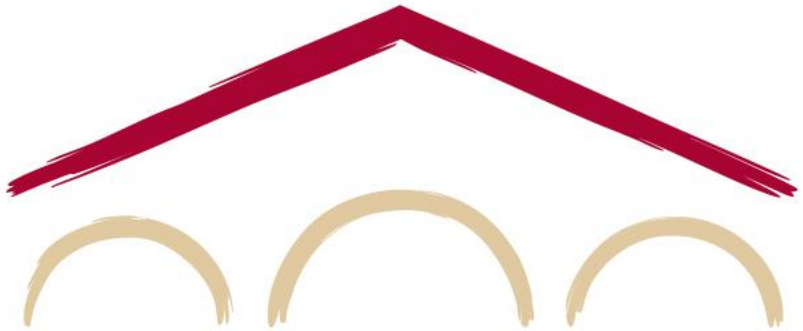
Assign/bakeoff 3 overview



Christopher Potts

CS224u: Natural Language Understanding

Homework and bakeoff: Compositional generalization

```
__author__ = "Christopher Potts and Zhengxuan Wu"
__version__ = "CS224u, Stanford, Spring 2023"
```

COGS: A Compositional Generalization Challenge Based on Semantic Interpretation

Najoung Kim
Johns Hopkins University
n.kim@jhu.edu

Tal Linzen
New York University
linzen@nyu.edu

ReCOGS: How Incidental Details of a Logical Form Overshadow an Evaluation of Semantic Interpretation

Zhengxuan Wu **Christopher D. Manning** **Christopher Potts**
Stanford University
{wuzhengx, manning, cgpotts}@stanford.edu

The ReCOGS task

Input: A rose was helped by a dog .

Output: rose (53) ; dog (38) ; help (7) AND theme (7 , 53) AND agent (7 , 38)

Input: The sailor dusted a boy .

Output: * sailor (48) ; boy (53) ; dust (10) AND agent (10 , 48) AND theme (10 , 53)

COGS and ReCOGS

COGS is the original. ReCOGS reworks COGS to focus on purely semantic phenomena (rather than incidental details of LFs).

Input: `The sailor saw Emma .`

ReCOGS: `* sailor ( 48 ) ; Emma ( 53 ) ; see ( 10 ) AND agent ( 10 , 48 ) AND theme ( 10 , 53 )`

COGS: `* sailor ( x _ 1 ) ; see . agent ( x _ 2 , x _ 1 ) AND see . theme ( x _ 2 , Emma )`

ReCOGS splits

- **Train**: 135,546 input/output pairs
- **Dev**: 3K input/output pairs like those in Train
- **Gen**: 21K examples in 21 categories – novel combinations of familiar elements

Gen split examples

Category	Train	Gen

subj_to_obj_proper	Lina gave the bottle to John.
	
	`Lina ( 1 ) ; John ( 7 ) ;`
	`* bottle ( 3 ) ;`
	`give ( 47 ) AND`
	`agent (47 , 1 ) AND`
	`theme ( 47 , 3 ) AND`
	`recipient ( 47 , 7 )`
		A cat rolled Lina.
		
		`Lina ( 3 ) ; cat( 45 ) ;`
		`roll( 9 ) AND agent ( 9 ,`
		`45) AND theme ( 9 , 3 )`

prim_to_subj	Bella	Bella baked the cake
cp_recursion	Emma said that Noah knew that the cat danced.	Emma said that Noah knew that Lucas saw that the cat danced.

Question 1: Proper names & their semantic roles

Task 1: Pattern-based analysis function

```python
import re

def get_propername_role(s):
    """Extract from `s` all the pairs `(name, role)` determined by
    binding relationships. There can be multiple tokens of the same
    name with different variables, as in "Kim ( 1 )" and "Kim ( 47 )",
    and there can be instances in which a single name with variable
    like "Kim ( 1 )" binds into multiple role expressions like
    "agent ( 4 , 1 )" and "theme ( 6 , 1 )". Your function should
    cover all these cases.

    We've suggested a particular program design to get you started,
    but you are free to do something different and perhaps cleverer
    if you wish!

    Parameters
    ----------
    s: str

    Returns
    -------
    set of tuples `(name, role)` where `name` and `role` are str
    """
```

Task 2: Finding challenging names

```python
from collections import defaultdict

def find_name_roles(split_df, colname="output"):
    """Create a map from names to dicts mapping roles to counts: the
    number of time the name appears with role in `split_df`:

    Parameters
    ----------
    split_df : pd.DataFrame
        Needs to have a column called `colname`.
    colname: str
        Column to target with `get_propername_role`. Default: "output".

    Returns
    -------
    `defaultdict` mapping names to roles to counts
    """
    # This is a convenient way to create a multidimensional count dict:
    # You can access it out of the box as `all_roles[key1][key2] += 1`.
    all_roles = defaultdict(lambda : defaultdict(int))
```

