ECE 5831 - Neural Networks and Pattern Recognition

Context Derivation for Knowledge Graph Expansion

Compiled and written by: Haard Rao, Rohit Sanjay, Neel Khakhar

Abstract

Humans can speak, read, and write with the aid of language, which is a form of communication. For instance, we use natural language—more specifically, words—to think, decide, plan, and do other things. The key question, though, is whether we can converse similarly with machines in the age of AI. In other words, is it possible for people to speak naturally to computers? Because computers require organized data but human speech is unstructured and frequently confusing in nature, it is difficult for us to create NLP applications.

This makes it possible to define Natural Language Processing (NLP) as the area of computer science, particularly Artificial Intelligence (AI), that deals with teaching computers how to comprehend and use human language. Technically speaking, the main function of NLP would be to program computers to process and analyze vast amounts of natural language data.

Introduction

Data from an information extraction task can be stored in a knowledge graph. A concept known as a "triple"—a group of three items—a subject, a predicate, and an object—that we might use to hold information about something—is used in many fundamental knowledge graph implementations.

OUR GOAL

02

What do we expect from the system?

WHAT DO WE EXPECT FROM SYSTEM?

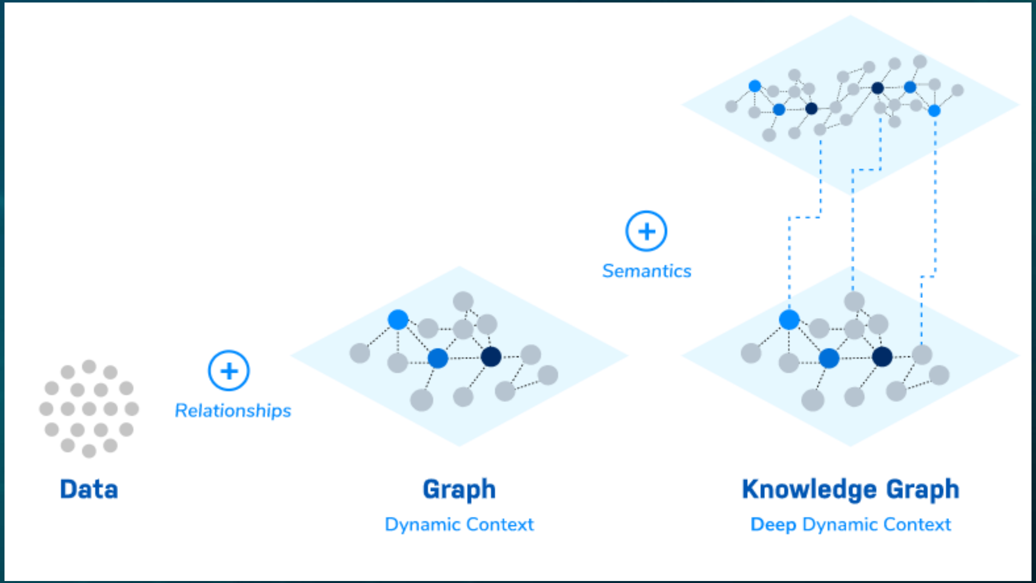
We expect our system to process the given dataset.

Extract entities, tokens and relations from sentences present in dataset.

To analyze all the sentences in biderectional form.

And give us a knowledge graph as an output.

WHAT DO WE EXPECT FROM SYSTEM?



PROJECT COMPONENTS

03

What do we use to
develop system?

WHAT DO WE USE TO DEVELOP SYSTEM?

A dataset refers to a collection of connected data. Datasets can contain standardized, relevant information that can be used for a variety of purposes. We have used wiki_sentence v2 as our dataset for sentences

DATASET

Bidirectional Encoded Representations from Transformers is referred to as BERT. BERT uses the surrounding text to provide context in order to help computers understand the meaning of ambiguous words in text.

BERT

SpaCy enables you to create applications that handle and "understand" massive amounts of text because it is made primarily for usage in production environments. Systems for information extraction or natural language understanding can be created using it.

spaCy

WHAT DO WE USE TO DEVELOP SYSTEM?

The most popular algorithms, including part-of-speech tagging, stemming, sentiment analysis, topic segmentation, and named entity recognition, are all included in SpaCy. SpaCy assists the computer with text analysis, preprocessing, and comprehension.

