CS236299 Project Segment 4: Semantic Interpretation – Question Answering

July 21, 2022

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
# my files are in 'labs/lab0-0'
!cp -r /content/drive/MyDrive/project4-OfekGlick-master/* .
!pip install -r requirements.txt
# restart the runtime
import os
os._exit(00)
```

```
Mounted at /content/drive
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages
(from -r requirements.txt (line 1)) (3.7)
Collecting cryptography
  Downloading cryptography-37.0.4-cp36-abi3-manylinux_2_24_x86_64.whl (4.1 MB)
     |                 | 4.1 MB 30.9 MB/s
Collecting torchtext==0.11.2
  Downloading torchtext-0.11.2-cp37-cp37m-manylinux1_x86_64.whl (8.0 MB)
     |                 | 8.0 MB 17.0 MB/s
Requirement already satisfied: torch>=1.9.0 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 4))
(1.12.0+cu113)
Collecting func_timeout
  Downloading func_timeout-4.3.5.tar.gz (44 kB)
     |                 | 44 kB 2.2 MB/s
Collecting otter-grader==1.0.0
  Downloading otter_grader-1.0.0-py3-none-any.whl (163 kB)
     |                 | 163 kB 68.5 MB/s
Collecting transformers
  Downloading transformers-4.20.1-py3-none-any.whl (4.4 MB)
     |                 | 4.4 MB 64.0 MB/s
Collecting wget
  Downloading wget-3.2.zip (10 kB)
Collecting torch>=1.9.0
  Downloading torch-1.10.2-cp37-cp37m-manylinux1_x86_64.whl (881.9 MB)
     |                 | 834.1 MB 1.2 MB/s eta
```

```
0:00:41tcmalloc: large alloc 1147494400 bytes == 0x39664000 @  0x7f565030f615
0x592b76 0x4df71e 0x59afff 0x515655 0x549576 0x593fce 0x548ae9 0x51566f 0x549576
0x593fce 0x548ae9 0x5127f1 0x598e3b 0x511f68 0x598e3b 0x511f68 0x598e3b 0x511f68
0x4bc98a 0x532e76 0x594b72 0x515600 0x549576 0x593fce 0x548ae9 0x5127f1 0x549576
0x593fce 0x5118f8 0x593dd7
     |                 | 881.9 MB 1.9 kB/s
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-
packages (from torchtext==0.11.2->-r requirements.txt (line 3)) (4.64.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from torchtext==0.11.2->-r requirements.txt (line 3)) (1.21.6)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-
packages (from torchtext==0.11.2->-r requirements.txt (line 3)) (2.23.0)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.7/dist-packages (from torch>=1.9.0->-r requirements.txt
(line 4)) (4.1.1)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.7/dist-
packages (from otter-grader==1.0.0->-r requirements.txt (line 6)) (5.6.1)
Requirement already satisfied: tornado in /usr/local/lib/python3.7/dist-packages
(from otter-grader==1.0.0->-r requirements.txt (line 6)) (5.1.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages
(from otter-grader==1.0.0->-r requirements.txt (line 6)) (1.3.5)
Collecting PyPDF2
  Downloading PyPDF2-2.6.0-py3-none-any.whl (201 kB)
     |                 | 201 kB 76.2 MB/s
Requirement already satisfied: ipython in /usr/local/lib/python3.7/dist-
packages (from otter-grader==1.0.0->-r requirements.txt (line 6)) (5.5.0)
Collecting docker
  Downloading docker-5.0.3-py2.py3-none-any.whl (146 kB)
     |                 | 146 kB 72.6 MB/s
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-
packages (from otter-grader==1.0.0->-r requirements.txt (line 6)) (3.13)
Requirement already satisfied: nbformat in /usr/local/lib/python3.7/dist-
packages (from otter-grader==1.0.0->-r requirements.txt (line 6)) (5.4.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages
(from otter-grader==1.0.0->-r requirements.txt (line 6)) (2.11.3)
Requirement already satisfied: dill in /usr/local/lib/python3.7/dist-packages
(from otter-grader==1.0.0->-r requirements.txt (line 6)) (0.3.5.1)
Collecting pdfkit
  Downloading pdfkit-1.0.0-py3-none-any.whl (12 kB)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-
packages (from otter-grader==1.0.0->-r requirements.txt (line 6)) (57.4.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages
(from nltk->-r requirements.txt (line 1)) (1.1.0)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages
(from nltk->-r requirements.txt (line 1)) (7.1.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.7/dist-
packages (from nltk->-r requirements.txt (line 1)) (2022.6.2)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.7/dist-
```

```
packages (from cryptography->-r requirements.txt (line 2)) (1.15.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-
packages (from cffi>=1.12->cryptography->-r requirements.txt (line 2)) (2.21)
Collecting pyyaml
  Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manyl
inux_2_12_x86_64.manylinux2010_x86_64.whl (596 kB)
     |                 | 596 kB 72.6 MB/s
Requirement already satisfied: importlib-metadata in
/usr/local/lib/python3.7/dist-packages (from transformers->-r requirements.txt
(line 7)) (4.12.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-
packages (from transformers->-r requirements.txt (line 7)) (21.3)
Collecting tokenizers!=0.11.3,<0.13,>=0.11.1
  Downloading
tokenizers-0.12.1-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (6.6
MB)
     |                 | 6.6 MB 59.0 MB/s
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-
packages (from transformers->-r requirements.txt (line 7)) (3.7.1)
Collecting huggingface-hub<1.0,>=0.1.0
  Downloading huggingface_hub-0.8.1-py3-none-any.whl (101 kB)
     |                 | 101 kB 14.4 MB/s
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from packaging>=20.0->transformers->-r
requirements.txt (line 7)) (3.0.9)
Collecting websocket-client>=0.32.0
  Downloading websocket_client-1.3.3-py3-none-any.whl (54 kB)
     |                 | 54 kB 3.5 MB/s
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests->torchtext==0.11.2->-r
requirements.txt (line 3)) (2022.6.15)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests->torchtext==0.11.2->-r
requirements.txt (line 3)) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->torchtext==0.11.2->-r requirements.txt (line 3)) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests->torchtext==0.11.2->-r
requirements.txt (line 3)) (1.24.3)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-metadata->transformers->-r requirements.txt (line 7))
(3.8.1)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.7/dist-
packages (from ipython->otter-grader==1.0.0->-r requirements.txt (line 6))
(0.7.5)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in
/usr/local/lib/python3.7/dist-packages (from ipython->otter-grader==1.0.0->-r
requirements.txt (line 6)) (1.0.18)
```

```
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-
packages (from ipython->otter-grader==1.0.0->-r requirements.txt (line 6))
(5.1.1)
Requirement already satisfied: simplegeneric>0.8 in
/usr/local/lib/python3.7/dist-packages (from ipython->otter-grader==1.0.0->-r
requirements.txt (line 6)) (0.8.1)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-
packages (from ipython->otter-grader==1.0.0->-r requirements.txt (line 6))
(4.4.2)
Requirement already satisfied: pexpect in /usr/local/lib/python3.7/dist-packages
(from ipython->otter-grader==1.0.0->-r requirements.txt (line 6)) (4.8.0)
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-
packages (from ipython->otter-grader==1.0.0->-r requirements.txt (line 6))
(2.6.1)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages
(from prompt-toolkit<2.0.0,>=1.0.4->ipython->otter-grader==1.0.0->-r
requirements.txt (line 6)) (0.2.5)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-
packages (from prompt-toolkit<2.0.0,>=1.0.4->ipython->otter-grader==1.0.0->-r
requirements.txt (line 6)) (1.15.0)
Requirement already satisfied: MarkupSafe>=0.23 in
/usr/local/lib/python3.7/dist-packages (from jinja2->otter-grader==1.0.0->-r
requirements.txt (line 6)) (2.0.1)
Requirement already satisfied: testpath in /usr/local/lib/python3.7/dist-
packages (from nbconvert->otter-grader==1.0.0->-r requirements.txt (line 6))
(0.6.0)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.7/dist-packages (from nbconvert->otter-grader==1.0.0->-r
requirements.txt (line 6)) (1.5.0)
Requirement already satisfied: bleach in /usr/local/lib/python3.7/dist-packages
(from nbconvert->otter-grader==1.0.0->-r requirements.txt (line 6)) (5.0.1)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.7/dist-
packages (from nbconvert->otter-grader==1.0.0->-r requirements.txt (line 6))
(0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in
/usr/local/lib/python3.7/dist-packages (from nbconvert->otter-grader==1.0.0->-r
requirements.txt (line 6)) (0.4)
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.7/dist-
packages (from nbconvert->otter-grader==1.0.0->-r requirements.txt (line 6))
(4.11.1)
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.7/dist-packages (from nbconvert->otter-grader==1.0.0->-r
requirements.txt (line 6)) (0.8.4)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.7/dist-
packages (from nbformat->otter-grader==1.0.0->-r requirements.txt (line 6))
(4.3.3)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.7/dist-
packages (from nbformat->otter-grader==1.0.0->-r requirements.txt (line 6))
```

```
(2.15.3)
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.7/dist-
packages (from jsonschema>=2.6->nbformat->otter-grader==1.0.0->-r
requirements.txt (line 6)) (21.4.0)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in
/usr/local/lib/python3.7/dist-packages (from jsonschema>=2.6->nbformat->otter-
grader==1.0.0->-r requirements.txt (line 6)) (0.18.1)
Requirement already satisfied: importlib-resources>=1.4.0 in
/usr/local/lib/python3.7/dist-packages (from jsonschema>=2.6->nbformat->otter-
grader==1.0.0->-r requirements.txt (line 6)) (5.8.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.7/dist-
packages (from bleach->nbconvert->otter-grader==1.0.0->-r requirements.txt (line
6)) (0.5.1)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-
packages (from pandas->otter-grader==1.0.0->-r requirements.txt (line 6))
(2022.1)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas->otter-grader==1.0.0->-r
requirements.txt (line 6)) (2.8.2)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-
packages (from pexpect->ipython->otter-grader==1.0.0->-r requirements.txt (line
6)) (0.7.0)
Building wheels for collected packages: func-timeout, wget
  Building wheel for func-timeout (setup.py) … done
  Created wheel for func-timeout: filename=func_timeout-4.3.5-py3-none-any.whl
size=15098
sha256=9c03ca619c6bfe1a21fa271f46f4f681a497de5e064abb82b24a5d4c1bb666c9
  Stored in directory: /root/.cache/pip/wheels/68/b5/a5/67c4364c354e141f5a1bd3ec
568126f77877ab7554cf5af8cb
  Building wheel for wget (setup.py) … done
  Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=9675
sha256=f6081d0f668a7cf7ba1fde799cc52cfddcfd51640d5b898a057ec00851ca893a
  Stored in directory: /root/.cache/pip/wheels/a1/b6/7c/0e63e34eb06634181c63adac
ca38b79ff8f35c37e3c13e3c02
Successfully built func-timeout wget
Installing collected packages: websocket-client, pyyaml, torch, tokenizers,
PyPDF2, pdfkit, huggingface-hub, docker, wget, transformers, torchtext, otter-
grader, func-timeout, cryptography
  Attempting uninstall: pyyaml
    Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
  Attempting uninstall: torch
    Found existing installation: torch 1.12.0+cu113
    Uninstalling torch-1.12.0+cu113:
      Successfully uninstalled torch-1.12.0+cu113
  Attempting uninstall: torchtext
    Found existing installation: torchtext 0.13.0
```

```
    Uninstalling torchtext-0.13.0:
      Successfully uninstalled torchtext-0.13.0
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
torchvision 0.13.0+cu113 requires torch==1.12.0, but you have torch 1.10.2 which
is incompatible.
torchaudio 0.12.0+cu113 requires torch==1.12.0, but you have torch 1.10.2 which
is incompatible.
Successfully installed PyPDF2-2.6.0 cryptography-37.0.4 docker-5.0.3 func-
timeout-4.3.5 huggingface-hub-0.8.1 otter-grader-1.0.0 pdfkit-1.0.0 pyyaml-6.0
tokenizers-0.12.1 torch-1.10.2 torchtext-0.11.2 transformers-4.20.1 websocket-
client-1.3.3 wget-3.2
```

[1]:
```python
# Please do not change this cell because some hidden tests might depend on it.
import os

# Otter grader does not handle ! commands well, so we define and use our
# own function to execute shell commands.
def shell(commands, warn=True):
    """Executes the string `commands` as a sequence of shell commands.

       Prints the result to stdout and returns the exit status.
       Provides a printed warning on non-zero exit status unless `warn`
       flag is unset.
    """
    file = os.popen(commands)
    print (file.read().rstrip('\n'))
    exit_status = file.close()
    if warn and exit_status != None:
        print(f"Completed with errors. Exit status: {exit_status}\n")
    return exit_status

shell("""
ls requirements.txt >/dev/null 2>&1
if [ ! $? = 0 ]; then
 rm -rf .tmp
 git clone https://github.com/cs236299-2022-spring/project4.git .tmp
 mv .tmp/requirements.txt ./
 rm -rf .tmp
fi
pip install -q -r requirements.txt
""")
```

[2]:
```
# Initialize Otter
import otter
grader = otter.Notebook()
```

1 236299 - Introduction to Natural Language Processing

1.1 Project 4: Semantic Interpretation – Question Answering

The goal of semantic parsing is to convert natural language utterances to a meaning representation such as a *logical form* expression or a *SQL query*. In the previous project segment, you built a parsing system to reconstruct parse trees from the natural-language queries in the ATIS dataset. However, that only solves an intermediary task, not the end-user task of obtaining answers to the queries.

In this final project segment, you will go further, building a semantic parsing system to convert English queries to SQL queries, so that by consulting a database you will be able to answer those questions. You will implement both a rule-based approach and an end-to-end sequence-to-sequence (seq2seq) approach. Both algorithms come with their pros and cons, and by the end of this segment you should have a basic understanding of the characteristics of the two approaches.

1.2 Goals

1. Build a semantic parsing algorithm to convert text to SQL queries based on the syntactic parse trees from the last project.
2. Build an attention-based end-to-end seq2seq system to convert text to SQL.
3. Improve the attention-based end-to-end seq2seq system with self-attention to convert text to SQL.
4. Discuss the pros and cons of the rule-based system and the end-to-end system.
5. (Optional) Use the state-of-the-art pretrained transformers for text-to-SQL conversion.

This will be an extremely challenging project, so we recommend that you start early.

2 Setup

[3]:
```
import copy
import datetime
import math
import re
import sys
import warnings

import wget
import nltk
import sqlite3
import torch
```

```python
import torch.nn as nn
import torchtext.legacy as tt

from cryptography.fernet import Fernet
from func_timeout import func_set_timeout
from torch.nn.utils.rnn import pack_padded_sequence as pack
from torch.nn.utils.rnn import pad_packed_sequence as unpack
from tqdm import tqdm
from transformers import BartTokenizer, BartForConditionalGeneration
```

[4]:
```python
# Set random seeds
seed = 1234
torch.manual_seed(seed)
# Set timeout for executing SQL
TIMEOUT = 3 # seconds

# GPU check: Set runtime type to use GPU where available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print (device)
```

```
cuda
```

[5]:
```python
## Download needed scripts and data
os.makedirs('data', exist_ok=True)
os.makedirs('scripts', exist_ok=True)
source_url = "https://raw.githubusercontent.com/nlp-236299/data/master"

# Grammar to augment for this segment
if not os.path.isfile('data/grammar'):
  wget.download(f"{source_url}/ATIS/grammar_distrib4.crypt", out="data/")

  # Decrypt the grammar file
  key = b'bfksTY2BJ5VKKK9xZb1PDDLaGkdu7KCDFYfVePSEfGY='
  fernet = Fernet(key)
  with open('./data/grammar_distrib4.crypt', 'rb') as f:
    restored = Fernet(key).decrypt(f.read())
  with open('./data/grammar', 'wb') as f:
    f.write(restored)

# Download scripts and ATIS database
wget.download(f"{source_url}/scripts/trees/transform.py", out="scripts/")
wget.download(f"{source_url}/ATIS/atis_sqlite.db", out="data/")
```

[5]: 'data//atis_sqlite (1).db'

[6]:
```python
# Import downloaded scripts for parsing augmented grammars
sys.path.insert(1, './scripts')
import transform as xform
```

3 Semantically augmented grammars

In the first part of this project segment, you'll be implementing a rule-based system for semantic interpretation of sentences. Before jumping into using such a system on the ATIS dataset – we'll get to that soon enough – let's first work with some trivial examples to get things going.

The fundamental idea of rule-based semantic interpretation is the rule of compositionality, that *the meaning of a constituent is a function of the meanings of its immediate subconstituents and the syntactic rule that combined them.* This leads to an infrastructure for specifying semantic interpretation in which each syntactic rule in a grammar (in our case, a context-free grammar) is associated with a semantic rule that applies to the meanings associated with the elements on the right-hand side of the rule.

3.1 Example: arithmetic expressions

As a first example, let's consider an augmented grammar for arithmetic expressions, familiar from lab 3-1. We again use the function `xform.parse_augmented_grammar` to parse the augmented grammar. You can read more about it in the file `scripts/transform.py`.

[7]:
```
arithmetic_grammar, arithmetic_augmentations = xform.parse_augmented_grammar(
    """
    ## Sample grammar for arithmetic expressions

    S -> NUM                              : lambda Num: Num
       | S OP S                           : lambda S1, Op, S2: Op(S1, S2)

    OP -> ADD                             : lambda Op: Op
        | SUB
        | MULT
        | DIV

    NUM -> 'zero'                         : lambda: 0
         | 'one'                          : lambda: 1
         | 'two'                          : lambda: 2
         | 'three'                        : lambda: 3
         | 'four'                         : lambda: 4
         | 'five'                         : lambda: 5
         | 'six'                          : lambda: 6
         | 'seven'                        : lambda: 7
         | 'eight'                        : lambda: 8
         | 'nine'                         : lambda: 9
         | 'ten'                          : lambda: 10

    ADD -> 'plus' | 'added' 'to'          : lambda: lambda x, y: x + y
    SUB -> 'minus'                        : lambda: lambda x, y: x - y
    MULT -> 'times' | 'multiplied' 'by'   : lambda: lambda x, y: x * y
    DIV -> 'divided' 'by'                 : lambda: lambda x, y: x / y
    """
```

```
)
```

Recall that in this grammar specification format, rules that are not explicitly provided with an augmentation (like all the `OP` rules after the first `OP -> ADD`) are associated with the textually most recent one (`lambda Op: Op`).

The `parse_augmented_grammar` function returns both an NLTK grammar and a dictionary that maps from productions in the grammar to their associated augmentations. Let's examine the returned grammar.

[8]:
```
for production in arithmetic_grammar.productions():
  print(f"{repr(production):25}     {arithmetic_augmentations[production]}")
```

```
S -> NUM                     <function <lambda> at 0x7fdb3bb56d40>
S -> S OP S                  <function <lambda> at 0x7fdb3bb56dd0>
OP -> ADD                    <function <lambda> at 0x7fdb3bb56ef0>
OP -> SUB                    <function <lambda> at 0x7fdb3bb58050>
OP -> MULT                   <function <lambda> at 0x7fdb3bb58170>
OP -> DIV                    <function <lambda> at 0x7fdb3bb58290>
NUM -> 'zero'                <function <lambda> at 0x7fdb3bb583b0>
NUM -> 'one'                 <function <lambda> at 0x7fdb3bb584d0>
NUM -> 'two'                 <function <lambda> at 0x7fdb3bb585f0>
NUM -> 'three'               <function <lambda> at 0x7fdb3bb58710>
NUM -> 'four'                <function <lambda> at 0x7fdb3bb58830>
NUM -> 'five'                <function <lambda> at 0x7fdb3bb58950>
NUM -> 'six'                 <function <lambda> at 0x7fdb3bb58a70>
NUM -> 'seven'               <function <lambda> at 0x7fdb3bb58b90>
NUM -> 'eight'               <function <lambda> at 0x7fdb3bb58cb0>
NUM -> 'nine'                <function <lambda> at 0x7fdb3bb58dd0>
NUM -> 'ten'                 <function <lambda> at 0x7fdb3bb58ef0>
ADD -> 'plus'                <function <lambda> at 0x7fdb3bb5a0e0>
ADD -> 'added' 'to'          <function <lambda> at 0x7fdb3bb5a290>
SUB -> 'minus'               <function <lambda> at 0x7fdb3bb5a440>
MULT -> 'times'              <function <lambda> at 0x7fdb3bb5a5f0>
MULT -> 'multiplied' 'by'    <function <lambda> at 0x7fdb3bb5a7a0>
DIV -> 'divided' 'by'        <function <lambda> at 0x7fdb3bb5a950>
```

We can parse with the grammar using one of the built-in NLTK parsers.

[9]:
```
arithmetic_parser = nltk.parse.BottomUpChartParser(arithmetic_grammar)
parses = [p for p in arithmetic_parser.parse('three plus one times four'.
↪split())]
for parse in parses:
  parse.pretty_print()
```

```
            S
        ____|_________
       S         |    |
   ____|____     |    |
  S    OP   S    OP   S
```

```
  |     |     |     |     |
 NUM  ADD  NUM  MULT NUM
  |     |     |     |     |
three plus one times four
```

```
           S
   _________|____
  |    |        S
  |    |     ____|____
  S    OP   S    OP   S
  |    |    |    |    |
 NUM  ADD  NUM  MULT NUM
  |    |    |    |    |
three plus one times four
```

Now let's turn to the augmentations. They can be arbitrary Python functions applied to the semantic representations associated with the right-hand-side nonterminals, returning the semantic representation of the left-hand side. To interpret the semantic representation of the entire sentence (at the root of the parse tree), we can use the following pseudo-code:

```
to interpret a tree:
  interpret each of the nonterminal-rooted subtrees
  find the augmentation associated with the root production of the tree
    (it should be a function of as many arguments as there are nonterminals on the right-hand s
  return the result of applying the augmentation to the subtree values
```

(The base case of this recursion occurs when the number of nonterminal-rooted subtrees is zero, that is, a rule all of whose right-hand side elements are terminals.)

Suppose we had such a function, call it `interpret`. How would it operate on, for instance, the tree `(S (S (NUM three)) (OP (ADD plus)) (S (NUM one)))`?

