Natural Language Inference

Christopher Potts

Stanford Linguistics

CS 224U: Natural language understanding
April 29 and May 1
Overview

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

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

2. Homework 4 and bake-off 4: hw4_wordentail.ipynb

3. Core readings: Bowman et al. 2015a; Rocktäschel et al. 2016

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

<table>
<thead>
<tr>
<th>Premise</th>
<th>Relation</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>turtle</td>
<td>contradicts</td>
<td>linguist</td>
</tr>
<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>
<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 $\equiv$ paraphrase</td>
</tr>
<tr>
<td>Summarization</td>
<td>text $\sqsubset$ summary</td>
</tr>
<tr>
<td>Information retrieval</td>
<td>query $\sqsupset$ document</td>
</tr>
<tr>
<td>Question answering</td>
<td>question $\sqsupset$ answer</td>
</tr>
<tr>
<td></td>
<td>$Who$ left? $\Rightarrow$ $Someone$ left</td>
</tr>
<tr>
<td></td>
<td>$Someone$ left $\sqsupset$ Sandy left</td>
</tr>
</tbody>
</table>
Models for NLI

- **Logic and theorem proving** (Bos & Markert 2005)
- **Natural Logic**
- **Semantic graphs**
- **Clever hand-built features**
- **Deep learning (2015)**
- **N-gram variations**

Depth of representations:
- Deep, brittle

Effectiveness:
- Robust, shallow

A standard baseline, often very robust!

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

- **Bos & Markert 2005**: Logic and theorem proving
- **MacCartney 2009**: Natural Logic
- **Hickl et al. 2006; de Marneffe et al. 2006**: Semantic graphs
- **Deep learning (2019)**: Clever hand-built features
- **N-gram variations**

*Depth of representations vs. Effectiveness* 

- Deep, brittle: Standard baseline, often very robust!
- Robust, shallow: See the Excitement Open Platform (Pado et al. 2012)
Other NLI datasets

- The FraCaS textual inference test suite
  https://nlp.stanford.edu/~wcmac/downloads/

- SemEval 2013
  https://www.cs.york.ac.uk/semeval-2013/

- SemEval 2014: Sentences Involving Compositional Knowledge (SICK)

- MedNLI (derived from MIMIC III)
  https://physionet.org/physiotools/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/

- Related: 30M Factoid Question-Answer Corpus
  http://agarciaduran.org/

- Related: The Penn Paraphrase Database
  http://paraphrase.org/

- The GLUE benchmark (diverse tasks including NLI)
  https://gluebenchmark.com
## 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></td>
</tr>
<tr>
<td><strong>turtle</strong> linguist</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Yes</strong> entailment</td>
<td><strong>Yes</strong> entailment</td>
<td><strong>P ≡ Q</strong> equivalence</td>
</tr>
<tr>
<td><strong>non-entailment</strong></td>
<td><strong>Unknown</strong> non-entailment</td>
<td><strong>P ⊏ Q</strong> forward</td>
</tr>
<tr>
<td><strong>No</strong></td>
<td><strong>No</strong> contradiction</td>
<td><strong>P ⊐ Q</strong> reverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>P # Q</strong> non-entailment</td>
</tr>
</tbody>
</table>
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)

- Why does it hold? We can trace this partly to artificial biases in the texts people create, but part of the effect is the result of the way semantic spaces are organized:
  - 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.
SNLI and MultiNLI

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

1. Bowman et al. 2015a
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 a 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>

### Examples

- **Premise:** A man inspects the uniform of a figure in some East Asian country.
  - **Relation:** contradiction
  - **Hypothesis:** The man is sleeping.

- **Premise:** An older and younger man smiling.
  - **Relation:** neutral
  - **Hypothesis:** Two men are smiling and laughing at the cats playing on the floor.

- **Premise:** A black race car starts up in front of a crowd of people.
  - **Relation:** contradiction
  - **Hypothesis:** A man is driving down a lonely road.

- **Premise:** A soccer game with multiple males playing.
  - **Relation:** entailment
  - **Hypothesis:** Some men are playing a sport.