SpaCy

THE PAPER AND CODE

Target "Knowledge-driven Data Construction for Zero-shot Evaluation in Commonsense Question Answering"

while an individual knowledge graph is better suited for specific tasks, a global knowledge graph brings consistent gains across different tasks. In addition, both preserving the structure of the task as well as generating fair and informative questions help language models learn more effectively

Theory

question answering, machine translation, reading comprehension, and summarization; language models begin to learn these tasks without any explicit supervision when trained on a new dataset of millions of webpages ('WebText')

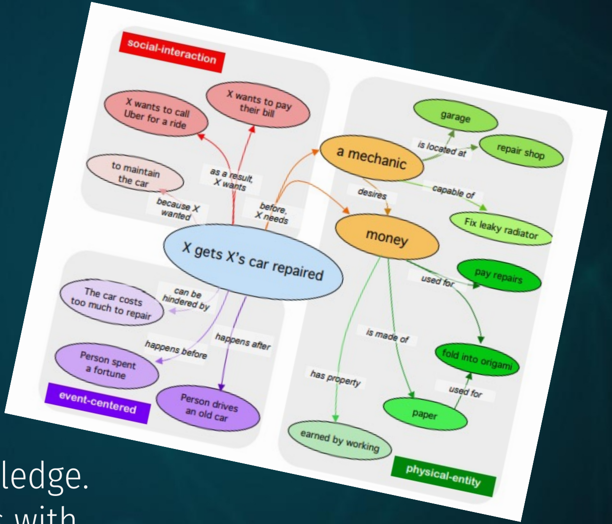
Transformer

An Atlas of Machine Commonsense for If-Then Reasoning:

Experimental results demonstrate that multitask models that incorporate the hierarchical structure of if-then relation types lead to more accurate inference compared to models trained in isolation, as measured by both automatic and human evaluation.

Knowledge

Implementation



Exploration of Zero-shot question answering using entity-relation based knowledge. ATOMIC focuses on inferential knowledge organized as typed if-then relations with variables (e.g., "if X pays Y a compliment, then Y will likely return the compliment"). We utilize this knowledge graph with Generative Pretrained Transformer 2 (GPT2).

The capacity of the language model is essential to the success of zero-shot task transfer and increasing it improves performance in a log-linear fashion across tasks. , GPT-2, is a 1.5B parameter Transformer that achieves state of the art results on 7 out of 8 tested language modeling datasets in a zero-shot setting but still underfits WebText.

ATOMIC Knowledge graph translation

elp generate_from_ATOMIC2020.py - comet-atomic-2020-master - Visual Studio Code

nain\src\Data_generation\data\atomic | dev_random.jsonl C:\...\NK-AKG-main\... | generate_from_ATOMIC.py 1 | generate_from_ATOMIC2020.py 1

C: > Users > neelk > Paper > trialS > NK-AKG-main > src > Data_generation > generate_from_ATOMIC2020.py > ATOMICProcessor > __init__

```python

class ATOMICProcessor(object):
    def __init__(self, args):
        self.mapping = {
        'xAttr' : '. PersonX is seen as',
        'xIntent' : '. Before, PersonX wanted',
        'xNeed' : '. Before, PersonX needed to',
        'xReact': '. As a result, PersonX felt',
        'xWant': '. As a result, PersonX wanted to',
        'xEffect': '. PersonX then',
        'oReact': '. As a result, others felt',
        'oWant': '. As a result, others wanted to',
        'oEffect': '. Others then',
        'isBefore': '. What happened before was',
        'isAfter': '. What happened after was',
        'AtLocation': ' can be found in',
        'CapableOf': '. It is capable of',
        'Causes': '. That caused',
        'Desires': '. Resulting in desire of',
        'HasProperty': '. It also contains',
        'HasSubEvent': '. But first need to',
        'HinderedBy': '. It can not happen because',
        'MadeUpOf': '. It is made up of',
        'NotDesires': '. It does not want to',
        'ObjectUse': '. It is used to',
        'xReason': '. PersonX did that because',
        'isFilledBy': ''
        }
        self.xset = ['PersonX', 'Personx', 'personX', 'personx', 'Person X', 'Person x', 'person X', 'person x']
        self.yset = ['PersonY', 'Persony', 'personY', 'persony', 'Person Y', 'Person y', 'person Y', 'person y']
        self.zset = ['PersonZ', 'Personz', 'personZ', 'personz', 'Person Z', 'Person z', 'person Z', 'person z']
        self.xset1 = [' X ', ' x ', ' X\'', ' x\'', ' X.', ' x.']
```