```
interpret (S (S (NUM three)) (OP (ADD plus)) (S (NUM one)))
    |->interpret (S (NUM three))
    |      |->interpret (NUM three)
    |      |      |->(no subconstituents to evaluate)
    |      |      |->apply the augmentation for the rule NUM -> three to the empty set of valu
    |      |      |      (lambda: 3) () ==> 3
    |      |      \==> 3
    |      |->apply the augmentation for the rule S -> NUM to the value 3
    |      |      (lambda NUM: NUM)(3) ==> 3
    |      \==> 3
    |->interpret (OP (ADD plus))
    |      |...
    |      \==> lambda x, y: x + y
    |->interpret (S (NUM one))
    |      |...
    |      \==> 1
    |->apply the augmentation for the rule S -> S OP S to the values 3, (lambda x, y: x + y), a
```

```
    |       (lambda S1, Op, S2: Op(S1, S2))(3, (lambda x, y: x + y), 1) ==> 4
    \==> 4
```

Thus, the string "three plus one" is semantically interpreted as the value 4.

We provide the `interpret` function to carry out this recursive process, copied over from lab 4-2:

[10]:
```python
def interpret(tree, augmentations):
  syntactic_rule = tree.productions()[0]
  semantic_rule = augmentations[syntactic_rule]
  child_meanings = [interpret(child, augmentations)
                    for child in tree
                    if isinstance(child, nltk.Tree)]
  return semantic_rule(*child_meanings)
```

Now we should be able to evaluate the arithmetic example from above.

[11]:
```python
interpret(parses[0], arithmetic_augmentations)
```

[11]: 16

And we can even write a function that parses and interprets a string. We'll have it evaluate each of the possible parses and print the results.

[12]:
```python
def parse_and_interpret(string, grammar, augmentations):
  parser = nltk.parse.BottomUpChartParser(grammar)
  parses = parser.parse(string.split())
  for parse in parses:
    parse.pretty_print()
    print(parse, "==>", interpret(parse, augmentations))
```

[13]:
```python
parse_and_interpret("three plus one times four", arithmetic_grammar,
 ↪arithmetic_augmentations)
```

```
            S
        ____|_________
       S         |    |
   ____|____     |    |
  S    OP   S    OP   S
  |    |    |    |    |
 NUM  ADD  NUM  MULT NUM
  |    |    |    |    |
three plus one times four

(S
  (S (S (NUM three)) (OP (ADD plus)) (S (NUM one)))
  (OP (MULT times))
  (S (NUM four))) ==> 16
            S
   _________|____
```

```
  |    |         S
  |    |     ____|____
  S    OP   S    OP   S
  |    |    |    |    |
 NUM  ADD  NUM  MULT NUM
  |    |    |    |    |
three plus one times four

(S
  (S (NUM three))
  (OP (ADD plus))
  (S (S (NUM one)) (OP (MULT times)) (S (NUM four)))) ==> 7
```

Since the string is syntactically ambiguous according to the grammar, it is semantically ambiguous as well.

3.2 Some grammar specification conveniences

Before going on, it will be useful to have a few more conveniences in writing augmentations for rules. First, since the augmentations are arbitrary Python expressions, they can be built from and make use of other functions. For instance, you'll notice that many of the augmentations at the leaves of the tree took no arguments and returned a constant. We can define a function `constant` that returns a function that ignores its arguments and returns a particular value.

[14]:
```python
def constant(value):
    """Return `value`, ignoring any arguments"""
    return lambda *args: value
```

Similarly, several of the augmentations are functions that just return their first argument. Again, we can define a generic form `first` of such a function:

[15]:
```python
def first(*args):
    """Return the value of the first (and perhaps only) subconstituent,
       ignoring any others"""
    return args[0]
```

We can now rewrite the grammar above to take advantage of these shortcuts.

> In the call to `parse_augmented_grammar` below, we pass in the global environment, extracted via a `globals()` function call, via the named argument `globals`. This allows the `parse_augmented_grammar` function to make use of the global bindings for `constant`, `first`, and the like when evaluating the augmentation expressions to their values. You can check out the code in `transform.py` to see how the passed in `globals` bindings are used. To help understand what's going on, see what happens if you don't include the `globals=globals()`.

[16]:
```python
arithmetic_grammar_2, arithmetic_augmentations_2 = xform.
 ↪parse_augmented_grammar(
    """
    ## Sample grammar for arithmetic expressions
```

```
    S -> NUM                               : first
       | S OP S                            : lambda S1, Op, S2: Op(S1, S2)

    OP -> ADD                              : first
       | SUB
       | MULT
       | DIV

    NUM -> 'zero'                          : constant(0)
         | 'one'                           : constant(1)
         | 'two'                           : constant(2)
         | 'three'                         : constant(3)
         | 'four'                          : constant(4)
         | 'five'                          : constant(5)
         | 'six'                           : constant(6)
         | 'seven'                         : constant(7)
         | 'eight'                         : constant(8)
         | 'nine'                          : constant(9)
         | 'ten'                           : constant(10)

    ADD -> 'plus' | 'added' 'to'           : constant(lambda x, y: x + y)
    SUB -> 'minus'                         : constant(lambda x, y: x - y)
    MULT -> 'times' | 'multiplied' 'by'    : constant(lambda x, y: x * y)
    DIV -> 'divided' 'by'                  : constant(lambda x, y: x / y)
    """,
    globals=globals())
```

Finally, it might make our lives easier to write a template of augmentations whose instantiation depends on the right-hand side of the rule.

We use a reserved keyword `_RHS` to denote the right-hand side of the syntactic rule, which will be replaced by a **list** of the right-hand-side strings. For example, an augmentation `numeric_template(_RHS)` would be as if written as `numeric_template(['zero'])` when the rule is `NUM -> 'zero'`, and `numeric_template(['one'])` when the rule is `NUM -> 'one'`. The details of how this works can be found at `scripts/transform.py`.

This would allow us to use a single template function, for example,

[17]:
```
def numeric_template(rhs):
  """Ignore the subphrase meanings and lookup the first right-hand-side symbol
     as a number"""
  return constant({'zero':0, 'one':1, 'two':2, 'three':3, 'four':4, 'five':5,
          'six':6, 'seven':7, 'eight':8, 'nine':9, 'ten':10}[rhs[0]])
```

and then further simplify the grammar specification:

[18]:
```
arithmetic_grammar_3, arithmetic_augmentations_3 = xform.
 ↪parse_augmented_grammar(
```

```
    """
    ## Sample grammar for arithmetic expressions

    S -> NUM                              : first
       | S OP S                           : lambda S1, Op, S2: Op(S1, S2)

    OP -> ADD                             : first
       | SUB
       | MULT
       | DIV

    NUM -> 'zero'  | 'one'   | 'two'      : numeric_template(_RHS)
         | 'three' | 'four'  | 'five'
         | 'six'   | 'seven' | 'eight'
         | 'nine'  | 'ten'

    ADD -> 'plus' | 'added' 'to'          : constant(lambda x, y: x + y)
    SUB -> 'minus'                        : constant(lambda x, y: x - y)
    MULT -> 'times' | 'multiplied' 'by'   : constant(lambda x, y: x * y)
    DIV -> 'divided' 'by'                 : constant(lambda x, y: x / y)
    """,
    globals=globals())
```

[19]:
```
parse_and_interpret("six divided by three", arithmetic_grammar_3,␣
 ↪arithmetic_augmentations_3)
```

```
       S
  _____|______________
 S          OP       S
 |          |        |
NUM        DIV      NUM
 |      _____|___    |
six divided      by three

(S (S (NUM six)) (OP (DIV divided by)) (S (NUM three))) ==> 2.0
```

3.3 Example: *Green Eggs and Ham* revisited

This stuff is tricky, so it's useful to see more examples before jumping in the deep end. In this simple GEaH fragment grammar, we use a larger set of auxiliary functions to build the augmentations.

[20]:
```
def forward(F, A):
  """Forward application: Return the application of the first
     argument to the second"""
  return F(A)

def backward(A, F):
  """Backward application: Return the application of the second
```

```
        argument to the first"""
    return F(A)

def second(*args):
    """Return the value of the second subconstituent, ignoring any others"""
    return args[1]

def ignore(*args):
    """Return `None`, ignoring everything about the constituent. (Good as a
        placeholder until a better augmentation can be devised.)"""
    return None
```

Using these, we can build and test the grammar.

[21]:
```
geah_grammar_spec = """
    ## Productions
    S -> NP VP              : backward
    VP -> V NP              : forward

    ## Lexicon
    V -> 'likes'            : constant(lambda Object: lambda Subject:␣
  ↪f"like({Subject}, {Object})")
    NP -> 'Sam' | 'sam'     : constant(_RHS[0])
    NP -> 'ham'
    NP -> 'eggs'
"""
```

[22]:
```
geah_grammar, geah_augmentations = xform.
  ↪parse_augmented_grammar(geah_grammar_spec,
                                                                         ␣
  ↪globals=globals())
```

[23]:
```
parse_and_interpret("Sam likes ham", geah_grammar, geah_augmentations)
```

```
     S
  ____|____
 |         VP
 |     ____|___
 NP   V        NP
 |    |        |
Sam likes     ham

(S (NP Sam) (VP (V likes) (NP ham))) ==> like(Sam, ham)
```

4 Semantics of ATIS queries

Now you're in a good position to understand and add augmentations to a more comprehensive grammar, say, one that parses ATIS queries and generates SQL queries.

In preparation for that, we need to load the ATIS data, both NL and SQL queries.

4.1 Loading and preprocessing the corpus

To simplify things a bit, we'll only consider ATIS queries whose question type (remember that from project segment 1?) is `flight_id`. We download training, development, and test splits for this subset of the ATIS corpus, including corresponding SQL queries.

[24]:
```
# Acquire the datasets - training, development, and test splits of the
# ATIS queries and corresponding SQL queries
wget.download(f"{source_url}/ATIS/test_flightid.nl", out="data/")
wget.download(f"{source_url}/ATIS/test_flightid.sql", out="data/")
wget.download(f"{source_url}/ATIS/dev_flightid.nl", out="data/")
wget.download(f"{source_url}/ATIS/dev_flightid.sql", out="data/")
wget.download(f"{source_url}/ATIS/train_flightid.nl", out="data/")
wget.download(f"{source_url}/ATIS/train_flightid.sql", out="data/")
```

[24]:
```
'data//train_flightid (1).sql'
```

Let's take a look at the data: the NL queries are in `.nl` files, and the SQL queries are in `.sql` files.

[25]:
```
shell("head -1 data/dev_flightid.nl")
shell("head -1 data/dev_flightid.sql")
```

```
what flights are available tomorrow from denver to philadelphia
SELECT DISTINCT flight_1.flight_id FROM flight flight_1 , airport_service
airport_service_1 , city city_1 , airport_service airport_service_2 , city
city_2 , days days_1 , date_day date_day_1 WHERE flight_1.from_airport =
airport_service_1.airport_code AND airport_service_1.city_code =
city_1.city_code AND city_1.city_name = 'DENVER' AND ( flight_1.to_airport =
airport_service_2.airport_code AND airport_service_2.city_code =
city_2.city_code AND city_2.city_name = 'PHILADELPHIA' AND flight_1.flight_days
= days_1.days_code AND days_1.day_name = date_day_1.day_name AND date_day_1.year
= 1991 AND date_day_1.month_number = 1 AND date_day_1.day_number = 20 )
```

4.2 Corpus preprocessing

We'll use `torchtext` to process the data. We use two `Field`s: `SRC` for the questions, and `TGT` for the SQL queries. We'll use the tokenizer from project segment 3.

[26]:
```
## Tokenizer
tokenizer = nltk.tokenize.RegexpTokenizer('\d+|st\.|[\w-]+|\$[\d\.]+|\S+')
def tokenize(string):
  return tokenizer.tokenize(string.lower())

## Demonstrating the tokenizer
## Note especially the handling of `"11pm"` and hyphenated words.
print(tokenize("Are there any first-class flights from St. Louis at 11pm for␣
↪less than $3.50?"))
```

```
['are', 'there', 'any', 'first-class', 'flights', 'from', 'st.', 'louis', 'at',
'11', 'pm', 'for', 'less', 'than', '$3.50', '?']
```

[27]:
```
SRC = tt.data.Field(include_lengths=True,         # include lengths
                    batch_first=False,            # batches will be max_len x␣
 ↪batch_size
                    tokenize=tokenize,            # use our tokenizer
                   )
TGT = tt.data.Field(include_lengths=False,
                    batch_first=False,            # batches will be max_len x␣
 ↪batch_size
                    tokenize=lambda x: x.split(), # use split to tokenize
                    init_token="<bos>",           # prepend <bos>
                    eos_token="<eos>")            # append <eos>
fields = [('src', SRC), ('tgt', TGT)]
```

> Note that we specified `batch_first=False` (as in lab 4-4), so that the returned batched tensors would be of size `max_length x batch_size`, which facilitates seq2seq implementation.

Now, we load the data using `torchtext`. We use the `TranslationDataset` class here because our task is essentially a translation task: "translating" questions into the corresponding SQL queries. Therefore, we also refer to the questions as the *source* side (`SRC`) and the SQL queries as the *target* side (`TGT`).

[28]:
```
# Make splits for data
train_data, val_data, test_data = tt.datasets.TranslationDataset.splits(
    ('_flightid.nl', '_flightid.sql'), fields, path='./data/',
    train='train', validation='dev', test='test')

MIN_FREQ = 3
SRC.build_vocab(train_data.src, min_freq=MIN_FREQ)
TGT.build_vocab(train_data.tgt, min_freq=MIN_FREQ)

print (f"Size of English vocab: {len(SRC.vocab)}")
print (f"Most common English words: {SRC.vocab.freqs.most_common(10)}\n")

print (f"Size of SQL vocab: {len(TGT.vocab)}")
print (f"Most common SQL words: {TGT.vocab.freqs.most_common(10)}\n")

print (f"Index for start of sequence token: {TGT.vocab.stoi[TGT.init_token]}")
print (f"Index for end of sequence token: {TGT.vocab.stoi[TGT.eos_token]}")
```

```
Size of English vocab: 421
Most common English words: [('to', 3478), ('from', 3019), ('flights', 2094),
('the', 1550), ('on', 1230), ('me', 973), ('flight', 972), ('show', 845),
('what', 833), ('boston', 813)]

Size of SQL vocab: 392
```

```
Most common SQL words: [('=', 38876), ('AND', 36564), (',', 22772),
('airport_service', 8314), ('city', 8313), ('(', 6432), (')', 6432),
('flight_1.flight_id', 4536), ('flight', 4221), ('SELECT', 4178)]

Index for start of sequence token: 2
Index for end of sequence token: 3
```

Next, we batch our data to facilitate processing on a GPU. Batching is a bit tricky because the source and target will typically be of different lengths. Fortunately, `torchtext` allows us to pass in a `sort_key` function. By sorting on length, we can minimize the amount of padding on the source side, but since there is still some padding, we need to handle them with `pack` and `unpack` later on in the seq2seq part (as in lab 4-5).

[29]:
```
BATCH_SIZE = 16 # batch size for training/validation
TEST_BATCH_SIZE = 1 # batch size for test, we use 1 to make beam search␣
 ↪implementation easier

train_iter, val_iter = tt.data.BucketIterator.splits((train_data, val_data),
                                                     batch_size=BATCH_SIZE,
                                                     device=device,
                                                     repeat=False,
                                                     sort_key=lambda x: len(x.
 ↪src),
                                                     sort_within_batch=True)
test_iter = tt.data.BucketIterator(test_data,
                                   batch_size=TEST_BATCH_SIZE,
                                   device=device,
                                   repeat=False,
                                   sort=False,
                                   train=False)
```

Let's look at a single batch from one of these iterators.

[30]:
```
batch = next(iter(train_iter))
train_batch_text, train_batch_text_lengths = batch.src
print (f"Size of text batch: {train_batch_text.shape}")
print (f"Third sentence in batch: {train_batch_text[:, 2]}")
print (f"Length of the third sentence in batch: {train_batch_text_lengths[2]}")
print (f"Converted back to string: {' '.join([SRC.vocab.itos[i] for i in␣
 ↪train_batch_text[:, 2]])}")

train_batch_sql = batch.tgt
print (f"Size of sql batch: {train_batch_sql.shape}")
print (f"Third SQL in batch: {train_batch_sql[:, 2]}")
print (f"Converted back to string: {' '.join([TGT.vocab.itos[i] for i in␣
 ↪train_batch_sql[:, 2]])}")
```

```
Size of text batch: torch.Size([7, 16])
Third sentence in batch: tensor([ 8, 89,  3, 14,  2, 13, 16], device='cuda:0')
```

```
Length of the third sentence in batch: 7
Converted back to string: flight information from denver to san francisco
Size of sql batch: torch.Size([111, 16])
Third SQL in batch: tensor([ 2, 14, 31, 11, 13, 12, 16,  6,  7, 22,  6,  8, 23,
6,  7, 29,  6,  8,
        30, 15, 21,  4, 18,  5, 19,  4, 17,  5, 20,  4, 55,  5, 24,  4, 25,  5,
        26,  4, 27,  5, 28,  4, 54, 56,  3,  1,  1,  1,  1,  1,  1,  1,  1,  1,
         1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
         1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
         1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
         1,  1,  1], device='cuda:0')
Converted back to string: <bos> SELECT DISTINCT flight_1.flight_id FROM flight
flight_1 , airport_service airport_service_1 , city city_1 , airport_service
airport_service_2 , city city_2 WHERE flight_1.from_airport =
airport_service_1.airport_code AND airport_service_1.city_code =
city_1.city_code AND city_1.city_name = 'DENVER' AND flight_1.to_airport =
airport_service_2.airport_code AND airport_service_2.city_code =
city_2.city_code AND city_2.city_name = 'SAN FRANCISCO' <eos> <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
```

Alternatively, we can directly iterate over the raw examples:

[31]:
```python
for example in train_iter.dataset[:1]:
  train_text_1 = ' '.join(example.src) # detokenized question
  train_sql_1 = ' '.join(example.tgt)  # detokenized sql
  print (f"Question: {train_text_1}\n")
  print (f"SQL: {train_sql_1}")
```

```
Question: list all the flights that arrive at general mitchell international
from various cities