- **Premise:** A smiling costumed woman is holding an umbrella.
  - **Relation:** neutral
  - **Hypothesis:** A happy woman in a fairy costume holds an umbrella.
### 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.
**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/
## 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>NEGANATION</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>
<tr>
<td><strong>Total</strong></td>
<td>767</td>
<td>753</td>
</tr>
</tbody>
</table>
**Overview**

SNLI and MultiNLI

Hand-built features

nli.experiment

Sentence-encoding

Chained

Attention

Error analysis

---

**Code snippets: Readers and Example objects**

```python
In [1]: import nli
   ...:
   ...: import os

In [2]: SNLI_HOME = os.path.join("data", "nlidata", "snli_1.0")
   ...:
   ...: MULTINLI_HOME = os.path.join("data", "nlidata", "multinli_1.0")

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

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

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

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

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

In [8]: snli_iterator = iter(nli.SNLITrainReader(SNLI_HOME).read())

In [9]: snli_ex = next(snli_iterator)

In [10]: print(snli_ex)

A person on a horse jumps over a broken down airplane.
neutral

A person is training his horse for a competition.
```
Code snippets: Readers and Example objects

In [11]: `snli_ex.sentence1`
Out[11]: 'A person on a horse jumps over a broken down airplane.'

In [12]: `snli_ex.sentence2`
Out[12]: 'A person is training his horse for a competition.'

In [13]: `snli_ex.gold_label`
Out[13]: 'neutral'

In [14]: `snli_ex.sentence1_binary_parse`
Out[14]:

```
X
/   \
X   X
/     \
X
/ \
X
/ \
A person on
a horse
```

In [15]: `snli_ex.sentence1_parse`
Out[15]:

```
X
/   \
X   X
/     \
X
/ \
X
/ \
jumps
over
a
broken
don
```

```
X
/   \
X   X
/     \
X
/ \
X
down airplane
```
Code snippets: Readers and Example objects

In [11]: snli_ex.sentence1
Out[11]: 'A person on a horse jumps over a broken down airplane.'

In [12]: snli_ex.sentence2
Out[12]: 'A person is training his horse for a competition.'

In [13]: snli_ex.gold_label
Out[13]: 'neutral'

In [15]: snli_ex.sentence1_parse
Out[15]:

```
  S
    |ROOT
    |  |
    |  VP
    |   |
    |   NP
    |    |
    |    NP
    |     |
    |     OP
    |      |
    |      NP
    |       |
    |       DT
    |        |
    |        NN
    |         |
    |         A
    |          |
    |          person
    |           on
    |            DT
    |             NN
    |              a
    |               horse
    |                over
    |                 DT
    |                  JJ
    |                   JJ
    |                    NN
    |                     a
    |                      broken
    |                       down
    |                        airplane
```

In [14]: snli_ex.sentence1_binary_parse
Out[14]:

In [15]: snli_ex.sentence1_parse
Out[15]:

2
Code snippets: MultiNLI annotations

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

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

In [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")

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

In [5]: len(matched_ann)

Out[5]: 495

In [6]: 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 and MultiNLI
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

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

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

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

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

In [5]: 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, '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

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

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

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

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

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

In [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)

In [5]: def hypernym_features(t1, t2):
    return wordnet_features(t1, t2, 'hyponyms')

In [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 ⊆ some; see MacCartney & Manning 2009)
7. Named entity features
Combining dense and sparse representations

- short and dense
- long and sparse
Combining dense and sparse representations

- short and dense
- short and dense

Model external transformation, or with learned parameters

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

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

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

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

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

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

In [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

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

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

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

In [4]: 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

In [5]: # Select hyperparameters based on a subset of the data:
    tuning_experiment = 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
```
In [1]: from collections import Counter
   import nli
   import os
   from sklearn.linear_model import LogisticRegression
   import utils

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

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

In [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

In [7]: # Use the selected hyperparameters 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.SNЛИDevReader(SNLI_HOME))
Hyperparameter selection with a few iterations

In [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

In [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

In [1]: from collections import Counter
   import nli
   import os
   from sklearn.dummy import DummyClassifier
   from sklearn.linear_model import LogisticRegression
   import utils

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

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

In [4]: 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

In [5]: hypothesis_only_experiment = nli.experiment(
   nli.SNLITrainReader(SNLI_HOME),
   hypothesis_only_unigrams_phi,
   fit_softmax_classifier_with_preselected_params,
   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>micro avg</td>
<td>0.653</td>
<td>0.653</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
In [6]: def fit_dummy_classifier(X, y):
    mod = DummyClassifier(strategy='stratified')
    mod.fit(X, y)
    return mod

In [7]: 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></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>contradiction</td>
<td>0.336</td>
<td>0.338</td>
<td>0.337</td>
<td>3278</td>
</tr>
<tr>
<td>entailment</td>
<td>0.336</td>
<td>0.330</td>
<td>0.333</td>
<td>3329</td>
</tr>
<tr>
<td>neutral</td>
<td>0.331</td>
<td>0.335</td>
<td>0.333</td>
<td>3235</td>
</tr>
<tr>
<td>micro avg</td>
<td>0.334</td>
<td>0.334</td>
<td>0.334</td>
<td>9842</td>
</tr>
<tr>
<td>macro avg</td>
<td>0.334</td>
<td>0.334</td>
<td>0.334</td>
<td>9842</td>
</tr>
<tr>
<td>weighted avg</td>
<td>0.334</td>
<td>0.334</td>
<td>0.334</td>
<td>9842</td>
</tr>
</tbody>
</table>
```
A premise-only experiment

