language processing. Ms., Bar Ilan University.


References


Christopher D. Manning. 2006. Local textual inference: It’s hard to circumscribe, but you know it when you see it – and NLP needs it. Ms., Stanford University.


Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2019b. Adversarial NLI: A new benchmark for natural language understanding. UNC CHapel Hill and Facebook AI Research.


References III