PROBLEMS 2 | OUTPUT | DEBUG CONSOLE | TERMINAL

The Marketplace has extensions that can help with '.jsonl'

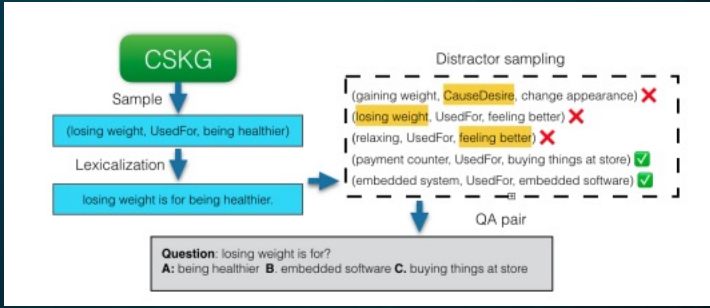
Search Marketplace | Don't Show Again f

{"id": "1", "dim": "xAttr", "context": "Cody plays a ___ in the war. Cody is seen as", "correct": 1, "candidates": ["eager.", "combative.", "informed."], "keywords": ["plays", "war"]}

Sample output

Negative Sample generation

```python
def negative_sample(self, prefix, dim, correct_ones, data, person_set, question, correct_answer):
    negatives = []
    curr_data = random.choices(data, k=self.downsample_size)
    distractors = list(set([neg for sample in curr_data for neg in sample[1][dim]]))
    distractors = [neg for neg in distractors if len(set(prefix).intersection(self.tail_keywords[(neg, dim)])) == 0]
    distractors_mapping = {i:self.tail_index[neg] for i, neg in enumerate(distractors)}
    distractors_indices = list(distractors_mapping.values())
    distractor_emb = self.embeddings[distractors_indices]
    correct_emb = self.embeddings[self.tail_index[correct_answer]]
    cos_scores = util.pytorch_cos_sim(correct_emb, distractor_emb)[0]
    high_prob = self.high_prob
    low_prob = self.low_prob
    midpoint = np.argwhere((cos_scores.numpy()>low_prob) & (cos_scores.numpy() < high_prob)).squeeze(1)
    midinf = 0
    while len(midpoint) < self.patience and midinf < self.patience:
        midinf += 1
        low_prob -= self.step_size
        midpoint = np.argwhere((cos_scores.numpy()>low_prob) & (cos_scores.numpy() < high_prob)).squeeze(1)
    if len(midpoint) == 0:
        print ('empty')
        return None
    infinite = 0
    while len(negatives) < 2 and infinite < self.patience:
        infinite += 1
        sample_idx = random.choice(midpoint)
        neg = self.reverse_tail_index[distractors_mapping[sample_idx.item()]]
        if neg in correct_ones:
            continue
        if neg in negatives:
            continue
        if neg[:-1] in correct_answer[:-1].split() or correct_answer[:-1] in neg[:-1].split():
            continue
        if len(person_set) < len(self.xset+self.xset1)*2 and any([y in neg for y in self.yset+self.yset1]):
            continue
        if len(person_set) < len(self.xset+self.xset1)*3 and any([z in neg for z in self.zset+self.zset1]):
            continue
        negatives.append(neg)
    self.lower_bounds[low_prob] += 1
    if len(negatives) < 2:
        return None
    return negatives
```



Output

IMPLEMENTATION

04

How do we do it?

HOW DO WE DO IT?

The implementation is based on three main parts

PROCESSING ON DATASET

We use python code so that system extracts entities, tokens and relationships

LEARNING ENGLISH

It would be impossible for system to understand sentences from dataset

CREATING KNOWLEDGE GRAPH

Knowledge graph is created by the system using extracted entities, tokens and relationships