Spoilers: Charlie is only a theme in train, only an agent in gen;
Lina is only an agent in train, only a theme in gen

Modeling interlude

1. Hugging Face `PreTrainedTokenizerFast`
2. PyTorch `Dataset`
3. `EncoderDecoderModel.from_pretrained("ReCOGS/ReCOGS-model")`
4. `RecogsLoss(nn.Module)`
5. `RecogsModule(nn.Module)`
6. `RecogsModel(TorchModelBase)`

```
# Main interface. No need to worry
# about 1-5 if you are not training
# models for your original system.
```

Question 2: Exploring predictions

For this question, you just use the trained ReCOGS model to continue your analysis from Question 1.

```
def category_assess(gen_df, model, category):
    """Assess `model` against the `category` examples in `gen_df`.

    Parameters
    ----------
    gen_df: pd.DataFrame
        Should be `dataset["gen"]`
    model: A `RecogsModel instance
    category: str
        A string from `gen_df.category`

    Returns
    -------
    `pd.DataFrame` limited to `category` examples and with columns
    "prediction" and "correct" added by this function
    """
```

You will discover that the model struggles the most with proper names in unfamiliar positions.

A note about ReCOGS assessment

```
# The precise names of bound variables do not matter:

recogs_exact_match(
    "dog ( 4 ) AND happy ( 4 )",
    "dog ( 7 ) AND happy ( 7 ) ")
```

```
True
```

```
# The order of conjuncts does not matter:

recogs_exact_match(
    "dog ( 4 ) AND happy ( 4 )",
    "happy ( 7 ) AND dog ( 7 )")
```

```
True
```

```
# Consistency of variable names does matter:

recogs_exact_match(
    "dog ( 4 ) AND happy ( 4 )",
    "dog ( 4 ) AND happy ( 7 )")
```

```
False
```

Question 3: A basic in-context learning approach

```
Translate sentences into logical forms.

---

Follow the following format.

Input: ${the sentence to be translated}
Output: ${a logical form}

---

Input: A cake was painted by Mason .
Output: cake ( 30 ) ; Mason ( 40 ) ; paint ( 22 ) AND theme ( 22 , 30 ) AND agent ( 22 , 40 )

Input: The boy painted a rose .
Output: * boy ( 36 ) ; rose ( 20 ) ; paint ( 43 ) AND agent ( 43 , 36 ) AND theme ( 43 , 20 )

Input: A rose was helped by a dog .
Output:
```

```python
@dsp.transformation
def recogs_dsp(example, train=dsp_recogs_train, k=2):
    pass
    # Step 1: Sample k train cases and add them to the `demos`
    # attribute of `example`:
    ##### YOUR CODE HERE



    # Run your program using `cogs_template`:
    ##### YOUR CODE HERE



    # Return the `dsp.Completions`:
    ##### YOUR CODE HERE
```

Question 4: Original systems

For your original system, you can do anything at all. The only constraint:

> **You cannot train your system on any examples from `dataset["gen"]`, nor can the output representations from those examples be included in any prompts used for in-context learning.**

Original system ideas

- DSP program
- Further training of our model
- Using a pretrained model
- Training from scratch
- Symbolic solver?
- …

```
recogs_ff = RecogsModel(
    batch_size=5,
    gradient_accumulation_steps=20,
    max_iter=100,
    early_stopping=True,
    n_iter_no_change=10,
    optimizer_class=torch.optim.Adam,
    eta=0.00001)
```

```
_ = recogs_ff.fit(dataset['dev'].input[: 40], dataset['dev'].output[: 40])
```

```
import torch.nn as nn
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

class T5RecogsModule(nn.Module):
    def __init__(self):
        super().__init__()
        self.encdec = AutoModelForSeq2SeqLM.from_pretrained("t5-small")

    def forward(self, X_pad, X_mask, y_pad, y_mask, labels=None):
        outputs = self.encdec(
            input_ids=X_pad,
            attention_mask=X_mask,
            decoder_attention_mask=y_mask,
            labels=y_pad)
        return outputs

class T5RecogsModel(RecogsModel):
    def __init__(self, *args, initialize=True, **kwargs):
        super().__init__(*args, **kwargs)
        self.enc_tokenizer = AutoTokenizer.from_pretrained("t5-small")
        self.dec_tokenizer = self.enc_tokenizer

    def build_graph(self):
        return T5RecogsModule()
```

Bakeoff

```
bakeoff_df = pd.read_csv(
    os.path.join(SRC_DIRNAME, "cs224u-recogs-test-unlabeled.tsv"),
    sep="\t", index_col=0)
```

For the bakeoff entry, you should add a column "prediction" containing your predicted LFs and then use the following command to write the file to disk:

```
bakeoff_df.to_csv("cs224u-recogs-bakeoff-entry.tsv", sep="\t")
```

You cannot train your system on any examples from dataset["gen"], nor can the output representations from those examples be included in any prompts used for in-context learning.