SQL: SELECT DISTINCT flight_1.flight_id FROM flight flight_1 , airport airport_1
, airport_service airport_service_1 , city city_1 WHERE flight_1.to_airport =
airport_1.airport_code AND airport_1.airport_code = 'MKE' AND
flight_1.from_airport = airport_service_1.airport_code AND
airport_service_1.city_code = city_1.city_code AND 1 = 1
```

4.3 Establishing a SQL database for evaluating ATIS queries

The output of our systems will be SQL queries. How should we determine if the generated queries are correct? We can't merely compare against the gold SQL queries, since there are many ways to implement a SQL query that answers any given NL query.

Instead, we will execute the queries – both the predicted SQL query and the gold SQL query – on an actual database, and verify that the returned responses are the same. For that purpose, we need a SQL database server to use. We'll set one up here, using the Python `sqlite3` module.

[32]:
```
@func_set_timeout(TIMEOUT)
def execute_sql(sql):
  conn = sqlite3.connect('data/atis_sqlite.db')  # establish the DB based on␣
 ↪the downloaded data
  c = conn.cursor()                              # build a "cursor"
  c.execute(sql)
  results = list(c.fetchall())
  c.close()
  conn.close()
  return results
```

To run a query, we use the cursor's `execute` function, and retrieve the results with `fetchall`. Let's get all the flights that arrive at General Mitchell International – the query `train_sql_1` above. There's a lot, so we'll just print out the first few.

[33]:
```
predicted_ret = execute_sql(train_sql_1)

print(f"""
Executing: {train_sql_1}

Result: {len(predicted_ret)} entries starting with

{predicted_ret[:10]}
""")
```

```
Executing: SELECT DISTINCT flight_1.flight_id FROM flight flight_1 , airport
airport_1 , airport_service airport_service_1 , city city_1 WHERE
flight_1.to_airport = airport_1.airport_code AND airport_1.airport_code = 'MKE'
AND flight_1.from_airport = airport_service_1.airport_code AND
airport_service_1.city_code = city_1.city_code AND 1 = 1

Result: 534 entries starting with

[(107929,), (107930,), (107931,), (107932,), (107933,), (107934,), (107935,),
(107936,), (107937,), (107938,)]
```

For your reference, the SQL database we are using has a database schema described at https://github.com/jkkummerfeld/text2sql-data/blob/master/data/atis-schema.csv, and is consistent with the SQL queries provided in the various `.sql` files loaded above.

5 Rule-based parsing and interpretation of ATIS queries

First, you will implement a rule-based semantic parser using a grammar like the one you completed in the third project segment. We've placed an initial grammar in the file `data/grammar`. In addition to the helper functions defined above (`constant`, `first`, etc.), it makes use of some other simple functions. We've included those below, but you can (and almost certainly should) augment this set with others that you define as you build out the full set of augmentations.

[34]:
```
def upper(term):
  return '"' + term.upper() + '"'

def weekday(day):
  return f"flight.flight_days IN (SELECT days.days_code FROM days WHERE days.
↪day_name = '{day.upper()}')"

def month_name(month):
  return {'JANUARY' : 1,
          'FEBRUARY' : 2,
          'MARCH' : 3,
          'APRIL' : 4,
          'MAY' : 5,
          'JUNE' : 6,
          'JULY' : 7,
          'AUGUST' : 8,
          'SEPTEMBER' : 9,
          'OCTOBER' : 10,
          'NOVEMBER' : 11,
          'DECEMBER' : 12}[month.upper()]

def airports_from_airport_name(airport_name):
  return f"(SELECT airport.airport_code FROM airport WHERE airport.airport_name␣
↪= {upper(airport_name)})"

def airports_from_city(city):
  return f"""
    (SELECT airport_service.airport_code FROM airport_service WHERE␣
↪airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = {upper(city)}))
  """

def null_condition(*args, **kwargs):
  return 1

def depart_around(time):
  return f"""
    flight.departure_time >= {add_delta(miltime(time), -15).strftime('%H%M')}
    AND flight.departure_time <= {add_delta(miltime(time), 15).strftime('%H%M')}
    """.strip()

def add_delta(tme, delta):
    # transform to a full datetime first
    return (datetime.datetime.combine(datetime.date.today(), tme) +
```

```
            datetime.timedelta(minutes=delta)).time()

def miltime(minutes):
  return datetime.time(hour=int(minutes/100), minute=(minutes % 100))


# MY FUNCTIONS

def meals(meal):
    return f"""
    flight_id in (
    select flight_id
    from flight
    where flight.meal_code in (select meal_code
                               from food_service
                               where meal_description = {upper(meal)}))

    """
def any_meals(meal):
    return f"""
    flight_id in (
    select flight_id
    from flight
    where flight.meal_code in (select meal_code
                               from food_service
                               where meal_description <> null))

    """

def round_trip():
    return f"""
    flight.flight_id in
        (select flight_rt.flight_id
        from fare fare_rt, flight flight_rt, flight_fare flight_fare_rt
        where flight_rt.flight_id = flight_fare_rt.flight_id
        AND fare_rt.fare_id = flight_fare_rt.fare_id
        AND fare.ROUND_TRIP_REQUIRED = 'YES')
    """
def one_way():
    return f"""
    flight.flight_id in
        (select flight_rt.flight_id
        from fare fare_rt, flight flight_rt, flight_fare flight_fare_rt
        where flight_rt.flight_id = flight_fare_rt.flight_id
        AND fare_rt.fare_id = flight_fare_rt.fare_id
        AND fare.ROUND_TRIP_REQUIRED = 'NO')
    """
```

```
def flight_type(flight_type):
    if "economy" in flight_type:
        cond = "fair_basis.class_type = 'COACH'"
    else:
        cond = "fair_basis.class_type = 'FIRST'"

    return f"""
    flight.flight_id in
        (select flight_rt.flight_id
        from fare fare_rt, flight flight_rt, flight_fare flight_fare_rt
        where flight_rt.flight_id = flight_fare_rt.flight_id
        AND fare_rt.fare_id = flight_fare_rt.fare_id
        AND {cond})
    """


def flight_date(day,month,year):
    cond = ""
    if year != -1:
        cond += f"date_day.year = {year}"
    else:
        cond += f"date_day.year = 1991 "
        if month != -1:
            cond += f"AND date_day.month_number = {month}"
        else:
            if day == -1:
                return ""
            else:
                cond += f" AND date_day.day_number = {day}"
    return f"""
      flight.flight_days IN (
      SELECT days.days_code FROM days WHERE days.day_name IN(
          SELECT
            date_day.day_name
          FROM
            date_day
          WHERE {cond}

    ))
  """
```

We can build a parser with the augmented grammar:

[35]:
```
atis_grammar, atis_augmentations = xform.read_augmented_grammar('data/grammar', 
 ↪globals=globals())
atis_parser = nltk.parse.BottomUpChartParser(atis_grammar)
```

We'll define a function to return a parse tree for a string according to the ATIS grammar (if available).

[36]:
```
def parse_tree(sentence):
  """Parse a sentence and return the parse tree, or None if failure."""
  try:
    parses = list(atis_parser.parse(tokenize(sentence)))
    if len(parses) == 0:
      return None
    else:
      return parses[0]
  except:
    return None
```

We can check the overall coverage of this grammar on the training set by using the `parse_tree` function to determine if a parse is available. The grammar that we provide should get about a 40% coverage of the training set.

[37]:
```
# Check coverage on training set
parsed = 0
with open("data/train_flightid.nl") as train:
  examples = train.readlines()[:]
for sentence in tqdm(examples):
  if parse_tree(sentence):
    parsed += 1
  else:
    next

print(f"\nParsed {parsed} of {len(examples)} ({parsed*100/(len(examples)):.
↪2f}%)")
```

```
100%|      | 3651/3651 [00:16<00:00, 225.38it/s]


Parsed 1609 of 3651 (44.07%)
```

5.1 Goal 1: Construct SQL queries from a parse tree and evaluate the results

It's time to turn to the first major part of this project segment, implementing a rule-based semantic parsing system to answer flight-ID-type ATIS queries.

Recall that in rule-based semantic parsing, each syntactic rule is associated with a semantic composition rule. The grammar we've provided has semantic augmentations for some of the low-level phrases – cities, airports, times, airlines – but not the higher level syntactic types. You'll be adding those.

In the ATIS grammar that we provide, as with the earlier toy grammars, the augmentation for a rule with n nonterminals and m terminals on the right-hand side is assumed to be called with n

positional arguments (the values for the corresponding children). The `interpret` function you've already defined should therefore work well with this grammar.

Let's run through one way that a semantic derivation might proceed, for the sample query "flights to boston":

[38]:
```python
sample_query = "flights to boston"
print(tokenize(sample_query))
sample_tree = parse_tree(sample_query)
sample_tree.pretty_print()
```

```
['flights', 'to', 'boston']
                S
                |
            NP_FLIGHT
                |
            NOM_FLIGHT
                |
             N_FLIGHT
      __________|_________
     |                    PP
     |                    |
     |                 PP_PLACE
     |          _________|_________
  N_FLIGHT     |                 N_PLACE
     |         |                    |
TERM_FLIGHT  P_PLACE            TERM_PLACE
     |         |                    |
  flights      to                 boston
```

Given a sentence, we first construct its parse tree using the syntactic rules, then compose the corresponding semantic rules bottom-up, until eventually we arrive at the root node with a finished SQL statement. For this query, we will go through what the possible meaning representations for the subconstituents of "flights to boston" might be. But this is just one way of doing things; other ways are possible, and you should feel free to experiment.

Working from bottom up:

1. The `TERM_PLACE` phrase "boston" uses the composition function template `constant(airports_from_city(' '.join(_RHS)))`, which will be instantiated as `constant(airports_from_city(' '.join(['boston'])))` (recall that `_RHS` is replaced by the right-hand side of the rule). The meaning of `TERM_PLACE` will be the SQL snippet

```
SELECT airport_service.airport_code
FROM airport_service
WHERE airport_service.city_code IN
    (SELECT city.city_code
     FROM city
     WHERE city.city_name = "BOSTON")
```

(This query generates a list of all of the airports in Boston.)

2. The `N_PLACE` phrase "boston" can have the same meaning as the `TERM_PLACE`.
3. The `P_PLACE` phrase "to" might be associated with a function that maps a SQL query for a list of airports to a SQL condition that holds of flights that go to one of those airports, i.e., `flight.to_airport IN (...)`.
4. The `PP_PLACE` phrase "to boston" might apply the `P_PLACE` meaning to the `TERM_PLACE` meaning, thus generating a SQL condition that holds of flights that go to one of the Boston airports:

```
flight.to_airport IN
    (SELECT airport_service.airport_code
     FROM airport_service
     WHERE airport_service.city_code IN
         (SELECT city.city_code
          FROM city
          WHERE city.city_name = "BOSTON"))
```

5. The `PP` phrase "to Boston" can again get its meaning from the `PP_PLACE`.
6. The `TERM_FLIGHT` phrase "flights" might also return a condition on flights, this time the "null condition", represented by the SQL truth value `1`. Ditto for the `N_FLIGHT` phrase "flights".
7. The `N_FLIGHT` phrase "flights to boston" can conjoin the two conditions, yielding the SQL condition

```
flight.to_airport IN
    (SELECT airport_service.airport_code
     FROM airport_service
     WHERE airport_service.city_code IN
         (SELECT city.city_code
          FROM city
          WHERE city.city_name = "BOSTON"))
AND 1
```

which can be inherited by the `NOM_FLIGHT` and `NP_FLIGHT` phrases.

8. The `S` phrase "flights to boston" can use the condition provided by the `NP_FLIGHT` phrase to select all flights satisfying the condition with a SQL query like

```
SELECT DISTINCT flight.flight_id
FROM flight
WHERE flight.to_airport IN
        (SELECT airport_service.airport_code
         FROM airport_service
         WHERE airport_service.city_code IN
             (SELECT city.city_code
              FROM city
              WHERE city.city_name = "BOSTON"))
      AND 1
```

This SQL query is then taken to be a representation of the meaning for the NL query "flights to boston", and can be executed against the ATIS database to retrieve the requested flights.

Now, it's your turn to add augmentations to `data/grammar` to make this example work. The augmentations that we have provided for the grammar make use of a set of auxiliary functions that we defined above. You should feel free to add your own auxiliary functions that you make use of in the grammar.

[39]:
```
#TODO: add augmentations to `data/grammar` to make this example work
atis_grammar, atis_augmentations = xform.read_augmented_grammar('data/grammar',␣
↪globals=globals())
atis_parser = nltk.parse.BottomUpChartParser(atis_grammar)
predicted_sql = interpret(sample_tree, atis_augmentations)
print("Predicted SQL:\n\n", predicted_sql, "\n")
```

```
Predicted SQL:

 SELECT DISTINCT flight.flight_id FROM flight WHERE (1 AND flight.to_airport IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "BOSTON"))
  )
```

Verification on some examples With a rule-based semantic parsing system, we can generate SQL queries given questions, and then execute those queries on a SQL database to answer the given questions. To evaluate the performance of the system, we compare the returned results against the results of executing the ground truth queries.

We provide a function `verify` to compare the results from our generated SQL to the ground truth SQL. It should be useful for testing individual queries.

[40]:
```
def verify(predicted_sql, gold_sql, silent=True):
  """
  Compare the correctness of the generated SQL by executing on the
  ATIS database and comparing the returned results.
  Arguments:
      predicted_sql: the predicted SQL query
      gold_sql: the reference SQL query to compare against
      silent: print outputs or not
  Returns: True if the returned results are the same, otherwise False
  """
  # Execute predicted SQL
  try:
    predicted_result = execute_sql(predicted_sql)
  except BaseException as e:
    if not silent:
      print(f"predicted sql exec failed: {e}")
    return False
```

```
  if not silent:
    print("Predicted DB result:\n\n", predicted_result[:10], "\n")

  # Execute gold SQL
  try:
    gold_result = execute_sql(gold_sql)
  except BaseException as e:
    if not silent:
      print(f"gold sql exec failed: {e}")
    return False
  if not silent:
    print("Gold DB result:\n\n", gold_result[:10], "\n")

  # Verify correctness
  if gold_result == predicted_result:
    return True
```

Let's try this methodology on a simple example: "flights from phoenix to milwaukee". we provide it along with the gold SQL query.

[41]:
```
def rule_based_trial(sentence, gold_sql):
  print("Sentence: ", sentence, "\n")
  tree = parse_tree(sentence)
  print("Parse:\n\n")
  tree.pretty_print()

  predicted_sql = interpret(tree, atis_augmentations)
  print("Predicted SQL:\n\n", predicted_sql, "\n")

  if verify(predicted_sql, gold_sql, silent=False):
    print ('Correct!')
  else:
    print ('Incorrect!')
```

[42]:
```
# Run this cell to reload augmentations after you make changes to `data/grammar`
atis_grammar, atis_augmentations = xform.read_augmented_grammar('data/grammar',␣
↪globals=globals())
atis_parser = nltk.parse.BottomUpChartParser(atis_grammar)
```

[43]:
```
#TODO: add augmentations to `data/grammar` to make this example work
# Example 1
example_1 = 'flights from phoenix to milwaukee'
gold_sql_1 = """
  SELECT DISTINCT flight_1.flight_id
  FROM flight flight_1 ,
       airport_service airport_service_1 ,
       city city_1 ,
```

```
       airport_service airport_service_2 ,
       city city_2
  WHERE flight_1.from_airport = airport_service_1.airport_code
        AND airport_service_1.city_code = city_1.city_code
        AND city_1.city_name = 'PHOENIX'
        AND flight_1.to_airport = airport_service_2.airport_code
        AND airport_service_2.city_code = city_2.city_code
        AND city_2.city_name = 'MILWAUKEE'
  """

rule_based_trial(example_1, gold_sql_1)
```

```
Sentence:  flights from phoenix to milwaukee

Parse:


                                  S
                                  |
                              NP_FLIGHT
                                  |
                              NOM_FLIGHT
                                  |
                               N_FLIGHT
                  ________________|_________________
            N_FLIGHT                                |
      _________|________                            |
     |                  PP                          PP
     |                  |                           |
     |               PP_PLACE                    PP_PLACE
     |          ________|_________            _______|_________
  N_FLIGHT     |                N_PLACE      |              N_PLACE
     |         |                  |          |                 |
TERM_FLIGHT P_PLACE           TERM_PLACE P_PLACE          TERM_PLACE
     |         |                  |          |                 |
  flights     from             phoenix       to            milwaukee

Predicted SQL:

 SELECT DISTINCT flight.flight_id FROM flight WHERE ((1 AND flight.from_airport
IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "PHOENIX"))
  ) AND flight.to_airport IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "MILWAUKEE"))
```

```
  )

Predicted DB result:

 [(108086,), (108087,), (301763,), (301764,), (301765,), (301766,), (302323,),
(304881,), (310619,), (310620,)]

Gold DB result:

 [(108086,), (108087,), (301763,), (301764,), (301765,), (301766,), (302323,),
(304881,), (310619,), (310620,)]

Correct!
```

To make development faster, we recommend starting with a few examples before running the full evaluation script. We've taken some examples from the ATIS dataset including the gold SQL queries that they provided. Of course, yours (and those of the project segment solution set) may differ.

[44]:
```
#TODO: add augmentations to `data/grammar` to make this example work
# Example 2
example_2 = 'i would like a united flight'
gold_sql_2 = """
  SELECT DISTINCT flight_1.flight_id
  FROM flight flight_1
  WHERE flight_1.airline_code = 'UA'
  """

rule_based_trial(example_2, gold_sql_2)
```

```
Sentence:  i would like a united flight

Parse:


                                                S
____________|___________________________________________________
                                    |
NP_FLIGHT
                                    |
|
                               PREIGNORE
NOM_FLIGHT
        ____________________________|____________
___________|___________
       |                                    PREIGNORE
ADJ                        |
       |                ________________________|____________
|                         |
```

```
       |               |                                PREIGNORE
ADJ_AIRLINE             NOM_FLIGHT
       |               |                        ____________|____________
|                       |
       |               |                         |                      PREIGNORE
TERM_AIRLINE              N_FLIGHT
       |               |                         |                        |
|                       |
PREIGNORESYMBOL PREIGNORESYMBOL           PREIGNORESYMBOL
PREIGNORESYMBOL TERM_AIRBRAND            TERM_FLIGHT
       |               |                         |                        |
|                       |
       i             would                      like                      a
united                 flight

Predicted SQL:

 SELECT DISTINCT flight.flight_id FROM flight WHERE 1 AND flight.airline_code =
'UA'

Predicted DB result:

 [(100094,), (100099,), (100145,), (100158,), (100164,), (100167,), (100169,),
(100203,), (100204,), (100296,)]

Gold DB result:

 [(100094,), (100099,), (100145,), (100158,), (100164,), (100167,), (100169,),
(100203,), (100204,), (100296,)]

Correct!
```

[45]:

```
#TODO: add augmentations to `data/grammar` to make this example work
# Example 3
example_3 = 'i would like a flight between boston and dallas'
gold_sql_3 = """
  SELECT DISTINCT flight_1.flight_id
  FROM flight flight_1 ,
       airport_service airport_service_1 ,
       city city_1 ,
       airport_service airport_service_2 ,
       city city_2
  WHERE flight_1.from_airport = airport_service_1.airport_code
        AND airport_service_1.city_code = city_1.city_code
        AND city_1.city_name = 'BOSTON'
        AND flight_1.to_airport = airport_service_2.airport_code
        AND airport_service_2.city_code = city_2.city_code
```

```
        AND city_2.city_name = 'DALLAS'
  """

# Note that the parse tree might appear wrong: instead of
# `PP_PLACE -> 'between' N_PLACE 'and' N_PLACE`, the tree appears to be
# `PP_PLACE -> 'between' 'and' N_PLACE N_PLACE`. But it's only a visualization
# error of tree.pretty_print() and you should assume that the production is
# `PP_PLACE -> 'between' N_PLACE 'and' N_PLACE` (you can verify by printing out
# all productions).
rule_based_trial(example_3, gold_sql_3)
```

```
Sentence:  i would like a flight between boston and dallas

Parse:


         S
____________________________________________________|__________
                                    |
NP_FLIGHT
                                    |
|
                                    |
NOM_FLIGHT
                                    |
|
                                PREIGNORE
N_FLIGHT
        ____________________________|____________
__________|_________
       |                                     PREIGNORE
|                    PP
       |                ________________________|____________
|                    |
       |               |                                     PREIGNORE
|                 PP_PLACE
       |               |                         ____________|____________
|           _________|___________________
       |               |                         |                    PREIGNORE
N_FLIGHT      |          |      N_PLACE    N_PLACE
       |               |                         |                        |
|          |          |         |          |
PREIGNORESYMBOL PREIGNORESYMBOL           PREIGNORESYMBOL
PREIGNORESYMBOL TERM_FLIGHT     |          |     TERM_PLACE TERM_PLACE
       |               |                         |                        |
|          |          |         |          |
       i               would                     like                     a
flight     between     and       boston     dallas
```

```
Predicted SQL:

 SELECT DISTINCT flight.flight_id FROM flight WHERE (1 AND flight.from_airport
IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "BOSTON"))
   AND flight.to_airport IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "DALLAS"))
  )

Predicted DB result:

 [(103171,), (103172,), (103173,), (103174,), (103175,), (103176,), (103177,),
(103178,), (103179,), (103180,)]

Gold DB result:

 [(103171,), (103172,), (103173,), (103174,), (103175,), (103176,), (103177,),
(103178,), (103179,), (103180,)]

Correct!
```

[46]:

```
#TODO: add augmentations to `data/grammar` to make this example work
# Example 4
example_4 = 'show me the united flights from denver to baltimore'
gold_sql_4 = """
  SELECT DISTINCT flight_1.flight_id
  FROM flight flight_1 ,
       airport_service airport_service_1 ,
       city city_1 ,
       airport_service airport_service_2 ,
       city city_2
  WHERE flight_1.airline_code = 'UA'
        AND ( flight_1.from_airport = airport_service_1.airport_code
              AND airport_service_1.city_code = city_1.city_code
              AND city_1.city_name = 'DENVER'
              AND flight_1.to_airport = airport_service_2.airport_code
              AND airport_service_2.city_code = city_2.city_code
              AND city_2.city_name = 'BALTIMORE' )

  """

rule_based_trial(example_4, gold_sql_4)
```

Sentence: show me the united flights from denver to baltimore

Parse:

```
                                                                                S
-----------------------------------------------------|----------
                      |
NP_FLIGHT
                      |
|
                      |
NOM_FLIGHT
                      |
----------------------|------------------
                      |                                       |
NOM_FLIGHT
                      |                                       |
|
                      |                                       |
N_FLIGHT
                      |                                       |
------------------|----------------
                      |                                       |
N_FLIGHT                                 |
                      |                                       |
----------|--------                                    |
                   PREIGNORE                                 ADJ            |
PP                          PP
        ---------------|-----------                          |            |
|                           |
       |                    PREIGNORE                   ADJ_AIRLINE       |
PP_PLACE                    PP_PLACE
       |              ------------|------------                |            |
---------|---------          -------|---------
       |              |                     PREIGNORE     TERM_AIRLINE
N_FLIGHT      |                N_PLACE      |             N_PLACE
       |              |                           |             |            |
|                   |         |                 |
PREIGNORESYMBOL PREIGNORESYMBOL           PREIGNORESYMBOL TERM_AIRBRAND
TERM_FLIGHT  P_PLACE            TERM_PLACE P_PLACE          TERM_PLACE
       |              |                           |             |            |
|                   |         |                 |
      show             me                         the           united
flights      from               denver      to            baltimore
```

Predicted SQL:

```
 SELECT DISTINCT flight.flight_id FROM flight WHERE ((1 AND flight.from_airport
IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "DENVER"))
  ) AND flight.to_airport IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "BALTIMORE"))
  ) AND flight.airline_code = 'UA'

Predicted DB result:

 [(101231,), (101233,), (305983,)]

Gold DB result:

 [(101231,), (101233,), (305983,)]

Correct!
```

[47]:

```
#TODO: add augmentations to `data/grammar` to make this example work
# Example 5
example_5 = 'show flights from cleveland to miami that arrive before 4pm'
gold_sql_5 = """
  SELECT DISTINCT flight_1.flight_id
  FROM flight flight_1 ,
       airport_service airport_service_1 ,
       city city_1 ,
       airport_service airport_service_2 ,
       city city_2
  WHERE flight_1.from_airport = airport_service_1.airport_code
        AND airport_service_1.city_code = city_1.city_code
        AND city_1.city_name = 'CLEVELAND'
        AND ( flight_1.to_airport = airport_service_2.airport_code
              AND airport_service_2.city_code = city_2.city_code
              AND city_2.city_name = 'MIAMI'
              AND flight_1.arrival_time < 1600 )
  """

rule_based_trial(example_5, gold_sql_5)
```

```
Sentence:  show flights from cleveland to miami that arrive before 4pm

Parse:

                               S
```

```
       ______________________|__________________________________________
       |
NP_FLIGHT
       |                                                                      |
       |
NOM_FLIGHT
       |                                                                      |
       |
N_FLIGHT
       |
___________________________|_____________________
       |                                       N_FLIGHT
|
       |                        _________________|_______________
|
       |                   N_FLIGHT                             |
PP
       |             _________|________                         |
|
       |             |                 PP                           PP
PP_TIME
       |             |                 |                            |
_______|__________________
       |             |              PP_PLACE                    PP_PLACE
|                       NP_TIME
       |             |          ________|_________          _______|_________
|                        |
   PREIGNORE      N_FLIGHT     |               N_PLACE      |
N_PLACE          |                     TERM_TIME
       |             |         |                  |         |                 |
|                 _________|__________
PREIGNORESYMBOL TERM_FLIGHT P_PLACE           TERM_PLACE P_PLACE
TERM_PLACE      P_TIME         TERM_TIME           TERM_TIMEMOD
       |             |         |                  |         |                 |
_____|_______         |                    |
      show        flights     from            cleveland     to
miami    that arrive  before     4                   pm
```

Predicted SQL:

```
 SELECT DISTINCT flight.flight_id FROM flight WHERE (((1 AND flight.from_airport
IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "CLEVELAND"))
  ) AND flight.to_airport IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
```

```
      (SELECT city.city_code FROM city WHERE city.city_name = "MIAMI"))
  ) AND flight.arrival_time <= 1600)
```

Predicted DB result:

```
[(107698,), (301117,)]
```

Gold DB result:

```
[(107698,), (301117,)]
```

Correct!

[48]:

```
#TODO: add augmentations to `data/grammar` to make this example work
# Example 6
example_6 = 'okay how about a flight on sunday from tampa to charlotte'
gold_sql_6 = """
  SELECT DISTINCT flight_1.flight_id
  FROM flight flight_1 ,
       airport_service airport_service_1 ,
       city city_1 ,
       airport_service airport_service_2 ,
       city city_2 ,
       days days_1 ,
       date_day date_day_1
  WHERE flight_1.from_airport = airport_service_1.airport_code
        AND airport_service_1.city_code = city_1.city_code
        AND city_1.city_name = 'TAMPA'
        AND ( flight_1.to_airport = airport_service_2.airport_code
              AND airport_service_2.city_code = city_2.city_code
              AND city_2.city_name = 'CHARLOTTE'
              AND flight_1.flight_days = days_1.days_code
              AND days_1.day_name = date_day_1.day_name
              AND date_day_1.year = 1991
              AND date_day_1.month_number = 8
              AND date_day_1.day_number = 27 )
  """

# You might notice that the gold answer above used the exact date, which is
# not easily implementable. A more implementable way (generated by the project
# segment 4 solution code) is:
gold_sql_6b = """
  SELECT DISTINCT flight.flight_id
  FROM flight
  WHERE ((((1
            AND flight.flight_days IN (SELECT days.days_code
                                       FROM days
```

```
                                        WHERE days.day_name = 'SUNDAY')
            )
           AND flight.from_airport IN (SELECT airport_service.airport_code
                                        FROM airport_service
                                        WHERE airport_service.city_code IN␣
 ↪(SELECT city.city_code
                                                                             FROM␣
 ↪city
                                                                            ␣
 ↪WHERE city.city_name = "TAMPA")))
          AND flight.to_airport IN (SELECT airport_service.airport_code
                                     FROM airport_service
                                     WHERE airport_service.city_code IN (SELECT␣
 ↪city.city_code
                                                                          FROM␣
 ↪city
                                                                          WHERE␣
 ↪city.city_name = "CHARLOTTE"))))
 """

rule_based_trial(example_6, gold_sql_6b)
```

```
Sentence:  okay how about a flight on sunday from tampa to charlotte

Parse:


                   S
                                     ___________________________________________
___________________|______________________________
                                  |
NP_FLIGHT
                                  |
|
                                  |
NOM_FLIGHT
                                  |
|
                                  |
N_FLIGHT
                                  |
___________|_____________________________________
                                  |
N_FLIGHT                                           |
                                  |
__________________|____________________                              |
                               PREIGNORE
```

```
N_FLIGHT                                    |                           |
        ____________________________|____________
_________|________                            |                           |
       |                                     PREIGNORE
|                 PP                              PP
PP
       |               _________________________|____________
|                 |                              |                           |
       |              |                                 PREIGNORE
|              PP_DATE                         PP_PLACE
PP_PLACE
       |              |                          ____________|____________
|          ________|_________              _________|_________
_______|_________
       |              |                         |                     PREIGNORE
N_FLIGHT     |              NP_DATE        |              N_PLACE      |
N_PLACE
       |              |                         |                        |
|         |                 |          |                 |         |
|
PREIGNORESYMBOL PREIGNORESYMBOL           PREIGNORESYMBOL
PREIGNORESYMBOL TERM_FLIGHT  P_DATE          TERM_WEEKDAY  P_PLACE
TERM_PLACE P_PLACE          TERM_PLACE
       |              |                         |                        |
|         |                 |          |                 |         |
|
      okay            how                     about                      a
flight       on                sunday       from              tampa       to
charlotte
```

Predicted SQL:

```
 SELECT DISTINCT flight.flight_id FROM flight WHERE (((1 AND flight.flight_days
IN (SELECT days.days_code FROM days WHERE days.day_name = 'SUNDAY')) AND
flight.from_airport IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "TAMPA"))
  ) AND flight.to_airport IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "CHARLOTTE"))
  )
```

Predicted DB result:

```
[(101860,), (101861,), (101862,), (101863,), (101864,), (101865,), (305231,)]
```

Gold DB result:

```
[(101860,), (101861,), (101862,), (101863,), (101864,), (101865,), (305231,)]
```

Correct!

[49]:
```
#TODO: add augmentations to `data/grammar` to make this example work
# Example 7
example_7 = 'list all flights going from boston to atlanta that leaves before 7␣
↪am on thursday'
gold_sql_7 = """
  SELECT DISTINCT flight_1.flight_id
  FROM flight flight_1 ,
       airport_service airport_service_1 ,
       city city_1 ,
       airport_service airport_service_2 ,
       city city_2 ,
       days days_1 ,
       date_day date_day_1
  WHERE flight_1.from_airport = airport_service_1.airport_code
        AND airport_service_1.city_code = city_1.city_code
        AND city_1.city_name = 'BOSTON'
        AND ( flight_1.to_airport = airport_service_2.airport_code
              AND airport_service_2.city_code = city_2.city_code
              AND city_2.city_name = 'ATLANTA'
              AND ( flight_1.flight_days = days_1.days_code
                    AND days_1.day_name = date_day_1.day_name
                    AND date_day_1.year = 1991
                    AND date_day_1.month_number = 5
                    AND date_day_1.day_number = 24
                    AND flight_1.departure_time < 700 ) )
  """

# Again, the gold answer above used the exact date, as opposed to the
# following approach:
gold_sql_7b = """
  SELECT DISTINCT flight.flight_id
  FROM flight
  WHERE ((1
          AND ((((1
                  AND flight.from_airport IN (SELECT airport_service.
↪airport_code
                                              FROM airport_service
                                              WHERE airport_service.city_code␣
↪IN (SELECT city.city_code
                                                                              ␣
↪  FROM city
```

```
↪  WHERE city.city_name = "BOSTON")))
                AND flight.to_airport IN (SELECT airport_service.airport_code
                                          FROM airport_service
                                          WHERE airport_service.city_code IN␣
↪(SELECT city.city_code
                                                                              ␣
↪FROM city
                                                                              ␣
↪WHERE city.city_name = "ATLANTA")))
               AND flight.departure_time <= 0700)
              AND flight.flight_days IN (SELECT days.days_code
                                         FROM days
                                         WHERE days.day_name = 'THURSDAY'))))
 """

rule_based_trial(example_7, gold_sql_7b)
```

```
Sentence:  list all flights going from boston to atlanta that leaves before 7 am
on thursday

Parse:


                                  S
__________________________|_____________________________________________________
       |
NP_FLIGHT
       |         ______________________________________________________________
________|__________
       |         |
NOM_FLIGHT
       |         |
|
       |         |
N_FLIGHT
       |         |
__________|_________________________________________________________________
       |         |
N_FLIGHT                                                                        |
       |         |
_____________________________________|___________________________
|
       |         |                           N_FLIGHT
|                                              |
       |         |                      ________|___________________________
|                                              |
```

```
       |         |                    N_FLIGHT                               |
PP                                               |
       |         |        _______________|________                           |
|                                                |
       |         |       |                        PP                           PP
PP_TIME                                            PP
       |         |       |                        |                            |
_______|__________________                                   |
       |         |       |                     PP_PLACE
PP_PLACE                          |                       NP_TIME
PP_DATE
       |         |       |                ________|_________
_______|_________                     |                          |
_______|_________
   PREIGNORE     |    N_FLIGHT            |                NP_PLACE      |
N_PLACE               |                      TERM_TIME                |
NP_DATE
       |         |       |               |                   |         |
|                    |                ________|__________           |
|
PREIGNORESYMBOL DET TERM_FLIGHT       P_PLACE          TERM_PLACE P_PLACE
TERM_PLACE            P_TIME         TERM_TIME          TERM_TIMEMOD P_DATE
TERM_WEEKDAY
       |         |       |           ______|________           |         |
|           ________|_______         |                   |         |
|
      list      all   flights   going            from     boston      to
atlanta      that    leaves  before     7                    am        on
thursday
```

Predicted SQL:

```
 SELECT DISTINCT flight.flight_id FROM flight WHERE ((((1 AND
flight.from_airport IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "BOSTON"))
  ) AND flight.to_airport IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "ATLANTA"))
  ) AND flight.departure_time <= 700) AND flight.flight_days IN (SELECT
days.days_code FROM days WHERE days.day_name = 'THURSDAY'))
```

Predicted DB result:

```
 [(100014,)]
```

```
Gold DB result:

 [(100014,)]

Correct!
```

[50]:
```python
#TODO: add augmentations to `data/grammar` to make this example work
# Example 8
example_8 = 'list the flights from dallas to san francisco on american airlines'
gold_sql_8 = """
  SELECT DISTINCT flight_1.flight_id
  FROM flight flight_1 ,
       airport_service airport_service_1 ,
       city city_1 ,
       airport_service airport_service_2 ,
       city city_2
  WHERE flight_1.airline_code = 'AA'
        AND ( flight_1.from_airport = airport_service_1.airport_code
              AND airport_service_1.city_code = city_1.city_code
              AND city_1.city_name = 'DALLAS'
              AND flight_1.to_airport = airport_service_2.airport_code
              AND airport_service_2.city_code = city_2.city_code
              AND city_2.city_name = 'SAN FRANCISCO' )
  """

rule_based_trial(example_8, gold_sql_8)
```

```
Sentence:  list the flights from dallas to san francisco on american airlines

Parse:


      S
-----------------------------------------------------------------|--------
                    |
NP_FLIGHT
                    |
|
                    |
NOM_FLIGHT
                    |
|
                    |
N_FLIGHT
                    |
------------------|------------------------------------------
                    |
N_FLIGHT                                                              |
```

```
                    |
_________________|_________________
|
                    |                                  N_FLIGHT
|                                              |
                    |                            _________|________
|                                              |
                    |                            |                  PP
PP                                            PP
                    |                            |                  |
|                                              |
               PREIGNORE                         |              PP_PLACE
PP_PLACE                                    PP_AIRLINE
        ____________|____________                |          ________|_________
________|__________                         ___________|____________
       |                    PREIGNORE      N_FLIGHT     |               N_PLACE
|               N_PLACE                  |                  TERM_AIRLINE
       |                         |             |         |                   |
|                   |                    |
____________|______________
PREIGNORESYMBOL           PREIGNORESYMBOL TERM_FLIGHT P_PLACE
TERM_PLACE P_PLACE            TERM_PLACE          P_AIRLINE TERM_AIRBRAND
TERM_AIRBRANDTYP
       |                         |             |         |                   |
|                   |                    |           |
E
       |                         |             |         |                   |
|         __________|__________          |           |
|
      list                      the          flights     from              dallas
to      san                francisco     on       american
airlines
```

Predicted SQL:

```
 SELECT DISTINCT flight.flight_id FROM flight WHERE (((1 AND flight.from_airport
IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "DALLAS"))
  ) AND flight.to_airport IN
    (SELECT airport_service.airport_code FROM airport_service WHERE
airport_service.city_code IN
      (SELECT city.city_code FROM city WHERE city.city_name = "SAN FRANCISCO"))
  ) AND flight.airline_code = 'AA')
```

Predicted DB result:

```
 [(108452,), (108454,), (108456,), (111083,), (111085,), (111086,), (111090,),
(111091,), (111092,), (111094,)]

Gold DB result:

 [(108452,), (108454,), (108456,), (111083,), (111085,), (111086,), (111090,),
(111091,), (111092,), (111094,)]

Correct!
```

5.1.1 Systematic evaluation on a test set

We can perform a more systematic evaluation by checking the accuracy of the queries on an entire test set for which we have gold queries. The `evaluate` function below does just this, calculating precision, recall, and F1 metrics for the test set. It takes as argument a "predictor" function, which maps token sequences to predicted SQL queries. We've provided a predictor function for the rule-based model in the next cell (and a predictor for the seq2seq system below when we get to that system).

The rule-based system does not generate predictions for all queries; many queries won't parse. The precision and recall metrics take this into account in measuring the efficacy of the method. The recall metric captures what proportion of *all of the test examples* for which the system generates a correct query. The precision metric captures what proportion of *all of the test examples for which a prediction is generated* for which the system generates a correct query. (Recall that F1 is just the geometric mean of precision and recall.)

Once you've made some progress on adding augmentations to the grammar, you can evaluate your progress by seeing if the precision and recall have improved. For reference, the solution code achieves precision of about 66% and recall of about 28% for an F1 of 39%.

[51]:
```python
def evaluate(predictor, dataset, num_examples=0, silent=True):
  """Evaluate accuracy of `predictor` by executing predictions on a
  SQL database and comparing returned results against those of gold queries.

