In [1]:

```
!gdown --id 1ToWmWzYJYI_fP4J92wOLPbh1evIam8i2
```

```
/usr/local/lib/python3.10/dist-packages/gdown/cli.py:121: FutureWarning: Option `--id` wa
s deprecated in version 4.3.1 and will be removed in 5.0. You don't need to pass it anymo
re to use a file ID.
  warnings.warn(
Downloading...
From: https://drive.google.com/uc?id=1ToWmWzYJYI_fP4J92wOLPbh1evIam8i2
To: /content/requirements.txt
100% 123/123 [00:00<00:00, 723kB/s]
```

In [2]:

```
!pip install -r /content/requirements.txt
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publi
c/simple/
Requirement already satisfied: torch>=1.9.0 in /usr/local/lib/python3.10/dist-packages (f
rom -r /content/requirements.txt (line 1)) (2.0.1+cu118)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (fro
m -r /content/requirements.txt (line 2)) (3.7.1)
Collecting otter-grader==1.0.0 (from -r /content/requirements.txt (line 3))
  Downloading otter_grader-1.0.0-py3-none-any.whl (163 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 164.0/164.0 kB 13.1 MB/s eta 0:00:00
Collecting wget (from -r /content/requirements.txt (line 4))
  Downloading wget-3.2.zip (10 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from -r /
content/requirements.txt (line 5)) (4.65.0)
Requirement already satisfied: pandas==1.5.3 in /usr/local/lib/python3.10/dist-packages (
from -r /content/requirements.txt (line 6)) (1.5.3)
Collecting transformers==4.26.0 (from -r /content/requirements.txt (line 7))
  Downloading transformers-4.26.0-py3-none-any.whl (6.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.3/6.3 MB 88.7 MB/s eta 0:00:00
Collecting tokenizers==0.13.2 (from -r /content/requirements.txt (line 8))
  Downloading tokenizers-0.13.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.wh
l (7.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.6/7.6 MB 103.5 MB/s eta 0:00:00
Collecting datasets==2.9.0 (from -r /content/requirements.txt (line 9))
  Downloading datasets-2.9.0-py3-none-any.whl (462 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 462.8/462.8 kB 40.4 MB/s eta 0:00:00
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from ot
ter-grader==1.0.0->-r /content/requirements.txt (line 3)) (6.0)
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-packages (from
otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (5.8.0)
Requirement already satisfied: ipython in /usr/local/lib/python3.10/dist-packages (from o
tter-grader==1.0.0->-r /content/requirements.txt (line 3)) (7.34.0)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-packages (from
otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (6.5.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (fro
m otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (67.7.2)
Requirement already satisfied: tornado in /usr/local/lib/python3.10/dist-packages (from o
tter-grader==1.0.0->-r /content/requirements.txt (line 3)) (6.3.1)
Collecting docker (from otter-grader==1.0.0->-r /content/requirements.txt (line 3))
  Downloading docker-6.1.3-py3-none-any.whl (148 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 148.1/148.1 kB 18.5 MB/s eta 0:00:00
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from ot
ter-grader==1.0.0->-r /content/requirements.txt (line 3)) (3.1.2)
Collecting dill (from otter-grader==1.0.0->-r /content/requirements.txt (line 3))
  Downloading dill-0.3.6-py3-none-any.whl (110 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.5/110.5 kB 13.3 MB/s eta 0:00:00
Collecting pdfkit (from otter-grader==1.0.0->-r /content/requirements.txt (line 3))
  Downloading pdfkit-1.0.0-py3-none-any.whl (12 kB)
Collecting PyPDF2 (from otter-grader==1.0.0->-r /content/requirements.txt (line 3))
  Downloading pypdf2-3.0.1-py3-none-any.whl (232 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.6/232.6 kB 24.3 MB/s eta 0:00:00
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-p
```

```
ackages (from pandas==1.5.3->-r /content/requirements.txt (line 6)) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (f
rom pandas==1.5.3->-r /content/requirements.txt (line 6)) (2022.7.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (
from pandas==1.5.3->-r /content/requirements.txt (line 6)) (1.22.4)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from
transformers==4.26.0->-r /content/requirements.txt (line 7)) (3.12.0)
Collecting huggingface-hub<1.0,>=0.11.0 (from transformers==4.26.0->-r /content/requireme
nts.txt (line 7))
  Downloading huggingface_hub-0.15.1-py3-none-any.whl (236 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 236.8/236.8 kB 21.0 MB/s eta 0:00:00