Get Entities Function

```python
def get_entities(sent):
  ## chunk 1
  ent1 = ""
  ent2 = ""

  prv_tok_dep = ""    # dependency tag of previous token in the sentence
  prv_tok_text = ""   # previous token in the sentence

  prefix = ""
  modifier = ""

  #############################################################

  for tok in nlp(sent):
    ## chunk 2
    # if token is a punctuation mark then move on to the next token
    if tok.dep_ != "punct":
      # check: token is a compound word or not
      if tok.dep_ == "compound":
        prefix = tok.text
        # if the previous word was also a 'compound' then add the current word to it
        if prv_tok_dep == "compound":
          prefix = prv_tok_text + " "+ tok.text

      # check: token is a modifier or not
      if tok.dep_.endswith("mod") == True:
        modifier = tok.text
        # if the previous word was also a 'compound' then add the current word to it
        if prv_tok_dep == "compound":
          modifier = prv_tok_text + " "+ tok.text

      ## chunk 3
      if tok.dep_.find("subj") == True:
        ent1 = modifier +" "+ prefix + " "+ tok.text
        prefix = ""
        modifier = ""
        prv_tok_dep = ""
        prv_tok_text = ""

      ## chunk 4
      if tok.dep_.find("obj") == True:
        ent2 = modifier +" "+ prefix +" "+ tok.text

      ## chunk 5
      # update variables
      prv_tok_dep = tok.dep_
      prv_tok_text = tok.text
  #############################################################

  return [ent1.strip(), ent2.strip()]
```

Get Relationship Function

[13]

```python
def get_relation(sent):

  doc = nlp(sent)

  # Matcher class object
  matcher = Matcher(nlp.vocab)

  #define the pattern
  pattern = [{'DEP':'ROOT'},
            {'DEP':'prep','OP':"?"},
            {'DEP':'agent','OP':"?"},
            {'POS':'ADJ','OP':"?"}]

  matcher.add("matching_1", [pattern])

  matches = matcher(doc)
  k = len(matches) - 1

  span = doc[matches[k][1]:matches[k][2]]

  return(span.text)
```

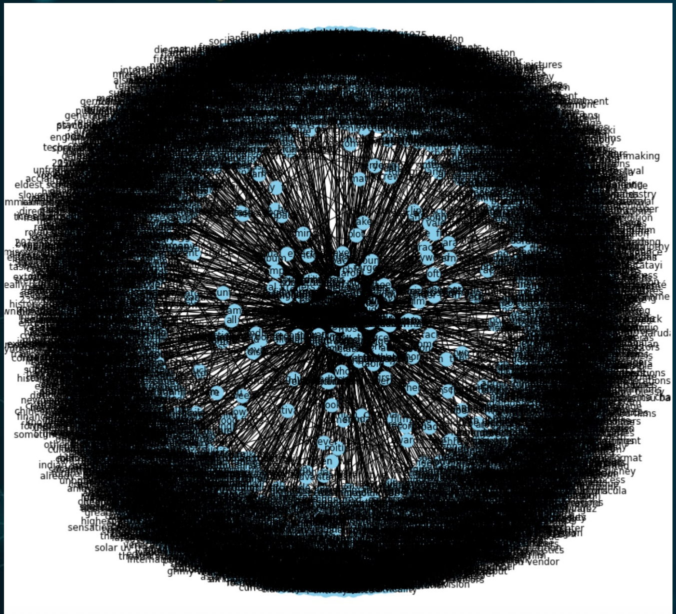
OUTCOME

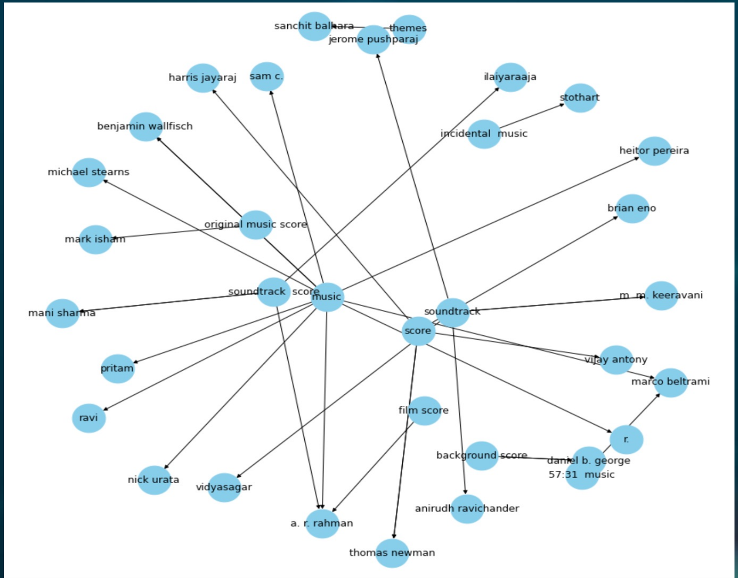
05

What do we get?

WHAT DO WE GET?

We get a knowledge graph using which we can get graph of particular queries





CONCLUSION

06

THANK YOU!