  Arguments:
      predictor:    a function that maps a token sequence (provided by␣
↪torchtext)
                    to a predicted SQL query string
      dataset:      the dataset of token sequences and gold SQL queries
      num_examples: number of examples from `dataset` to use; all of
                    them if 0
      silent: if set to False, will print out logs
  Returns: precision, recall, and F1 score
  """
  # Prepare to count results
  if num_examples <= 0:
    num_examples = len(dataset)
  example_count = 0
  predicted_count = 0
```

```
  correct = 0
  incorrect = 0

  # Process the examples from the dataset
  for example in tqdm(dataset[:num_examples]):
    example_count += 1
    # obtain query SQL
    predicted_sql = predictor(example.src)
    if predicted_sql == None:
      continue
    predicted_count += 1
    # obtain gold SQL
    gold_sql = ' '.join(example.tgt)

    # check that they're compatible
    if verify(predicted_sql, gold_sql):
      correct += 1
    else:
      incorrect += 1

  # Compute and return precision, recall, F1
  precision = correct / predicted_count if predicted_count > 0 else 0
  recall = correct / example_count
  f1 = (2 * precision * recall) / (precision + recall) if precision + recall >␣
↪0 else 0
  return precision, recall, f1
```

[52]:
```
def rule_based_predictor(tokens):
  query = ' '.join(tokens)    # detokenized query
  tree = parse_tree(query)
  if tree is None:
    return None
  try:
    predicted_sql = interpret(tree, atis_augmentations)
  except Exception as err:
    return None
  return predicted_sql
```

[53]:
```
precision, recall, f1 = evaluate(rule_based_predictor, test_iter.dataset,␣
 ↪num_examples=0)
print(f"precision: {precision:3.2f}")
print(f"recall:    {recall:3.2f}")
print(f"F1:        {f1:3.2f}")
```

```
100%|      | 332/332 [00:01<00:00, 191.42it/s]

precision: 0.80
recall:    0.26
```

```
F1:          0.39
```

6 End-to-End Seq2Seq Model

In this part, you will implement a seq2seq model **with attention mechanism** to directly learn the translation from NL query to SQL. You might find labs 4-4 and 4-5 particularly helpful, as the primary difference here is that we are using a different dataset.

Note: We recommend using GPUs to train the model in this part (one way to get GPUs is to use Google Colab and clicking Menu -> Runtime -> Change runtime type -> GPU), as we need to use a very large model to solve the task well. For development we recommend starting with a smaller model and training for only 1 epoch.

6.1 Goal 2: Implement a seq2seq model (with attention)

In lab 4-5, you implemented a neural encoder-decoder model with attention. That model was used to convert English number phrases to numbers, but one of the biggest advantages of neural models is that we can easily apply them to different tasks (such as machine translation and document summarization) by using different training datasets.

Implement the class `AttnEncoderDecoder` to convert natural language queries into SQL statements. You may find that you can reuse most of the code you wrote for lab 4-5. A reasonable way to proceed is to implement the following methods:

- **Model**

    1. `__init__`: an initializer where you create network modules.
    2. `forward`: given source word ids of size `(max_src_len, batch_size)`, source lengths of size `(batch_size)` and decoder input target word ids `(max_tgt_len, batch_size)`, returns logits `(max_tgt_len, batch_size, V_tgt)`. For better modularity you might want to implement it by implementing two functions `forward_encoder` and `forward_decoder`.

- **Optimization**

    3. `train_all`: compute loss on training data, compute gradients, and update model parameters to minimize the loss.
    4. `evaluate_ppl`: evaluate the current model's perplexity on a given dataset iterator, we use the perplexity value on the validation set to select the best model.

- **Decoding**

    5. `predict`: Generates the target sequence given a list of source tokens using beam search decoding. Note that here you can assume the batch size to be 1 for simplicity.

[ ]:
```python
class AttnEncoderDecoder(nn.Module):
    def __init__(self, src_field, tgt_field, hidden_size=64, layers=3):
        """
        Initializer. Creates network modules and loss function.
```

```
        Arguments:
        src_field: src field
        tgt_field: tgt field
        hidden_size: hidden layer size of both encoder and decoder
        layers: number of layers of both encoder and decoder
        """
        super().__init__()
        self.src_field = src_field
        self.tgt_field = tgt_field

        self.V_src = len(src_field.vocab.itos)
        self.V_tgt = len(tgt_field.vocab.itos)

        self.padding_id_src = src_field.vocab.stoi[src_field.pad_token]
        self.padding_id_tgt = tgt_field.vocab.stoi[tgt_field.pad_token]
        self.bos_id = tgt_field.vocab.stoi[tgt_field.init_token]
        self.eos_id = tgt_field.vocab.stoi[tgt_field.eos_token]

        self.embedding_size = hidden_size
        self.hidden_size = hidden_size
        self.layers = layers

        self.word_embeddings_src = nn.Embedding(self.V_src, self.embedding_size)
        self.word_embeddings_tgt = nn.Embedding(self.V_tgt, self.embedding_size)

        self.encoder_rnn = nn.LSTM(
            input_size = self.embedding_size,
            hidden_size = hidden_size // 2, # to match decoder hidden size
            num_layers = layers,
            bidirectional = True # bidirectional encoder
        )
        self.decoder_rnn = nn.LSTM(
            input_size = self.embedding_size,
            hidden_size = hidden_size,
            num_layers = layers,
            bidirectional = False # unidirectional decoder
        )

        self.hidden2output = nn.Linear(2*hidden_size, self.V_tgt) #


        self.loss_function = nn.
↪CrossEntropyLoss(reduction='sum',ignore_index=self.padding_id_tgt)

    def forward_encoder(self, src, src_lengths):
        src_embeddings = self.word_embeddings_src(src)
        src_lengths = src_lengths.tolist()
```

```
        packed_src = pack(src_embeddings, src_lengths)
        all_states, final_state = self.encoder_rnn(packed_src)
        memory_bank, _ = unpack(all_states)
        h, c = final_state


        def reshape_hidden_state(s):
            s = s.reshape(2, self.layers, -1, self.hidden_size // 2)
            s = s.transpose(0, 1).transpose(1, 2)
            s = s.reshape(self.layers, -1, self.hidden_size) #
            return s

        final_state = (reshape_hidden_state(h), reshape_hidden_state(c))
        memory_bank = memory_bank
        final_state = final_state
        context = None
        return memory_bank, (final_state, context)

    def forward_decoder(self, encoder_final_state, tgt_in, memory_bank,␣
↪src_mask):
        max_tgt_length = tgt_in.size(0)
        # Initialize decoder state, note that it's a tuple (state, context) here
        decoder_states = encoder_final_state
        all_logits = []
        for i in range(max_tgt_length):
            logits, decoder_states, attn = self.
↪forward_decoder_incrementally(decoder_states, tgt_in[i], memory_bank,␣
↪src_mask, normalize=False)
            all_logits.append(logits) # list of bsz, vocab_tgt
        all_logits = torch.stack(all_logits, 0) # tgt_len, bsz, vocab_tgt
        return all_logits

    def forward(self, src, src_lengths, tgt_in):
        src_mask = src.ne(self.padding_id_src) # max_src_len, bsz
        memory_bank, encoder_final_state = self.forward_encoder(src,␣
↪src_lengths)
        # Forward decoder
        logits = self.forward_decoder(encoder_final_state, tgt_in, memory_bank,␣
↪src_mask)
        return logits

    def forward_decoder_incrementally(self, prev_decoder_states,␣
↪tgt_in_onestep, memory_bank, src_mask, normalize=True):
        prev_decoder_state, prev_context = prev_decoder_states
        decoder_input = self.word_embeddings_tgt(tgt_in_onestep)
        if prev_context is not None:
            decoder_input = decoder_input + prev_context
```

```
        decoder_input = decoder_input.unsqueeze(0) # 1, bsz, embedding_size
        output, decoder_state = self.decoder_rnn(decoder_input,␣
↪prev_decoder_state)
        q = output
        k = memory_bank
        v = memory_bank
        mask = src_mask.transpose(0, 1).unsqueeze(1)
        attn, context = attention(q, k, v, mask=mask)
        context = context.squeeze(0)
        attn = attn.squeeze(1)
        output = output.squeeze(0)
        outputs = torch.cat([output, context], 1)
        logits = self.hidden2output(outputs)
        decoder_states = (decoder_state, context)
        if normalize:
            logits = torch.log_softmax(logits, dim=-1)
        return logits, decoder_states, attn

    def evaluate_ppl(self, iterator):
        """Returns the model's perplexity on a given dataset `iterator`."""
        self.eval()
        total_loss = 0
        total_words = 0
        for batch in iterator:
            src, src_lengths = batch.src
            tgt = batch.tgt
            tgt_in = tgt[:-1]
            tgt_out = tgt[1:]
            logits = self.forward(src, src_lengths, tgt_in)
            loss = self.loss_function(logits.view(-1, self.V_tgt), tgt_out.
↪view(-1))
            total_loss += loss.item()
            total_words += tgt_out.ne(self.padding_id_tgt).float().sum().item()
        return math.exp(total_loss/total_words)

    def train_all(self, train_iter, val_iter, epochs=10, learning_rate=0.001):
        self.train()
        optim = torch.optim.Adam(self.parameters(), lr=learning_rate)
        best_validation_ppl = float('inf')
        best_model = None
        for epoch in range(epochs):
            total_words = 0
            total_loss = 0.0
            for batch in tqdm(train_iter):
                self.zero_grad()
                src, src_lengths = batch.src
                tgt = batch.tgt
```

```
                tgt_in = tgt[:-1]
                tgt_out = tgt[1:]
                bsz = tgt.size(1)
                logits = self.forward(src, src_lengths, tgt_in)
                loss = self.loss_function(logits.view(-1, self.V_tgt), tgt_out.
↪view(-1))
                num_tgt_words = tgt_out.ne(self.padding_id_tgt).float().sum().
↪item()
                total_words += num_tgt_words
                total_loss += loss.item()
                loss.div(bsz).backward()
                optim.step()
            validation_ppl = self.evaluate_ppl(val_iter)
            self.train()
            if validation_ppl < best_validation_ppl:
                best_validation_ppl = validation_ppl
                self.best_model = copy.deepcopy(self.state_dict())
            epoch_loss = total_loss / total_words
            print (f'Epoch: {epoch} Training Perplexity: {math.exp(epoch_loss):.
↪4f} 'f'Validation Perplexity: {validation_ppl:.4f}')

    def predict(self, tokens, K=1, max_T=400):
        beam_searcher = BeamSearcher(self)
        tokens = torch.tensor([self.src_field.vocab.stoi[token] for token in␣
↪tokens]).int().unsqueeze(1).cuda()
        tokens_length = torch.tensor([len(tokens)]).cuda()
        pred, _ = beam_searcher.beam_search(tokens, tokens_length, K, max_T)
        pred = ' '.join([self.tgt_field.vocab.itos[prediction] for prediction␣
↪in pred if self.tgt_field.vocab.itos[prediction] not in ['<eos>','<bos>']])
        return pred
```

[ ]:
```
def attention(batched_Q, batched_K, batched_V, mask=None):

    # Check sizes
    D = batched_Q.size(-1)
    bsz = batched_Q.size(1)
    q_len = batched_Q.size(0)
    k_len = batched_K.size(0)
    assert batched_K.size(-1) == D and batched_V.size(-1) == D
    assert batched_K.size(1) == bsz and batched_V.size(1) == bsz
    assert batched_V.size(0) == k_len
    if mask is not None:
        assert mask.size() == torch.Size([bsz, q_len, k_len])
    q = batched_Q.transpose(0, 1) # bsz, q_len, hidden
    k = batched_K.transpose(0, 1).transpose(1, 2) # bsz, hidden, k_len
    scores = torch.bmm(q, k) # bsz, q_len, k_len
```

```
    if mask is not None:
        scores = scores.masked_fill(mask == False, -float('inf'))
    batched_A = torch.softmax(scores, dim=-1) # bsz, q_len, k_len
    batched_C = torch.bmm(batched_A, batched_V.transpose(0, 1)) # bsz, q_len, D
    batched_C = batched_C.transpose(0, 1) # q_len, bsz, D
    batched_A = batched_A
    batched_C = batched_C
    assert torch.all(torch.isclose(batched_A.sum(-1),

    torch.ones(bsz, q_len).to(device)))

    return batched_A, batched_C


MAX_T = 15
class Beam():
    def __init__(self, decoder_state, tokens, score):
        self.decoder_state = decoder_state
        self.tokens = tokens
        self.score = score


class BeamSearcher():
    def __init__(self, model):
        self.model = model
        self.bos_id = model.bos_id
        self.eos_id = model.eos_id
        self.padding_id_src = model.padding_id_src
        self.V = model.V_tgt

    def beam_search(self, src, src_lengths, K, max_T=MAX_T):
        finished = []
        all_attns = []
        # Initialize the beam
        self.model.eval()
        memory_bank, encoder_final_state = self.model.
 ↪forward_encoder(src,src_lengths)
        memory_bank = memory_bank
        encoder_final_state = encoder_final_state
        init_beam = Beam(encoder_final_state, [torch.LongTensor(1).fill_(self.
 ↪bos_id).to(device)], 0)
        beams = [init_beam]
        with torch.no_grad():
            for t in range(max_T): # main body of search over time steps
                # Expand each beam by all possible tokens y_{t+1}
                all_total_scores = []
                for beam in beams:
```

```
                    y_1_to_t, score, decoder_state = beam.tokens, beam.score,␣
↪beam.decoder_state
                    y_t = y_1_to_t[-1]
                    src_mask = src.ne(self.padding_id_src)
                    logits, decoder_state, attn = self.model.
↪forward_decoder_incrementally(decoder_state, y_t, memory_bank, src_mask,␣
↪normalize=True)
                    logits = logits
                    decoder_state = decoder_state
                    attn = attn
                    total_scores = logits + score
                    all_total_scores.append(total_scores)
                    all_attns.append(attn)
                    beam.decoder_state = decoder_state
                all_total_scores = torch.stack(all_total_scores)
                all_scores_flattened = all_total_scores.view(-1)
                topk_scores, topk_ids = all_scores_flattened.topk(K, 0)
                beam_ids = topk_ids.div(self.V, rounding_mode='floor')
                next_tokens = topk_ids - beam_ids * self.V
                new_beams = []
                for k in range(K):
                    beam_id = beam_ids[k]
                    y_t_plus_1 = next_tokens[k]
                    score = topk_scores[k]
                    beam = beams[beam_id]
                    decoder_state = beam.decoder_state
                    y_1_to_t = beam.tokens
                    new_beam = Beam(decoder_state, y_1_to_t + [y_t_plus_1],␣
↪score)
                    new_beams.append(new_beam)
                beams = new_beams
                new_beams = []
                for beam in beams:
                    if beam.tokens[-1] == self.eos_id:
                        finished.append(beam)
                    else:
                        new_beams.append(beam)
                beams = new_beams
                if len(beams) == 0:
                    break

        if len(finished) > 0:
            finished = sorted(finished, key=lambda beam: -beam.score)
            return finished[0].tokens, all_attns
        else:
            return beams[0].tokens, all_attns
```

We provide the recommended hyperparameters for the final model in the script below, but you are free to tune the hyperparameters or change any part of the provided code.

> For quick debugging, we recommend starting with smaller models (by using a very small **hidden_size**), and only a single epoch. If the model runs smoothly, then you can train the full model on GPUs.

[ ]:
```
EPOCHS = 50 # epochs; we recommend starting with a smaller number like 1
LEARNING_RATE = 1e-4 # learning rate

# Instantiate and train classifier
model = AttnEncoderDecoder(SRC, TGT,
  hidden_size    = 1024,
  layers         = 1,
).to(device)

model.train_all(train_iter, val_iter, epochs=EPOCHS,␣
 ↪learning_rate=LEARNING_RATE)
model.load_state_dict(model.best_model)

# Evaluate model performance, the expected value should be < 1.2
print (f'Validation perplexity: {model.evaluate_ppl(val_iter):.3f}')
```

```
100%|      | 229/229 [01:17<00:00,  2.96it/s]

Epoch: 0 Training Perplexity: 4.1562 Validation Perplexity: 1.7590

100%|      | 229/229 [01:16<00:00,  2.97it/s]

Epoch: 1 Training Perplexity: 1.4937 Validation Perplexity: 1.4007

100%|      | 229/229 [01:19<00:00,  2.87it/s]

Epoch: 2 Training Perplexity: 1.3047 Validation Perplexity: 1.2931

100%|      | 229/229 [01:18<00:00,  2.94it/s]

Epoch: 3 Training Perplexity: 1.2203 Validation Perplexity: 1.2322

100%|      | 229/229 [01:16<00:00,  2.98it/s]

Epoch: 4 Training Perplexity: 1.1697 Validation Perplexity: 1.1906

100%|      | 229/229 [01:16<00:00,  2.99it/s]

Epoch: 5 Training Perplexity: 1.1363 Validation Perplexity: 1.1620

100%|      | 229/229 [01:19<00:00,  2.88it/s]

Epoch: 6 Training Perplexity: 1.1114 Validation Perplexity: 1.1609

100%|      | 229/229 [01:17<00:00,  2.97it/s]

Epoch: 7 Training Perplexity: 1.0961 Validation Perplexity: 1.1352

100%|      | 229/229 [01:17<00:00,  2.95it/s]
```

```
Epoch: 8 Training Perplexity: 1.0783 Validation Perplexity: 1.1219
100%|      | 229/229 [01:17<00:00,  2.97it/s]
Epoch: 9 Training Perplexity: 1.0658 Validation Perplexity: 1.1176
100%|      | 229/229 [01:18<00:00,  2.93it/s]
Epoch: 10 Training Perplexity: 1.0574 Validation Perplexity: 1.1126
100%|      | 229/229 [01:16<00:00,  2.98it/s]
Epoch: 11 Training Perplexity: 1.0505 Validation Perplexity: 1.1174
100%|      | 229/229 [01:17<00:00,  2.95it/s]
Epoch: 12 Training Perplexity: 1.0444 Validation Perplexity: 1.1047
100%|      | 229/229 [01:16<00:00,  2.98it/s]
Epoch: 13 Training Perplexity: 1.0381 Validation Perplexity: 1.1057
100%|      | 229/229 [01:17<00:00,  2.95it/s]
Epoch: 14 Training Perplexity: 1.0319 Validation Perplexity: 1.1051
100%|      | 229/229 [01:17<00:00,  2.97it/s]
Epoch: 15 Training Perplexity: 1.0283 Validation Perplexity: 1.1035
100%|      | 229/229 [01:16<00:00,  2.99it/s]
Epoch: 16 Training Perplexity: 1.0240 Validation Perplexity: 1.0947
100%|      | 229/229 [01:17<00:00,  2.97it/s]
Epoch: 17 Training Perplexity: 1.0230 Validation Perplexity: 1.1108
100%|      | 229/229 [01:17<00:00,  2.95it/s]
Epoch: 18 Training Perplexity: 1.0271 Validation Perplexity: 1.0949
100%|      | 229/229 [01:17<00:00,  2.97it/s]
Epoch: 19 Training Perplexity: 1.0183 Validation Perplexity: 1.0932
100%|      | 229/229 [01:16<00:00,  2.99it/s]
Epoch: 20 Training Perplexity: 1.0149 Validation Perplexity: 1.0967
100%|      | 229/229 [01:15<00:00,  3.03it/s]
Epoch: 21 Training Perplexity: 1.0142 Validation Perplexity: 1.0904
100%|      | 229/229 [01:17<00:00,  2.95it/s]
Epoch: 22 Training Perplexity: 1.0118 Validation Perplexity: 1.0970
100%|      | 229/229 [01:16<00:00,  2.98it/s]
Epoch: 23 Training Perplexity: 1.0124 Validation Perplexity: 1.0995
100%|      | 229/229 [01:16<00:00,  2.99it/s]
```

```
Epoch: 24 Training Perplexity: 1.0113 Validation Perplexity: 1.0951
100%|     | 229/229 [01:16<00:00,  2.99it/s]
Epoch: 25 Training Perplexity: 1.0112 Validation Perplexity: 1.0915
100%|     | 229/229 [01:17<00:00,  2.97it/s]
Epoch: 26 Training Perplexity: 1.0089 Validation Perplexity: 1.0961
100%|     | 229/229 [01:16<00:00,  2.98it/s]
Epoch: 27 Training Perplexity: 1.0074 Validation Perplexity: 1.0934
100%|     | 229/229 [01:18<00:00,  2.93it/s]
Epoch: 28 Training Perplexity: 1.0056 Validation Perplexity: 1.0969
100%|     | 229/229 [01:16<00:00,  2.99it/s]
Epoch: 29 Training Perplexity: 1.0050 Validation Perplexity: 1.0964
100%|     | 229/229 [01:15<00:00,  3.03it/s]
Epoch: 30 Training Perplexity: 1.0059 Validation Perplexity: 1.1149
100%|     | 229/229 [01:17<00:00,  2.97it/s]
Epoch: 31 Training Perplexity: 1.0105 Validation Perplexity: 1.0991
100%|     | 229/229 [01:17<00:00,  2.96it/s]
Epoch: 32 Training Perplexity: 1.0070 Validation Perplexity: 1.1042
100%|     | 229/229 [01:15<00:00,  3.03it/s]
Epoch: 33 Training Perplexity: 1.0063 Validation Perplexity: 1.0961
100%|     | 229/229 [01:17<00:00,  2.96it/s]
Epoch: 34 Training Perplexity: 1.0043 Validation Perplexity: 1.0937
100%|     | 229/229 [01:16<00:00,  2.98it/s]
Epoch: 35 Training Perplexity: 1.0033 Validation Perplexity: 1.0958
100%|     | 229/229 [01:15<00:00,  3.03it/s]
Epoch: 36 Training Perplexity: 1.0025 Validation Perplexity: 1.0996
100%|     | 229/229 [01:17<00:00,  2.96it/s]
Epoch: 37 Training Perplexity: 1.0025 Validation Perplexity: 1.0979
100%|     | 229/229 [01:17<00:00,  2.97it/s]
Epoch: 38 Training Perplexity: 1.0075 Validation Perplexity: 1.1028
100%|     | 229/229 [01:17<00:00,  2.97it/s]
Epoch: 39 Training Perplexity: 1.0110 Validation Perplexity: 1.1009
100%|     | 229/229 [01:18<00:00,  2.93it/s]
```