In [8]: def premise_only_unigrams_phi(t1, t2):
   ...:     return Counter(t1.leaves())

In [9]: premise_only_experiment = nli.experiment(
   ...:     nli.SNLITrainReader(SNLI_HOME),
   ...:     premise_only_unigrams_phi,
   ...:     fit_softmax_classifier_with_preselected_params,
   ...:     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>0.337</td>
<td>0.255</td>
<td>0.290</td>
<td>3278</td>
</tr>
<tr>
<td>entailment</td>
<td>0.340</td>
<td>0.388</td>
<td>0.363</td>
<td>3329</td>
</tr>
<tr>
<td>neutral</td>
<td>0.330</td>
<td>0.364</td>
<td>0.346</td>
<td>3235</td>
</tr>
<tr>
<td>micro avg</td>
<td>0.336</td>
<td>0.336</td>
<td>0.336</td>
<td>9842</td>
</tr>
<tr>
<td>macro avg</td>
<td>0.336</td>
<td>0.336</td>
<td>0.333</td>
<td>9842</td>
</tr>
<tr>
<td>weighted avg</td>
<td>0.336</td>
<td>0.336</td>
<td>0.333</td>
<td>9842</td>
</tr>
</tbody>
</table>

2• A result of the data collection method: each premise is paired with one hypothesis from each class.
2• The logistic regression premise-only baseline for the word-entailment bake-off is $\approx 0.47$, vs. $\approx 0.50$ for hypothesis-only.
A premise-only experiment

In [8]: def premise_only_unigrams_phi(t1, t2):
   return Counter(t1.leaves())

In [9]: premise_only_experiment = nli.experiment(
   nli.SNLITrainReader(SNLI_HOME),
   premise_only_unigrams_phi,
   fit_softmax_classifier_with_preselected_params,
   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>0.337</td>
<td>0.255</td>
<td>0.290</td>
<td>3278</td>
</tr>
<tr>
<td>entailment</td>
<td>0.340</td>
<td>0.388</td>
<td>0.363</td>
<td>3329</td>
</tr>
<tr>
<td>neutral</td>
<td>0.330</td>
<td>0.364</td>
<td>0.346</td>
<td>3235</td>
</tr>
<tr>
<td>micro avg</td>
<td>0.336</td>
<td>0.336</td>
<td>0.336</td>
<td>9842</td>
</tr>
<tr>
<td>macro avg</td>
<td>0.336</td>
<td>0.336</td>
<td>0.333</td>
<td>9842</td>
</tr>
<tr>
<td>weighted avg</td>
<td>0.336</td>
<td>0.336</td>
<td>0.333</td>
<td>9842</td>
</tr>
</tbody>
</table>

- A result of the data collection method: each premise is paired with one hypothesis from each class.
- The logistic regression premise-only baseline for the word-entailment bake-off is $\approx 0.47$, vs. $\approx 0.50$ for hypothesis-only.
Sentence-encoding models

1. Overview
2. SNLI and MultiNLI
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)

\[ y \]

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

\[ x \]

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

Embedding look-up

\[ x_p \]

\[ x_h \]

\[ x_3 \]

\[ x_2 \]

\[ x_1 \]

\[ x_3 \]

\[ x_5 \]

\[ x_4 \]

every

dog

danced

every

poodle

moved
In [1]: import nli
   : import numpy as np
   : import os
   : from sklearn.linear_model import LogisticRegression
   : import utils

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

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

In [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

In [5]: def glove_leaves_sum_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))

In [6]: def glove_leaves_sum_phi(t1, t2):
   : return glove_leaves_phi(t1, t2, np_func=np.sum)

---

**Code: Distributed representations as features**

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

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

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

In [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

In [5]: def glove_leaves_sum_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))

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

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

In [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!