```
Epoch: 40 Training Perplexity: 1.0074 Validation Perplexity: 1.1047

100%|      | 229/229 [01:17<00:00,  2.96it/s]

Epoch: 41 Training Perplexity: 1.0048 Validation Perplexity: 1.1000

100%|      | 229/229 [01:16<00:00,  2.98it/s]

Epoch: 42 Training Perplexity: 1.0031 Validation Perplexity: 1.1172

100%|      | 229/229 [01:16<00:00,  2.98it/s]

Epoch: 43 Training Perplexity: 1.0038 Validation Perplexity: 1.1003

100%|      | 229/229 [01:15<00:00,  3.02it/s]

Epoch: 44 Training Perplexity: 1.0028 Validation Perplexity: 1.1002

100%|      | 229/229 [01:16<00:00,  2.99it/s]

Epoch: 45 Training Perplexity: 1.0018 Validation Perplexity: 1.1026

100%|      | 229/229 [01:16<00:00,  3.01it/s]

Epoch: 46 Training Perplexity: 1.0019 Validation Perplexity: 1.1004

100%|      | 229/229 [01:17<00:00,  2.96it/s]

Epoch: 47 Training Perplexity: 1.0037 Validation Perplexity: 1.1033

100%|      | 229/229 [01:16<00:00,  2.99it/s]

Epoch: 48 Training Perplexity: 1.0070 Validation Perplexity: 1.1089

100%|      | 229/229 [01:15<00:00,  3.01it/s]

Epoch: 49 Training Perplexity: 1.0067 Validation Perplexity: 1.1031
Validation perplexity: 1.090
```

With a trained model, we can convert questions to SQL statements. We recommend making sure that the model can generate at least reasonable results on the examples from before, before evaluating on the full test set.

[ ]:
```
def seq2seq_trial(sentence, gold_sql):
  print("Sentence: ", sentence, "\n")
  tokens = tokenize(sentence)
  predicted_sql = model.predict(tokens, K=1, max_T=400)
  print("Predicted SQL:\n\n", predicted_sql, "\n")

  if verify(predicted_sql, gold_sql, silent=False):
    print ('Correct!')
  else:
    print ('Incorrect!')
```

[ ]:
```
seq2seq_trial(example_1, gold_sql_1)
```

```
Sentence:  flights from phoenix to milwaukee

Predicted SQL:

 SELECT DISTINCT flight_1.flight_id FROM flight flight_1 , airport_service
airport_service_1 , city city_1 , airport_service airport_service_2 , city
city_2 WHERE flight_1.from_airport = airport_service_1.airport_code AND
airport_service_1.city_code = city_1.city_code AND city_1.city_name = 'PHOENIX'
AND flight_1.to_airport = airport_service_2.airport_code AND
airport_service_2.city_code = city_2.city_code AND city_2.city_name =
'MILWAUKEE'

Predicted DB result:

 [(108086,), (108087,), (301763,), (301764,), (301765,), (301766,), (302323,),
(304881,), (310619,), (310620,)]

Gold DB result:

 [(108086,), (108087,), (301763,), (301764,), (301765,), (301766,), (302323,),
(304881,), (310619,), (310620,)]

Correct!
```

[ ]:
```
seq2seq_trial(example_2, gold_sql_2)
```

```
Sentence:  i would like a united flight

Predicted SQL:

 SELECT DISTINCT flight_1.flight_id FROM flight flight_1 , airport_service
airport_service_1 , city city_1 WHERE flight_1.airline_code = 'UA' AND (
flight_1.from_airport = airport_service_1.airport_code AND
airport_service_1.city_code = city_1.city_code AND city_1.city_name = 'DENVER'
AND flight_1.to_airport = airport_service_2.airport_code AND
airport_service_2.city_code = city_2.city_code AND city_2.city_name = 'DENVER' )

predicted sql exec failed: no such column: airport_service_2.airport_code
Incorrect!
```

[ ]:
```
seq2seq_trial(example_3, gold_sql_3)
```

```
Sentence:  i would like a flight between boston and dallas

Predicted SQL:

 SELECT DISTINCT flight_1.flight_id FROM flight flight_1 , airport_service
airport_service_1 , city city_1 , airport_service airport_service_2 , city
city_2 WHERE flight_1.from_airport = airport_service_1.airport_code AND
```

airport_service_1.city_code = city_1.city_code AND city_1.city_name = 'BOSTON'
AND flight_1.to_airport = airport_service_2.airport_code AND
airport_service_2.city_code = city_2.city_code AND city_2.city_name = 'DALLAS'

Predicted DB result:

[(103171,), (103172,), (103173,), (103174,), (103175,), (103176,), (103177,),
(103178,), (103179,), (103180,)]

Gold DB result:

[(103171,), (103172,), (103173,), (103174,), (103175,), (103176,), (103177,),
(103178,), (103179,), (103180,)]

Correct!

[ ]:
```
seq2seq_trial(example_4, gold_sql_4)
```

Sentence: show me the united flights from denver to baltimore

Predicted SQL:

SELECT DISTINCT flight_1.flight_id FROM flight flight_1 , airport_service
airport_service_1 , city city_1 , airport_service airport_service_2 , city
city_2 WHERE flight_1.airline_code = 'UA' AND (flight_1.from_airport =
airport_service_1.airport_code AND airport_service_1.city_code =
city_1.city_code AND city_1.city_name = 'DENVER' AND flight_1.to_airport =
airport_service_2.airport_code AND airport_service_2.city_code =
city_2.city_code AND city_2.city_name = 'BALTIMORE')

Predicted DB result:

[(101231,), (101233,), (305983,)]

Gold DB result:

[(101231,), (101233,), (305983,)]

Correct!

[ ]:
```
seq2seq_trial(example_5, gold_sql_5)
```

Sentence: show flights from cleveland to miami that arrive before 4pm

Predicted SQL:

SELECT DISTINCT flight_1.flight_id FROM flight flight_1 , airport_service
airport_service_1 , city city_1 , airport_service airport_service_2 , city
city_2 WHERE flight_1.from_airport = airport_service_1.airport_code AND

airport_service_1.city_code = city_1.city_code AND city_1.city_name = 'CLEVELAND' AND (flight_1.to_airport = airport_service_2.airport_code AND airport_service_2.city_code = city_2.city_code AND city_2.city_name = 'MIAMI' AND flight_1.arrival_time < 1600)

Predicted DB result:

[(107698,), (301117,)]

Gold DB result:

[(107698,), (301117,)]

Correct!

[ ]:
```
seq2seq_trial(example_6, gold_sql_6b)
```

Sentence: okay how about a flight on sunday from tampa to charlotte

Predicted SQL:

SELECT DISTINCT flight_1.flight_id FROM flight flight_1 , airport_service airport_service_1 , city city_1 , airport_service airport_service_2 , city city_2 , days days_1 , date_day date_day_1 WHERE flight_1.from_airport = airport_service_1.airport_code AND airport_service_1.city_code = city_1.city_code AND city_1.city_name = 'TAMPA' AND (flight_1.to_airport = airport_service_2.airport_code AND airport_service_2.city_code = city_2.city_code AND city_2.city_name = 'CHARLOTTE' AND flight_1.flight_days = days_1.days_code AND days_1.day_name = date_day_1.day_name AND date_day_1.year = 1991 AND date_day_1.month_number = 8 AND date_day_1.day_number = 27)

Predicted DB result:

[(101860,), (101861,), (101862,), (101863,), (101864,), (101865,), (305231,)]

Gold DB result:

[(101860,), (101861,), (101862,), (101863,), (101864,), (101865,), (305231,)]

Correct!

[ ]:
```
seq2seq_trial(example_7, gold_sql_7b)
```

Sentence: list all flights going from boston to atlanta that leaves before 7 am on thursday

Predicted SQL:

SELECT DISTINCT flight_1.flight_id FROM flight flight_1 , airport_service airport_service_1 , city city_1 , airport_service airport_service_2 , city city_2 , days days_1 , date_day date_day_1 WHERE flight_1.from_airport = airport_service_1.airport_code AND airport_service_1.city_code = city_1.city_code AND city_1.city_name = 'BOSTON' AND (flight_1.to_airport = airport_service_2.airport_code AND airport_service_2.city_code = city_2.city_code AND city_2.city_name = 'ATLANTA' AND (flight_1.flight_days = days_1.days_code AND days_1.day_name = date_day_1.day_name AND date_day_1.year = 1991 AND date_day_1.month_number = 5 AND date_day_1.day_number = 24 AND flight_1.departure_time < 700))

Predicted DB result:

[(100014,)]

Gold DB result:

[(100014,)]

Correct!

```
[ ]: seq2seq_trial(example_8, gold_sql_8)
```

Sentence: list the flights from dallas to san francisco on american airlines

Predicted SQL:

SELECT DISTINCT flight_1.flight_id FROM flight flight_1 , airport_service airport_service_1 , city city_1 , airport_service airport_service_2 , city city_2 WHERE flight_1.airline_code = 'AA' AND (flight_1.from_airport = airport_service_1.airport_code AND airport_service_1.city_code = city_1.city_code AND city_1.city_name = 'DALLAS' AND flight_1.to_airport = airport_service_2.airport_code AND airport_service_2.city_code = city_2.city_code AND city_2.city_name = 'SAN FRANCISCO')

Predicted DB result:

[(108452,), (108454,), (108456,), (111083,), (111085,), (111086,), (111090,), (111091,), (111092,), (111094,)]

Gold DB result:

[(108452,), (108454,), (108456,), (111083,), (111085,), (111086,), (111090,), (111091,), (111092,), (111094,)]

Correct!

6.1.1 Evaluation

Now we are ready to run the full evaluation. A proper implementation should reach more than 35% precision/recall/F1.

[ ]:
```
def seq2seq_predictor(tokens):
  prediction = model.predict(tokens, K=1, max_T=400)
  return prediction
```

[ ]:
```
precision, recall, f1 = evaluate(seq2seq_predictor, test_iter.dataset,␣
 ↪num_examples=0)
print(f"precision: {precision:3.2f}")
print(f"recall:    {recall:3.2f}")
print(f"F1:        {f1:3.2f}")
```

```
100%|      | 332/332 [01:22<00:00,  4.05it/s]
```

```
precision: 0.43
recall:    0.43
F1:        0.43
```

6.2 Goal 3: Implement a seq2seq model (with cross attention and self attention)

In the previous section, you have implemented a seq2seq model with attention. The attention mechanism used in that section is usually referred to as "cross-attention", as at each decoding step, the decoder attends to encoder outputs, enabling a dynamic view on the encoder side as decoding proceeds.

Similarly, we can have a dynamic view on the decoder side as well as decoding proceeds, i.e., the decoder attends to decoder outputs at previous steps. This is called "self attention", and has been found very useful in modern neural architectures such as transformers.

Augment the seq2seq model you implemented before with a decoder self-attention mechanism as class `AttnEncoderDecoder2`. A model diagram can be found below:

At each decoding step, the decoder LSTM first produces an output state o_t, then it attends to all previous output states $o_1, \ldots, o_{t-1}$ (decoder self-attention). You need to special case the first decoding step to not perform self-attention, as there are no previous decoder states. The attention result is added to o_t itself and the sum is used as q_t to attend to the encoder side (encoder-decoder cross-attention). The rest of the model is the same as encoder-decoder with attention.

[81]:
```
#TODO - implement the `AttnEncoderDecoder2` class.
class AttnEncoderDecoder2(nn.Module):
    def __init__(self, src_field, tgt_field, hidden_size=64, layers=3):
        """
        Initializer. Creates network modules and loss function.
        Arguments:
        src_field: src field
        tgt_field: tgt field
```

```
        hidden_size: hidden layer size of both encoder and decoder
        layers: number of layers of both encoder and decoder
        """
        super().__init__()
        self.src_field = src_field
        self.tgt_field = tgt_field

        self.V_src = len(src_field.vocab.itos)
        self.V_tgt = len(tgt_field.vocab.itos)

        self.padding_id_src = src_field.vocab.stoi[src_field.pad_token]
        self.padding_id_tgt = tgt_field.vocab.stoi[tgt_field.pad_token]
        self.bos_id = tgt_field.vocab.stoi[tgt_field.init_token]
        self.eos_id = tgt_field.vocab.stoi[tgt_field.eos_token]

        self.embedding_size = hidden_size
        self.hidden_size = hidden_size
        self.layers = layers

        self.word_embeddings_src = nn.Embedding(self.V_src, self.embedding_size)
        self.word_embeddings_tgt = nn.Embedding(self.V_tgt, self.embedding_size)

        self.encoder_rnn = nn.LSTM(
            input_size = self.embedding_size,
            hidden_size = hidden_size // 2, # to match decoder hidden size
            num_layers = layers,
            bidirectional = True # bidirectional encoder
        )
        self.decoder_rnn = nn.LSTM(
            input_size = self.embedding_size,
            hidden_size = hidden_size,
            num_layers = layers,
            bidirectional = False # unidirectional decoder
        )

        self.hidden2output = nn.Linear(2*hidden_size, self.V_tgt) #
        self.loss_function = nn.
↪CrossEntropyLoss(reduction='sum',ignore_index=self.padding_id_tgt)

    def forward_encoder(self, src, src_lengths):
        src_embeddings = self.word_embeddings_src(src)
        src_lengths = src_lengths.tolist()
        packed_src = pack(src_embeddings, src_lengths)
        all_states, final_state = self.encoder_rnn(packed_src)
        memory_bank, _ = unpack(all_states)
        h, c = final_state
```

```
        def reshape_hidden_state(s):
            s = s.reshape(2, self.layers, -1, self.hidden_size // 2)
            s = s.transpose(0, 1).transpose(1, 2)
            s = s.reshape(self.layers, -1, self.hidden_size) #
            return s

        final_state = (reshape_hidden_state(h), reshape_hidden_state(c))
        memory_bank = memory_bank
        final_state = final_state
        context = None
        return memory_bank, (final_state, context)

    def forward_decoder(self, encoder_final_state, tgt_in, memory_bank,␣
↪src_mask):
        max_tgt_length = tgt_in.size(0)
        # Initialize decoder state, note that it's a tuple (state, context) here
        decoder_states = encoder_final_state
        all_logits = []
        self_att = None
        for i in range(max_tgt_length):
            logits, decoder_states, attn, self_att = self.
↪forward_decoder_incrementally(decoder_states, tgt_in[i], memory_bank,␣
↪src_mask, self_att, normalize=False)
            all_logits.append(logits) # list of bsz, vocab_tgt
        all_logits = torch.stack(all_logits, 0) # tgt_len, bsz, vocab_tgt
        return all_logits

    def forward(self, src, src_lengths, tgt_in):
        src_mask = src.ne(self.padding_id_src) # max_src_len, bsz
        memory_bank, encoder_final_state = self.forward_encoder(src,␣
↪src_lengths)
        # Forward decoder
        logits = self.forward_decoder(encoder_final_state, tgt_in, memory_bank,␣
↪src_mask)
        return logits

    def forward_decoder_incrementally(self, prev_decoder_states,␣
↪tgt_in_onestep, memory_bank, src_mask, self_att, normalize=True):

        prev_decoder_state, prev_context = prev_decoder_states
        decoder_input = self.word_embeddings_tgt(tgt_in_onestep)
        if prev_context is not None:
            decoder_input = decoder_input + prev_context
        decoder_input = decoder_input.unsqueeze(0) # 1, bsz, embedding_size
        output, decoder_state = self.decoder_rnn(decoder_input,␣
↪prev_decoder_state)
```

```
        if self_att is not None:
            batch_a, batch_c = attention_goal3(output, self_att, self_att)
            self_att = torch.cat([self_att, output], 0)
            q = output + batch_c
        else:
            self_att = output
            q = output

        k = memory_bank
        v = memory_bank
        mask = src_mask.transpose(0, 1).unsqueeze(1)
        attn, context = attention_goal3(q, k, v, mask=mask)
        context = context.squeeze(0)
        attn = attn.squeeze(1)

        output = output.squeeze(0)
        outputs = torch.cat([output, context], 1)

        logits = self.hidden2output(outputs)

        decoder_states = (decoder_state, context)
        if normalize:
            logits = torch.log_softmax(logits, dim=-1)
        return logits, decoder_states, attn, self_att

    def evaluate_ppl(self, iterator):
        """Returns the model's perplexity on a given dataset `iterator`."""
        self.eval()
        total_loss = 0
        total_words = 0
        for batch in iterator:
            src, src_lengths = batch.src
            tgt = batch.tgt
            tgt_in = tgt[:-1]
            tgt_out = tgt[1:]
            logits = self.forward(src, src_lengths, tgt_in)
            loss = self.loss_function(logits.view(-1, self.V_tgt), tgt_out.
↪view(-1))
            total_loss += loss.item()
            total_words += tgt_out.ne(self.padding_id_tgt).float().sum().item()
        return math.exp(total_loss/total_words)

    def train_all(self, train_iter, val_iter, epochs=10, learning_rate=0.001):
        self.train()
        optim = torch.optim.Adam(self.parameters(), lr=learning_rate)
```

```
        best_validation_ppl = float('inf')
        best_model = None
        for epoch in range(epochs):
            total_words = 0
            total_loss = 0.0
            for batch in tqdm(train_iter):
                self.zero_grad()
                src, src_lengths = batch.src
                tgt = batch.tgt
                tgt_in = tgt[:-1]
                tgt_out = tgt[1:]
                bsz = tgt.size(1)
                logits = self.forward(src, src_lengths, tgt_in)
                loss = self.loss_function(logits.view(-1, self.V_tgt), tgt_out.
↪view(-1))
                num_tgt_words = tgt_out.ne(self.padding_id_tgt).float().sum().
↪item()
                total_words += num_tgt_words
                total_loss += loss.item()
                loss.div(bsz).backward()
                optim.step()
            validation_ppl = self.evaluate_ppl(val_iter)
            self.train()
            if validation_ppl < best_validation_ppl:
                best_validation_ppl = validation_ppl
                self.best_model = copy.deepcopy(self.state_dict())
            epoch_loss = total_loss / total_words
            print (f'Epoch: {epoch} Training Perplexity: {math.exp(epoch_loss):.
↪4f} 'f'Validation Perplexity: {validation_ppl:.4f}')

    def predict(self, tokens, K=1, max_T=400):
        beam_searcher = BeamSearcher_goal3(self)
        tokens = torch.tensor([self.src_field.vocab.stoi[token] for token in␣
↪tokens]).int().unsqueeze(1).cuda()
        tokens_length = torch.tensor([len(tokens)]).cuda()
        pred, _ = beam_searcher.beam_search(tokens, tokens_length, K, max_T)
        pred = ' '.join([self.tgt_field.vocab.itos[prediction] for prediction␣
↪in pred if self.tgt_field.vocab.itos[prediction] not in ['<eos>','<bos>']])
        return pred
```

[82]:

```
def attention_goal3(batched_Q, batched_K, batched_V, mask=None):

    # Check sizes
    D = batched_Q.size(-1)
    bsz = batched_Q.size(1)
    q_len = batched_Q.size(0)
    k_len = batched_K.size(0)
```

```
    if mask is not None:
        assert mask.size() == torch.Size([bsz, q_len, k_len])
    q = batched_Q.transpose(0, 1) # bsz, q_len, hidden
    k = batched_K.transpose(0, 1).transpose(1, 2) # bsz, hidden, k_len
    scores = torch.bmm(q, k) # bsz, q_len, k_len
    if mask is not None:
        scores = scores.masked_fill(mask == False, -float('inf'))
    batched_A = torch.softmax(scores, dim=-1) # bsz, q_len, k_len
    batched_C = torch.bmm(batched_A, batched_V.transpose(0, 1)) # bsz, q_len, D
    batched_C = batched_C.transpose(0, 1) # q_len, bsz, D
    batched_A = batched_A
    batched_C = batched_C
    return batched_A, batched_C


MAX_T = 15
class Beam_goal3():
    def __init__(self, decoder_state, tokens, score, self_att):
        self.decoder_state = decoder_state
        self.tokens = tokens
        self.score = score
        self.self_att = self_att


class BeamSearcher_goal3():
    def __init__(self, model):
        self.model = model
        self.bos_id = model.bos_id
        self.eos_id = model.eos_id
        self.padding_id_src = model.padding_id_src
        self.V = model.V_tgt

    def beam_search(self, src, src_lengths, K, max_T=MAX_T):
        finished = []
        all_attns = []
        # Initialize the beam
        self.model.eval()
        memory_bank, encoder_final_state = self.model.
 ↪forward_encoder(src,src_lengths)
        memory_bank = memory_bank
        encoder_final_state = encoder_final_state
        init_beam = Beam_goal3(encoder_final_state, [torch.LongTensor(1).
 ↪fill_(self.bos_id).to(device)], 0, None)
        beams = [init_beam]
        with torch.no_grad():
            for t in range(max_T): # main body of search over time steps
                # Expand each beam by all possible tokens y_{t+1}
```

```python
                all_total_scores = []
                for beam in beams:
                    y_1_to_t, score, decoder_state, self_att = beam.tokens,␣
↪beam.score, beam.decoder_state, beam.self_att
                    y_t = y_1_to_t[-1]
                    src_mask = src.ne(self.padding_id_src)
                    logits, decoder_state, attn, self_att = self.model.
↪forward_decoder_incrementally(decoder_state, y_t, memory_bank, src_mask,␣
↪self_att, normalize=True)
                    logits = logits
                    decoder_state = decoder_state
                    attn = attn
                    total_scores = logits + score
                    all_total_scores.append(total_scores)
                    all_attns.append(attn)
                    beam.decoder_state = decoder_state
                all_total_scores = torch.stack(all_total_scores)
                all_scores_flattened = all_total_scores.view(-1)
                topk_scores, topk_ids = all_scores_flattened.topk(K, 0)
                beam_ids = topk_ids.div(self.V, rounding_mode='floor')
                next_tokens = topk_ids - beam_ids * self.V
                new_beams = []
                for k in range(K):
                    beam_id = beam_ids[k] # which beam it comes from
                    y_t_plus_1 = next_tokens[k] # which y_{t+1}
                    score = topk_scores[k]
                    beam = beams[beam_id]
                    decoder_state = beam.decoder_state
                    y_1_to_t = beam.tokens
                    self_att = beam.self_att
                    new_beam = Beam_goal3(decoder_state, y_1_to_t +␣
↪[y_t_plus_1], score, self_att)
                    new_beams.append(new_beam)
                beams = new_beams

                new_beams = []
                for beam in beams:
                    if beam.tokens[-1] == self.eos_id:
                        finished.append(beam)
                    else:
                        new_beams.append(beam)
                beams = new_beams

                if len(beams) == 0:
                    break
        # Return the best hypothesis
        if len(finished) > 0:
```

```
            finished = sorted(finished, key=lambda beam: -beam.score)
            return finished[0].tokens, all_attns
        else: # when nothing is finished, return an unfinished hypothesis
            return beams[0].tokens, all_attns
```

[86]:
```
EPOCHS = 50 # epochs, we recommend starting with a smaller number like 1
LEARNING_RATE = 1e-4 # learning rate

# Instantiate and train classifier
model2 = AttnEncoderDecoder2(SRC, TGT,
  hidden_size    = 1024,
  layers         = 1,
).to(device)

model2.train_all(train_iter, val_iter, epochs=EPOCHS,␣
 ↪learning_rate=LEARNING_RATE)
model2.load_state_dict(model2.best_model)

# Evaluate model performance, the expected value should be < 1.2
print (f'Validation perplexity: {model2.evaluate_ppl(val_iter):.3f}')
```

```
100%|      | 229/229 [01:26<00:00,  2.66it/s]

Epoch: 0 Training Perplexity: 4.2439 Validation Perplexity: 1.8106

100%|      | 229/229 [01:26<00:00,  2.65it/s]

Epoch: 1 Training Perplexity: 1.5466 Validation Perplexity: 1.4401

100%|      | 229/229 [01:27<00:00,  2.61it/s]

Epoch: 2 Training Perplexity: 1.3394 Validation Perplexity: 1.3293

100%|      | 229/229 [01:26<00:00,  2.65it/s]

Epoch: 3 Training Perplexity: 1.2544 Validation Perplexity: 1.2550

100%|      | 229/229 [01:28<00:00,  2.58it/s]

Epoch: 4 Training Perplexity: 1.1949 Validation Perplexity: 1.2240

100%|      | 229/229 [01:25<00:00,  2.67it/s]

Epoch: 5 Training Perplexity: 1.1624 Validation Perplexity: 1.1844

100%|      | 229/229 [01:25<00:00,  2.66it/s]

Epoch: 6 Training Perplexity: 1.1395 Validation Perplexity: 1.1694

100%|      | 229/229 [01:26<00:00,  2.65it/s]

Epoch: 7 Training Perplexity: 1.1164 Validation Perplexity: 1.1441

100%|      | 229/229 [01:26<00:00,  2.66it/s]

Epoch: 8 Training Perplexity: 1.0972 Validation Perplexity: 1.1422
```

```
100%|      | 229/229 [01:26<00:00,  2.64it/s]
Epoch: 9 Training Perplexity: 1.0896 Validation Perplexity: 1.1281
100%|      | 229/229 [01:26<00:00,  2.65it/s]
Epoch: 10 Training Perplexity: 1.0750 Validation Perplexity: 1.1189
100%|      | 229/229 [01:27<00:00,  2.63it/s]
Epoch: 11 Training Perplexity: 1.0671 Validation Perplexity: 1.1150
100%|      | 229/229 [01:26<00:00,  2.66it/s]
Epoch: 12 Training Perplexity: 1.0619 Validation Perplexity: 1.1153
100%|      | 229/229 [01:27<00:00,  2.62it/s]
Epoch: 13 Training Perplexity: 1.0519 Validation Perplexity: 1.1248
100%|      | 229/229 [01:25<00:00,  2.68it/s]
Epoch: 14 Training Perplexity: 1.0488 Validation Perplexity: 1.1014
100%|      | 229/229 [01:25<00:00,  2.69it/s]
Epoch: 15 Training Perplexity: 1.0393 Validation Perplexity: 1.0981
100%|      | 229/229 [01:26<00:00,  2.66it/s]
Epoch: 16 Training Perplexity: 1.0343 Validation Perplexity: 1.0949
100%|      | 229/229 [01:25<00:00,  2.69it/s]
Epoch: 17 Training Perplexity: 1.0297 Validation Perplexity: 1.0959
100%|      | 229/229 [01:25<00:00,  2.67it/s]
Epoch: 18 Training Perplexity: 1.0291 Validation Perplexity: 1.1017
100%|      | 229/229 [01:25<00:00,  2.69it/s]
Epoch: 19 Training Perplexity: 1.0268 Validation Perplexity: 1.1007
100%|      | 229/229 [01:26<00:00,  2.66it/s]
Epoch: 20 Training Perplexity: 1.0237 Validation Perplexity: 1.1029
100%|      | 229/229 [01:24<00:00,  2.71it/s]
Epoch: 21 Training Perplexity: 1.0263 Validation Perplexity: 1.0925
100%|      | 229/229 [01:26<00:00,  2.66it/s]
Epoch: 22 Training Perplexity: 1.0184 Validation Perplexity: 1.0898
100%|      | 229/229 [01:24<00:00,  2.71it/s]
Epoch: 23 Training Perplexity: 1.0153 Validation Perplexity: 1.0940
100%|      | 229/229 [01:26<00:00,  2.66it/s]
Epoch: 24 Training Perplexity: 1.0149 Validation Perplexity: 1.1015
```

```
100%|      | 229/229 [01:26<00:00,  2.66it/s]
Epoch: 25 Training Perplexity: 1.0136 Validation Perplexity: 1.0914
100%|      | 229/229 [01:24<00:00,  2.70it/s]
Epoch: 26 Training Perplexity: 1.0128 Validation Perplexity: 1.0889
100%|      | 229/229 [01:26<00:00,  2.65it/s]
Epoch: 27 Training Perplexity: 1.0120 Validation Perplexity: 1.0907
100%|      | 229/229 [01:25<00:00,  2.69it/s]
Epoch: 28 Training Perplexity: 1.0117 Validation Perplexity: 1.0915
100%|      | 229/229 [01:25<00:00,  2.67it/s]
Epoch: 29 Training Perplexity: 1.0110 Validation Perplexity: 1.1031
100%|      | 229/229 [01:23<00:00,  2.73it/s]
Epoch: 30 Training Perplexity: 1.0153 Validation Perplexity: 1.0973
100%|      | 229/229 [01:26<00:00,  2.64it/s]
Epoch: 31 Training Perplexity: 1.0095 Validation Perplexity: 1.0915
100%|      | 229/229 [01:25<00:00,  2.69it/s]
Epoch: 32 Training Perplexity: 1.0075 Validation Perplexity: 1.0917
100%|      | 229/229 [01:25<00:00,  2.67it/s]
Epoch: 33 Training Perplexity: 1.0054 Validation Perplexity: 1.0933
100%|      | 229/229 [01:26<00:00,  2.65it/s]
Epoch: 34 Training Perplexity: 1.0055 Validation Perplexity: 1.0938
100%|      | 229/229 [01:24<00:00,  2.72it/s]
Epoch: 35 Training Perplexity: 1.0046 Validation Perplexity: 1.0954
100%|      | 229/229 [01:26<00:00,  2.65it/s]
Epoch: 36 Training Perplexity: 1.0056 Validation Perplexity: 1.0953
100%|      | 229/229 [01:25<00:00,  2.69it/s]
Epoch: 37 Training Perplexity: 1.0111 Validation Perplexity: 1.0968
100%|      | 229/229 [01:27<00:00,  2.62it/s]
Epoch: 38 Training Perplexity: 1.0093 Validation Perplexity: 1.0924
100%|      | 229/229 [01:26<00:00,  2.64it/s]
Epoch: 39 Training Perplexity: 1.0054 Validation Perplexity: 1.0914
100%|      | 229/229 [01:25<00:00,  2.67it/s]
Epoch: 40 Training Perplexity: 1.0033 Validation Perplexity: 1.0929
```

```
100%|          | 229/229 [01:25<00:00,  2.66it/s]

Epoch: 41 Training Perplexity: 1.0027 Validation Perplexity: 1.0968

100%|          | 229/229 [01:25<00:00,  2.68it/s]

Epoch: 42 Training Perplexity: 1.0040 Validation Perplexity: 1.1015

100%|          | 229/229 [01:26<00:00,  2.65it/s]

Epoch: 43 Training Perplexity: 1.0074 Validation Perplexity: 1.0991

100%|          | 229/229 [01:24<00:00,  2.70it/s]

Epoch: 44 Training Perplexity: 1.0106 Validation Perplexity: 1.0950

100%|          | 229/229 [01:25<00:00,  2.67it/s]

Epoch: 45 Training Perplexity: 1.0070 Validation Perplexity: 1.0922

100%|          | 229/229 [01:25<00:00,  2.67it/s]

Epoch: 46 Training Perplexity: 1.0043 Validation Perplexity: 1.0955

100%|          | 229/229 [01:26<00:00,  2.65it/s]

Epoch: 47 Training Perplexity: 1.0044 Validation Perplexity: 1.0961

100%|          | 229/229 [01:26<00:00,  2.63it/s]

Epoch: 48 Training Perplexity: 1.0034 Validation Perplexity: 1.0936

100%|          | 229/229 [01:25<00:00,  2.67it/s]

Epoch: 49 Training Perplexity: 1.0025 Validation Perplexity: 1.0950
Validation perplexity: 1.089
```

6.2.1 Evaluation

Now we are ready to run the full evaluation. A proper implementation should reach more than 35% precision/recall/F1.

[87]:
```
def seq2seq_predictor2(tokens):
  prediction = model2.predict(tokens, K=1, max_T=400)
  return prediction
```

[88]:
```
precision, recall, f1 = evaluate(seq2seq_predictor2, test_iter.dataset,␣
 ↪num_examples=0)
print(f"precision: {precision:3.2f}")
print(f"recall:    {recall:3.2f}")
print(f"F1:        {f1:3.2f}")
```

```
100%|          | 332/332 [00:55<00:00,  5.93it/s]

precision: 0.41
recall:    0.41
F1:        0.41
```

6.3 Goal 4: Use state-of-the-art pretrained transformers

The most recent breakthrough in natural-language processing stems from the use of pretrained transformer models. For example, you might have heard of pretrained transformers such as GPT-3 and BERT. (BERT is already used in Google search.) These models are usually trained on vast amounts of text data using variants of language modeling objectives, and researchers have found that finetuning them on downstream tasks usually results in better performance as compared to training a model from scratch.

In the previous part, you implemented an LSTM-based sequence-to-sequence approach. To "upgrade" the model to be a state-of-the-art pretrained transformer only requires minor modifications.

The pretrained model that we will use is BART, which uses a bidirectional transformer encoder and a unidirectional transformer decoder, as illustrated in the below diagram (image courtesy https://arxiv.org/pdf/1910.13461):

We can see that this model is strikingly similar to the LSTM-based encoder-decoder model we've been using. The only difference is that they use transformers instead of LSTMs. Therefore, we only need to change the modeling parts of the code, as we will see later.

First, we download and load the pretrained BART model from the transformers package by Huggingface. Note that we also need to use the "tokenizer" of BART, which is actually a combination of a tokenizer and a mapping from strings to word ids.

[ ]:
```
pretrained_bart = BartForConditionalGeneration.from_pretrained('facebook/
 ↪bart-base')
bart_tokenizer = BartTokenizer.from_pretrained('facebook/bart-base')
```

```
Downloading:   0%|          | 0.00/1.68k [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/532M [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/878k [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/446k [00:00<?, ?B/s]
```

Below we demonstrate how to use BART's tokenizer to convert a sentence to a list of word ids, and vice versa.

[ ]:
```
# BART uses a predefined "tokenizer", which directly maps a sentence
# to a list of ids
def bart_tokenize(string):
  return bart_tokenizer(string)['input_ids'][:1024] # BART model can process at␣
 ↪most 1024 tokens

def bart_detokenize(token_ids):
    return bart_tokenizer.decode(token_ids, skip_special_tokens=True)

## Demonstrating the tokenizer
```

```
question = 'Are there any first-class flights from St. Louis at 11pm for less␣
↪than $3.50?'

tokenized_question = bart_tokenize(question)
print('tokenized:', tokenized_question)

detokenized_question = bart_detokenize(tokenized_question)
print('detokenized:', detokenized_question)
```

```
tokenized: [0, 13755, 89, 143, 78, 12, 4684, 4871, 31, 312, 4, 3217, 23, 365,
1685, 13, 540, 87, 68, 246, 4, 1096, 116, 2]
detokenized: Are there any first-class flights from St. Louis at 11pm for less
than $3.50?
```

We need to reprocess the data using our new tokenizer. Note that here we set `batch_first` to `True`, since that's the expected input shape of the transformers package.

```
[ ]: SRC_BART = tt.data.Field(include_lengths=True,   # include lengths
                             batch_first=True,       # batches will be batch_size x␣
      ↪max_len
                             tokenize=bart_tokenize, # use bart tokenizer
                             use_vocab=False,        # bart tokenizer already␣
      ↪converts to int ids
                             pad_token=bart_tokenizer.pad_token_id
                            )
     TGT_BART = tt.data.Field(include_lengths=False,
                             batch_first=True,       # batches will be batch_size x␣
      ↪max_len
                             tokenize=bart_tokenize, # use bart tokenizer
                             use_vocab=False,        # bart tokenizer already␣
      ↪converts to int ids
                             pad_token=bart_tokenizer.pad_token_id
                            )
     fields_bart = [('src', SRC_BART), ('tgt', TGT_BART)]

     # Make splits for data
     train_data_bart, val_data_bart, test_data_bart = tt.datasets.TranslationDataset.
      ↪splits(
         ('_flightid.nl', '_flightid.sql'), fields_bart, path='./data/',
         train='train', validation='dev', test='test')

     BATCH_SIZE = 1 # batch size for training/validation
     TEST_BATCH_SIZE = 1 # batch size for test, we use 1 to make beam search␣
      ↪implementation easier

     train_iter_bart, val_iter_bart = tt.data.BucketIterator.
      ↪splits((train_data_bart, val_data_bart),
```

```
                                                             batch_size=BATCH_SIZE,
                                                             device=device,
                                                             repeat=False,
                                                             sort_key=lambda x: len(x.
 ↪src),
                                                             sort_within_batch=True)
test_iter_bart = tt.data.BucketIterator(test_data_bart,
                                        batch_size=1,
                                        device=device,
                                        repeat=False,
                                        sort=False,
                                        train=False)
```

```
Token indices sequence length is longer than the specified maximum sequence
length for this model (1135 > 1024). Running this sequence through the model
will result in indexing errors
```

Let's take a look at the batch. Note that the shape of the batch is `batch_size x max_len`, instead of `max_len x batch_size` as in the previous part.

[ ]:
```
batch = next(iter(train_iter_bart))
train_batch_text, train_batch_text_lengths = batch.src
print (f"Size of text batch: {train_batch_text.shape}")
print (f"First sentence in batch: {train_batch_text[0]}")
print (f"Length of the third sentence in batch: {train_batch_text_lengths[0]}")
print (f"Converted back to string: {bart_detokenize(train_batch_text[0])}")

train_batch_sql = batch.tgt
print (f"Size of sql batch: {train_batch_sql.shape}")
print (f"First sql in batch: {train_batch_sql[0]}")
print (f"Converted back to string: {bart_detokenize(train_batch_sql[0])}")
```

```
Size of text batch: torch.Size([1, 17])
First sentence in batch: tensor([    0, 12005,   162,  4871,    31,  3774,
20285,   293,     7,   181,
         2582, 39710,    15,   475, 46328,  1559,     2], device='cuda:0')
Length of the third sentence in batch: 17
Converted back to string: show me flights from los angeles to pittsburgh on
monday evening
Size of sql batch: torch.Size([1, 262])
First sql in batch: tensor([    0, 49179,   211, 11595,  2444,  7164,  2524,
1215,   134,     4,
        15801,  1215,   808, 11974,  2524,  2524,  1215,   134,  2156,  3062,
         1215, 11131,  3062,  1215, 11131,  1215,   134,  2156,   343,   343,
         1215,   134,  2156,  3062,  1215, 11131,  3062,  1215, 11131,  1215,
          176,  2156,   343,   343,  1215,   176,  2156,   360,   360,  1215,
          134,  2156,  1248,  1215,  1208,  1248,  1215,  1208,  1215,   134,
        29919,  2524,  1215,   134,     4,  7761,  1215,  2456,  3427,  5457,
         3062,  1215, 11131,  1215,   134,     4,  2456,  3427,  1215, 20414,
```