<table>
<thead>
<tr>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>contradiction</td>
<td>0.505</td>
<td>0.476</td>
<td>0.490</td>
</tr>
<tr>
<td>entailment</td>
<td>0.500</td>
<td>0.561</td>
<td>0.529</td>
</tr>
<tr>
<td>neutral</td>
<td>0.549</td>
<td>0.513</td>
<td>0.530</td>
</tr>
<tr>
<td>micro avg</td>
<td>0.517</td>
<td>0.517</td>
<td>0.517</td>
</tr>
<tr>
<td>macro avg</td>
<td>0.518</td>
<td>0.516</td>
<td>0.516</td>
</tr>
<tr>
<td>weighted avg</td>
<td>0.518</td>
<td>0.517</td>
<td>0.516</td>
</tr>
</tbody>
</table>
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

$h_3$ and $h_B$ should be good sentence representations.
**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:

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

**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

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

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

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

\[ x_1 \quad \text{danced} \]

\[ x_2 \quad \text{dog} \]

\[ x_3 \quad \text{every} \]

Leaf nodes are looked up in the embedding.
Chained models

1. Overview
2. SNLI and MultiNLI
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

- Sentence-encoding

- Chained

- Attention

- Error analysis
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**

In [1]: import nli
   
   import os
   from torch_rnn_classifier import TorchRNNClassifier
   import utils

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

In [3]: # Consider adding a fixed boundary symbol between premise and hypothesis.
   
   def simple_chained_rep_rnn_phi(t1, t2):
       return t1.leaves() + t2.leaves()

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

In [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)
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.
# Attention

1. Overview
2. SNLI and MultiNLI
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

danced every dog danced some poodle

\[
\begin{align*}
& x_3 & & h_1 & & x_2 & & h_2 & & x_1 & & h_3 & & h_A & & h_B & & h_C \\
& \text{every} & & & & \text{dog} & & & & \text{danced} & & & & \text{some} & & \text{poodle} & & \text{danced} \\
\end{align*}
\]
Global attention

scores

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

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

scores

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

attention weights

\[
g = \text{softmax}(\tilde{g})
\]

\[
\tilde{g} = h_1^T h_1, h_2^T h_2, h_3^T h_3
\]
Global attention

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

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

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

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

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

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

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

\begin{figure}[h]
\begin{center}
\begin{tikzpicture}
\node (h1) at (0,0) {$h_1$};
\node (h2) at (1,0) {$h_2$};
\node (h3) at (2,0) {$h_3$};
\node (hA) at (3,0) {$h_A$};
\node (hB) at (4,0) {$h_B$};
\node (hc) at (5,0) {$h_c$};
\node (x1) at (0,-1) {$x_1$};
\node (x2) at (1,-1) {$x_2$};
\node (x3) at (2,-1) {$x_3$};
\node (x27) at (3,-1) {$x_{27}$};
\node (x21) at (4,-1) {$x_{21}$};
\node (x11) at (5,-1) {$x_{11}$};
\node (every) at (0,-2) {every};
\node (dog) at (1,-2) {dog};
\node (danced) at (2,-2) {danced};
\node (some) at (3,-2) {some};
\node (poodle) at (4,-2) {poodle};
\node (danced) at (5,-2) {danced};
\draw (h1) -- (x1);
\draw (h2) -- (x2);
\draw (h3) -- (x3);
\draw (hA) -- (x27);
\draw (hB) -- (x21);
\draw (hc) -- (x11);
\end{tikzpicture}
\end{center}
\end{figure}
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

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

\[ \tilde{h} = \text{tanh}(\left[0.07, 0.11, 0.1, 0.2\right]W) \]

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

\[ g = \left[0.35, 0.33, 0.31\right] \]

\[ \tilde{g} = \left[0.16, 0.10, 0.03\right] \]
Global attention

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

attention weights \( \alpha = [0.35, 0.33, 0.31] \)
scores \( \tilde{\alpha} = [0.16, 0.10, 0.03] \)
Global attention

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

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

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

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

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:  \( \tilde{\alpha} = [0.16, 0.10, 0.03] \)
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**  \[ \kappa = \text{mean}([0.35 \cdot [0.4, 0.6], 0.33 \cdot [0.2, 0.4], 0.31 \cdot [0.1, 0.1]]) \]

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

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

```
[0.4, 0.6]  [0.2, 0.4]  [0.1, 0.1]  [0.1, 0.2]
  x_3   x_2   x_1   x_27   x_21   x_11
  every  dog   danced  some  poodle  danced
```


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

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

weights at $B$

$$\alpha_B = \text{softmax}(Mw)$$

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

context at $B$

$$\kappa_B = \begin{bmatrix} 0.4 & 0.6 \\ 0.2 & 0.4 \\ 0.1 & 0.1 \end{bmatrix} \alpha_B + \tanh (\kappa_A W_A)$$

weights at $B$

$$\alpha_B = \text{softmax}(Mw)$$

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

classifier input
\[ \tilde{h} = \tanh([\kappa_C; h_C]W_{\kappa}) \]

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
\[ \alpha_B = \text{softmax}(M w) \]

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

- Vaswani et al. (2017) use attention for their primary connections, a reversal of the usual pattern.

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

1. Overview
2. SNLI and MultiNLI
3. Hand-built features
4. nli.experiment
5. Sentence-encoding models
6. Chained models
7. Attention
8. Error analysis
## 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>
## Matched MultiNLI annotations

<table>
<thead>
<tr>
<th>Model</th>
<th>Features</th>
<th>Macro-F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic regression</td>
<td>cross-product</td>
<td>0.58</td>
</tr>
<tr>
<td>Chained LSTM</td>
<td>random embedding</td>
<td>0.55</td>
</tr>
<tr>
<td>Sentence-encoding LSTM</td>
<td>random embedding</td>
<td>0.51</td>
</tr>
</tbody>
</table>

- Logistic regression tuned hyperparameters: $C (0.1 \text{ to } 1.2 \text{ by } 0.1)$ and penalty (L1, L2). Model file is $\approx 600\text{MB}$; $\approx 16\text{M}$ features.
- LSTM tuned hyperparameters: `embed_dim (50, 100), hidden_dim (50, 100, 150), learning rate (0.001, 0.01, 0.05), and activation function (Tanh, ReLU)`. Model files are $\approx 1\text{MB}$ each.
MultiNLI annotations: LSTMs by category

All models more correct than incorrect
MultiNLI annotations: LSTMs by category

All models more incorrect than correct

![Bar chart showing the number of #QUANTITY/TIME_REASONING examples for different models (LogisticRegression, Chained, Encoding). The chart indicates that all models have more incorrect examples than correct examples.](chart.png)
MultiNLI annotations: LSTMs by category

Only Logistic Regression more correct than incorrect
MultiNLI annotations: LSTMs by category

Only chained LSTM more correct than incorrect

#WORD_OVERLAP

<table>
<thead>
<tr>
<th>Encoding</th>
<th>correct</th>
<th>incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogisticRegression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chained</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MultiNLI annotations: LSTMs by category

Only sentence-encoding LSTM more incorrect than correct

(There were no categories in which only the sentence-encoding LSTM was more correct than incorrect.)
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?
Probing with artificial data

Negation (after MacCartney & Manning 2007)

<table>
<thead>
<tr>
<th></th>
<th>not-(p), not-(q)</th>
<th>(p), not-(q)</th>
<th>not-(p), (q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p) disjoint (q)</td>
<td>neutral</td>
<td>subset</td>
<td>superset</td>
</tr>
<tr>
<td>(p) equal (q)</td>
<td>equal</td>
<td>disjoint</td>
<td>disjoint</td>
</tr>
<tr>
<td>(p) neutral (q)</td>
<td>neutral</td>
<td>neutral</td>
<td>neutral</td>
</tr>
<tr>
<td>(p) subset (q)</td>
<td>superset</td>
<td>disjoint</td>
<td>neutral</td>
</tr>
<tr>
<td>(p) superset (q)</td>
<td>subset</td>
<td>neutral</td>
<td>disjoint</td>
</tr>
</tbody>
</table>

The issue

If your model does perfectly on a doubly negated dataset, will its performance generalize to triply negated cases? This would be evidence that it had truly learned the algebra of negation. See Bowman et al. 2015b; Evans et al. 2018; Geiger et al. 2018.
References


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