```
         4248,  3062,  1215, 11131,  1215,   134,     4, 14853,  1215, 20414,
         5457,   343,  1215,   134,     4, 14853,  1215, 20414,  4248,   343,
         1215,   134,     4, 14853,  1215, 13650,  5457,   128, 14502, 11420,
         1723,   108,  4248,    36,  2524,  1215,   134,     4,   560,  1215,
         2456,  3427,  5457,  3062,  1215, 11131,  1215,   176,     4,  2456,
         3427,  1215, 20414,  4248,  3062,  1215, 11131,  1215,   176,     4,
        14853,  1215, 20414,  5457,   343,  1215,   176,     4, 14853,  1215,
        20414,  4248,   343,  1215,   176,     4, 14853,  1215, 13650,  5457,
          128,   510,  2068,  2685, 10803, 17201,   108,  4248,    36,  2524,
         1215,   134,     4, 15801,  1215,  7033,  5457,   360,  1215,   134,
            4,  7033,  1215, 20414,  4248,   360,  1215,   134,     4,  1208,
         1215, 13650,  5457,  1248,  1215,  1208,  1215,   134,     4,  1208,
         1215, 13650,  4248,  1248,  1215,  1208,  1215,   134,     4,   180,
         5457,  9633,  4248,  1248,  1215,  1208,  1215,   134,     4,  2151,
         1215, 30695,  5457,   132,  4248,  1248,  1215,  1208,  1215,   134,
            4,  1208,  1215, 30695,  5457,   733,  4248,    36,  2524,  1215,
          134,     4, 17272,  2013,  2407,  1215,   958, 24844,  9112,  2796,
        18360,  4248,   132,  2619,  4248,   112,  5457,   112,  4839,  4839,
         4839,     2], device='cuda:0')
Converted back to string: SELECT DISTINCT flight_1.flight_id FROM flight
flight_1, airport_service airport_service_1, city city_1, airport_service
airport_service_2, city city_2, days days_1, date_day date_day_1 WHERE
flight_1.from_airport = airport_service_1.airport_code AND
airport_service_1.city_code = city_1.city_code AND city_1.city_name = 'LOS
ANGELES' AND ( flight_1.to_airport = airport_service_2.airport_code AND
airport_service_2.city_code = city_2.city_code AND city_2.city_name =
'PITTSBURGH' AND ( flight_1.flight_days = days_1.days_code AND days_1.day_name =
date_day_1.day_name AND date_day_1.year = 1991 AND date_day_1.month_number = 2
AND date_day_1.day_number = 21 AND ( flight_1.departure_time BETWEEN 1800 AND
2200 AND 1 = 1 ) ) )
```

Now we are ready to implement the BART-based approach for the text-to-SQL conversion problem. In the below `BART` class, we have provided the constructer `__init__`, the `forward` function, and the `predict` function. Your job is to implement the main optimization `train_all`, and `evaluate_ppl` for evaluating validation perplexity for model selection.

> Hint: you can use almost the same `train_all` and `evaluate_ppl` function you implemented before, but here a major difference is that due to setting `batch_first=True`, the batched source/target tensors are of size `batch_size x max_len`, as opposed to `max_len x batch_size` in the LSTM-based approach, and you need to make changes in `train_all` and `evaluate_ppl` accordingly.

[ ]:
```python
#TODO - finish implementing the `BART` class.
class BART(nn.Module):
  def __init__(self, tokenizer, pretrained_bart):
    """
    Initializer. Creates network modules and loss function.
    Arguments:
```

```
        tokenizer: BART tokenizer
        pretrained_bart: pretrained BART
    """
    super(BART, self).__init__()

    self.V_tgt = len(tokenizer)

    # Get special word ids
    self.padding_id_tgt = tokenizer.pad_token_id

    # Create essential modules
    self.bart = pretrained_bart

    # Create loss function
    self.loss_function = nn.CrossEntropyLoss(reduction="sum",
                                             ignore_index=self.padding_id_tgt)

  def forward(self, src, src_lengths, tgt_in):
    """
    Performs forward computation, returns logits.
    Arguments:
        src: src batch of size (batch_size, max_src_len)
        src_lengths: src lengths of size (batch_size)
        tgt_in:  a tensor of size (tgt_len, bsz)
    """
    # BART assumes inputs to be batch-first
    # This single function is forwarding both encoder and decoder (w/ cross␣
↪attn),
    # using `input_ids` as encoder inputs, and `decoder_input_ids`
    # as decoder inputs.
    logits = self.bart(input_ids=src,
                       decoder_input_ids=tgt_in,
                       use_cache=False
                      ).logits
    return logits

  def evaluate_ppl(self, iterator):
    """Returns the model's perplexity on a given dataset `iterator`."""
    #TODO - implement this function
    self.eval()
    total_loss = 0
    total_words = 0
    for batch in iterator:
      src, src_lengths = batch.src
      tgt = batch.tgt
      bsz = tgt.size(0)
      shift_right = torch.zeros(bsz,1,dtype=torch.int).cuda()
```

```
        tgt_in = torch.cat([shift_right,tgt[:,:-1].clone() ],dim=1)
        tgt_out = tgt
        logits = self.forward(src, src_lengths, tgt_in)
        loss = self.loss_function(logits.view(-1, self.V_tgt), tgt_out.view(-1))
        total_loss += loss.item()
        total_words += tgt_out.ne(self.padding_id_tgt).float().sum().item()
      ppl = math.exp(total_loss/total_words)
      return ppl

  def train_all(self, train_iter, val_iter, epochs=10, learning_rate=0.001):
    """Train the model."""
    self.train()
    optim = torch.optim.Adam(self.parameters(), lr=learning_rate)
    best_validation_ppl = float('inf')
    best_model = None
    for epoch in range(epochs):
      total_words = 0
      total_loss = 0.0
      for batch in tqdm(train_iter):
        self.zero_grad()
        src, src_lengths = batch.src
        tgt = batch.tgt
        bsz = tgt.size(0)
        shift_right = torch.zeros(bsz,1,dtype=torch.int).cuda()
        tgt_in = torch.cat([shift_right,tgt[:,:-1].clone()],dim=1)
        tgt_out = tgt
        logits = self.forward(src, src_lengths, tgt_in)
        loss = self.loss_function(logits.view(-1, self.V_tgt), tgt_out.view(-1))
        num_tgt_words = tgt_out.ne(self.padding_id_tgt).float().sum().item()
        total_words += num_tgt_words
        total_loss += loss.item()
        loss.div(bsz).backward()
        optim.step()

      validation_ppl = self.evaluate_ppl(val_iter)
      self.train()
      if validation_ppl < best_validation_ppl:
        best_validation_ppl = validation_ppl
        self.best_model = copy.deepcopy(self.state_dict())
      epoch_loss = total_loss / total_words
      print (f'Epoch: {epoch} Training Perplexity: {math.exp(epoch_loss):.4f} '
             f'Validation Perplexity: {validation_ppl:.4f}')

  def predict(self, tokens, K=1, max_T=400):
    """
    Generates the target sequence given the source sequence using beam search␣
↪decoding.
```

```
    Note that for simplicity, we only use batch size 1.
    Arguments:
        tokens: a list of strings, the source sentence.
        max_T: at most proceed this many steps of decoding
    Returns:
        a string of the generated target sentence.
    """
    string = ' '.join(tokens) # first convert to a string
    # Tokenize and map to a list of word ids
    inputs = torch.LongTensor(bart_tokenize(string)).to(device).view(1, -1)
    # The `transformers` package provides built-in beam search support
    prediction = self.bart.generate(inputs,
                                    num_beams=K,
                                    max_length=max_T,
                                    early_stopping=True,
                                    no_repeat_ngram_size=0,
                                    decoder_start_token_id=0,
                                    use_cache=True)[0]
    return bart_detokenize(prediction)
```

The code below will kick off training, and evaluate the validation perplexity. You should expect to see a value very close to 1.

[ ]:
```
EPOCHS = 5 # epochs, we recommend starting with a smaller number like 1
LEARNING_RATE = 1e-5 # learning rate

# Instantiate and train classifier
bart_model = BART(bart_tokenizer,
                  pretrained_bart
).to(device)

bart_model.train_all(train_iter_bart, val_iter_bart, epochs=EPOCHS,
 ↪learning_rate=LEARNING_RATE)
bart_model.load_state_dict(bart_model.best_model)

# Evaluate model performance, the expected value should be < 1.2
print (f'Validation perplexity: {bart_model.evaluate_ppl(val_iter_bart):.3f}')
```

```
100%|      | 3651/3651 [07:04<00:00,  8.61it/s]

Epoch: 0 Training Perplexity: 1.0247 Validation Perplexity: 1.0228

100%|      | 3651/3651 [07:04<00:00,  8.61it/s]

Epoch: 1 Training Perplexity: 1.0201 Validation Perplexity: 1.0185

100%|      | 3651/3651 [07:03<00:00,  8.62it/s]

Epoch: 2 Training Perplexity: 1.0162 Validation Perplexity: 1.0184

100%|      | 3651/3651 [07:03<00:00,  8.63it/s]
```

```
Epoch: 3 Training Perplexity: 1.0141 Validation Perplexity: 1.0170
```

```
100%|     | 3651/3651 [07:02<00:00,  8.64it/s]
```

```
Epoch: 4 Training Perplexity: 1.0121 Validation Perplexity: 1.0184
Validation perplexity: 1.017
```

As before, make sure that your model is making reasonable predictions on a few examples before evaluating on the entire test set.

[ ]:
```
def bart_trial(sentence, gold_sql):
  print("Sentence: ", sentence, "\n")
  tokens = tokenize(sentence)

  predicted_sql = bart_model.predict(tokens, K=1, max_T=300)
  print("Predicted SQL:\n\n", predicted_sql, "\n")

  if verify(predicted_sql, gold_sql, silent=False):
    print ('Correct!')
  else:
    print ('Incorrect!')
```

[ ]:
```
bart_trial(example_1, gold_sql_1)
```

```
Sentence:  flights from phoenix to milwaukee

Predicted SQL:

 SELECT DISTINCT flight_1.flight_id FROM flight flight_1, airport_service
airport_service_1, city city_1, airport_service airport_service_2, city city_2
WHERE flight_1.from_airport = airport_service_1.airport_code AND
airport_service_1.city_code = city_1.city_code AND city_1.city_name = 'PHOENIX'
AND flight_1.to_airport = airport_service_2.airport_code AND
airport_service_2.city_code = city_2.city_code AND city_2.city_name =
'MILWAUKEE'

Predicted DB result:

 [(108086,), (108087,), (301763,), (301764,), (301765,), (301766,), (302323,),
(304881,), (310619,), (310620,)]

Gold DB result:

 [(108086,), (108087,), (301763,), (301764,), (301765,), (301766,), (302323,),
(304881,), (310619,), (310620,)]

Correct!
```

[ ]:
```
bart_trial(example_2, gold_sql_2)
```

```
Sentence:  i would like a united flight

Predicted SQL:

 SELECT DISTINCT flight_1.flight_id FROM flight flight_1 WHERE
flight_1.airline_code = 'UA' AND 1 = 1

Predicted DB result:

 [(100094,), (100099,), (100145,), (100158,), (100164,), (100167,), (100169,),
(100203,), (100204,), (100296,)]

Gold DB result:

 [(100094,), (100099,), (100145,), (100158,), (100164,), (100167,), (100169,),
(100203,), (100204,), (100296,)]

Correct!
```

[ ]:
```
bart_trial(example_3, gold_sql_3)
```

```
Sentence:  i would like a flight between boston and dallas

Predicted SQL:

 SELECT DISTINCT flight_1.flight_id FROM flight flight_1, airport_service
airport_service_1, city city_1, airport_service airport_service_2, city city_2
WHERE flight_1.from_airport = airport_service_1.airport_code AND
airport_service_1.city_code = city_1.city_code AND city_1.city_name = 'BOSTON'
AND flight_1.to_airport = airport_service_2.airport_code AND
airport_service_2.city_code = city_2.city_code AND city_2.city_name = 'DALLAS'

Predicted DB result:

 [(103171,), (103172,), (103173,), (103174,), (103175,), (103176,), (103177,),
(103178,), (103179,), (103180,)]

Gold DB result:

 [(103171,), (103172,), (103173,), (103174,), (103175,), (103176,), (103177,),
(103178,), (103179,), (103180,)]

Correct!
```

[ ]:
```
bart_trial(example_4, gold_sql_4)
```

```
Sentence:  show me the united flights from denver to baltimore

Predicted SQL:
```

```
 SELECT DISTINCT flight_1.flight_id FROM flight flight_1, airport_service
airport_service_1, city city_1, airport_service airport_service_2, city city_2
WHERE flight_1.airline_code = 'UA' AND ( flight_1.from_airport =
airport_service_1.airport_code AND airport_service_1.city_code =
city_1.city_code AND city_1.city_name = 'DENVER' AND flight_1.to_airport =
airport_service_2.airport_code AND airport_service_2.city_code =
city_2.city_code AND city_2.city_name = 'BALTIMORE' )

Predicted DB result:

 [(101231,), (101233,), (305983,)]

Gold DB result:

 [(101231,), (101233,), (305983,)]

Correct!
```

[ ]:
```
bart_trial(example_5, gold_sql_5)
```

```
Sentence:  show flights from cleveland to miami that arrive before 4pm

Predicted SQL:

 SELECT DISTINCT flight_1.flight_id FROM flight flight_1, airport_service
airport_service_1, city city_1, airport_service airport_service_2, city city_2
WHERE flight_1.from_airport = airport_service_1.airport_code AND
airport_service_1.city_code = city_1.city_code AND city_1.city_name =
'CLEVELAND' AND ( flight_1.to_airport = airport_service_2.airport_code AND
airport_service_2.city_code = city_2.city_code AND city_2.city_name = 'MIAMI'
AND flight_1.arrival_time < 1600 )

Predicted DB result:

 [(107698,), (301117,)]

Gold DB result:

 [(107698,), (301117,)]

Correct!
```

[ ]:
```
bart_trial(example_6, gold_sql_6b)
```

```
Sentence:  okay how about a flight on sunday from tampa to charlotte

Predicted SQL:
```

SELECT DISTINCT flight_1.flight_id FROM flight flight_1, airport_service airport_service_1, city city_1, airport_service airport_service_2, city city_2, days days_1, date_day date_day_1 WHERE flight_1.from_airport = airport_service_1.airport_code AND airport_service_1.city_code = city_1.city_code AND city_1.city_name = 'TAMPA' AND (flight_1.to_airport = airport_service_2.airport_code AND airport_service_2.city_code = city_2.city_code AND city_2.city_name = 'CHARLOTTE' AND flight_1.flight_days = days_1.days_code AND days_1.day_name = date_day_1.day_name AND date_day_1.year = 1991 AND date_day_1.month_number = 8 AND date_day_1.day_number = 27)

Predicted DB result:

[(101860,), (101861,), (101862,), (101863,), (101864,), (101865,), (305231,)]

Gold DB result:

[(101860,), (101861,), (101862,), (101863,), (101864,), (101865,), (305231,)]

Correct!

```
[ ]: bart_trial(example_7, gold_sql_7b)
```

Sentence: list all flights going from boston to atlanta that leaves before 7 am on thursday

Predicted SQL:

SELECT DISTINCT flight_1.flight_id FROM flight flight_1, airport_service airport_service_1, city city_1, airport_service airport_service_2, city city_2, days days_1, date_day date_day_1 WHERE flight_1.from_airport = airport_service_1.airport_code AND airport_service_1.city_code = city_1.city_code AND city_1.city_name = 'BOSTON' AND (flight_1.to_airport = airport_service_2.airport_code AND airport_service_2.city_code = city_2.city_code AND city_2.city_name = 'ATLANTA' AND (flight_1.departure_time < 1900 AND flight_1.flight_days = days_1.days_code AND days_1.day_name = date_day_1.day_name AND date_day_1.year = 1991 AND date_day_1.month_number = 5 AND date_day_1.day_number = 24))

Predicted DB result:

[(100014,), (100015,), (100016,), (100017,), (100018,), (100019,), (304692,), (307330,), (100020,), (307329,)]

Gold DB result:

[(100014,)]

Incorrect!

[ ]:
```
bart_trial(example_8, gold_sql_8)
```

```
Sentence:  list the flights from dallas to san francisco on american airlines

Predicted SQL:

 SELECT DISTINCT flight_1.flight_id FROM flight flight_1, airport_service
airport_service_1, city city_1, airport_service airport_service_2, city city_2
WHERE flight_1.airline_code = 'AA' AND ( flight_1.from_airport =
airport_service_1.airport_code AND airport_service_1.city_code =
city_1.city_code AND city_1.city_name = 'DALLAS' AND flight_1.to_airport =
airport_service_2.airport_code AND airport_service_2.city_code =
city_2.city_code AND city_2.city_name = 'SAN FRANCISCO' )

Predicted DB result:

 [(108452,), (108454,), (108456,), (111083,), (111085,), (111086,), (111090,),
(111091,), (111092,), (111094,)]

Gold DB result:

 [(108452,), (108454,), (108456,), (111083,), (111085,), (111086,), (111090,),
(111091,), (111092,), (111094,)]

Correct!
```

6.3.1 Evaluation

The code below will evaluate on the entire test set. You should expect to see precision/recall/F1 greater than 40%.

[ ]:
```
def seq2seq_predictor_bart(tokens):
  prediction = bart_model.predict(tokens, K=4, max_T=400)
  return prediction
```

[ ]:
```
precision, recall, f1 = evaluate(seq2seq_predictor_bart, test_iter.dataset,␣
 ↪num_examples=0)
print(f"precision: {precision:3.2f}")
print(f"recall:    {recall:3.2f}")
print(f"F1:        {f1:3.2f}")
```

```
100%|      | 332/332 [11:01<00:00,  1.99s/it]

precision: 0.52
recall:    0.52
F1:        0.52
```

7 Discussion

7.1 Goal 5: Compare the pros and cons of rule-based and neural approaches.

Compare the pros and cons of the rule-based approach and the neural approaches with relevant examples from your experiments above. Concerning the accuracy, which approach would you choose to be used in a product? Explain.

7.2 Goal 5: Answer

- In Rule based approached it is not mandatory to have a dataset. However, in neural approaches it is mandatory. We can use this advantage in cases where we are starting with no data set or a very small one to slowly build one.
- In Rule based approaches we can utilize our domain knowledge to synthesize better grammar and augmentations and therefore improve the quality, in neural approaches there is no way to apply Domain knowledge. For example, as humans we know the proper syntax of SQL and can therefore define our set of rules that will always result in a valid SQL query as opposed to neural approaches which do not guarantee that all the time.
- In Rule based approaches we need to do the tedious task of manually create a grammar its augmentation and its rules which is prone to human error. In neural approaches we do need to create a grammar and its rules.
- In rule-based approaches some sentences cannot be parsed since it does not have a mechanism to handle sentences with grammar that does not follow the defined rules. In neural approaches this problem is not as prominent since our prediction method does not follow a specific grammar and augmentation so given an input, we can almost always predict something. For example: "I'd like to try flying on United Airlines" this sentence would not be parsed using the rule based approach since "flying on" cannot be derived from the grammar in any way. However a neural approach could attempt to learn how to process this sentence thanks to its generalizing power.
- In rule based its harder to generalize because to do so we need to add more rules sophisticated rules while neural approaches do not require any effort on our side.

Our choice in model for a product would obviously depend on the product, some products care more about precision rather then recall, and in that case we would choose the rule based approach.

If we are considering the product to be a "chatbot" which returns answers based on free text such as demonstrated in the project, we would prefer a model who maximizes both recall **and** precision, therefor we would choose the BERT model, since it has the maximum F1 score. Additionally, as mentioned above, neural approached have much better generalizing power, and that is a desired quality to add to your product.

8 Debrief

Question: We're interested in any thoughts you have about this project segment so that we can improve it for later years, and to inform later segments for this year. Please list any issues that arose or comments you have to improve the project segment. Useful things to comment on might include the following:

- Was the project segment clear or unclear? Which portions?
- Were the readings appropriate background for the project segment?
- Are there additions or changes you think would make the project segment better?

but you should comment on whatever aspects you found especially positive or negative.

The project was not clear. Much of the code was to be copied from lab 4-5 but that created confusion as to what was expected of us. In addition, putting this topic of attention and transformers in the end of the course gives us little time to process and comprehend such a heavy and important topic.

9 Instructions for submission of the project segment

This project segment should be submitted to Gradescope at `https://rebrand.ly/project4-submit-code` and `https://rebrand.ly/project4-submit-pdf`, which will be made available some time before the due date.

Project segment notebooks are manually graded, not autograded using otter as labs are. (Otter is used within project segment notebooks to synchronize distribution and solution code however.) **We will not run your notebook before grading it.** Instead, we ask that you submit the already freshly run notebook. The best method is to "restart kernel and run all cells", allowing time for all cells to be run to completion. You should submit your code to Gradescope at the code submission assignment at `https://rebrand.ly/project4-submit-code`. Make sure that you are also submitting your `data/grammar` file as part of your solution code as well.

We also request that you **submit a PDF of the freshly run notebook**. The simplest method is to use "Export notebook to PDF", which will render the notebook to PDF via LaTeX. If that doesn't work, the method that seems to be most reliable is to export the notebook as HTML (if you are using Jupyter Notebook, you can do so using `File -> Print Preview`), open the HTML in a browser, and print it to a file. Then make sure to add the file to your git commit. Please name the file the same name as this notebook, but with a `.pdf` extension. (Conveniently, the methods just described will use that name by default.) You can then perform a git commit and push and submit the commit to Gradescope at `https://rebrand.ly/project4-submit-pdf`.

10 End of project segment 4