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages
(from transformers==4.26.0->-r /content/requirements.txt (line 7)) (23.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packag
es (from transformers==4.26.0->-r /content/requirements.txt (line 7)) (2022.10.31)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from
transformers==4.26.0->-r /content/requirements.txt (line 7)) (2.27.1)
Requirement already satisfied: pyarrow>=6.0.0 in /usr/local/lib/python3.10/dist-packages
(from datasets==2.9.0->-r /content/requirements.txt (line 9)) (9.0.0)
Collecting xxhash (from datasets==2.9.0->-r /content/requirements.txt (line 9))
  Downloading xxhash-3.2.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (21
2 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 212.5/212.5 kB 22.7 MB/s eta 0:00:00
Collecting multiprocess (from datasets==2.9.0->-r /content/requirements.txt (line 9))
  Downloading multiprocess-0.70.14-py310-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.3/134.3 kB 15.5 MB/s eta 0:00:00
Requirement already satisfied: fsspec[http]>=2021.11.1 in /usr/local/lib/python3.10/dist-
packages (from datasets==2.9.0->-r /content/requirements.txt (line 9)) (2023.4.0)
Collecting aiohttp (from datasets==2.9.0->-r /content/requirements.txt (line 9))
  Downloading aiohttp-3.8.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1
.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 65.4 MB/s eta 0:00:00
Collecting responses<0.19 (from datasets==2.9.0->-r /content/requirements.txt (line 9))
  Downloading responses-0.18.0-py3-none-any.whl (38 kB)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packag
es (from torch>=1.9.0->-r /content/requirements.txt (line 1)) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from tor
ch>=1.9.0->-r /content/requirements.txt (line 1)) (1.11.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from
torch>=1.9.0->-r /content/requirements.txt (line 1)) (3.1)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (
from torch>=1.9.0->-r /content/requirements.txt (line 1)) (2.0.0)
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from tri
ton==2.0.0->torch>=1.9.0->-r /content/requirements.txt (line 1)) (3.25.2)
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from trito
n==2.0.0->torch>=1.9.0->-r /content/requirements.txt (line 1)) (16.0.5)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-package
s (from matplotlib->-r /content/requirements.txt (line 2)) (1.0.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (f
rom matplotlib->-r /content/requirements.txt (line 2)) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib->-r /content/requirements.txt (line 2)) (4.39.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib->-r /content/requirements.txt (line 2)) (1.4.4)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (
from matplotlib->-r /content/requirements.txt (line 2)) (8.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-package
s (from matplotlib->-r /content/requirements.txt (line 2)) (3.0.9)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (
from aiohttp->datasets==2.9.0->-r /content/requirements.txt (line 9)) (23.1.0)
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/python3.10/
dist-packages (from aiohttp->datasets==2.9.0->-r /content/requirements.txt (line 9)) (2.0
.12)
Collecting multidict<7.0,>=4.5 (from aiohttp->datasets==2.9.0->-r /content/requirements.t
xt (line 9))
  Downloading multidict-6.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(114 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 114.5/114.5 kB 12.0 MB/s eta 0:00:00
Collecting async-timeout<5.0,>=4.0.0a3 (from aiohttp->datasets==2.9.0->-r /content/requir
ements.txt (line 9))
  Downloading async_timeout-4.0.2-py3-none-any.whl (5.8 kB)
Collecting yarl<2.0,>=1.0 (from aiohttp->datasets==2.9.0->-r /content/requirements.txt (l
```

```
ine 9))
  Downloading yarl-1.9.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (268
kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 268.8/268.8 kB 19.8 MB/s eta 0:00:00
Collecting frozenlist>=1.1.1 (from aiohttp->datasets==2.9.0->-r /content/requirements.txt
(line 9))
  Downloading frozenlist-1.3.3-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylin
ux_2_17_x86_64.manylinux2014_x86_64.whl (149 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 149.6/149.6 kB 18.0 MB/s eta 0:00:00
Collecting aiosignal>=1.1.2 (from aiohttp->datasets==2.9.0->-r /content/requirements.txt
(line 9))
  Downloading aiosignal-1.3.1-py3-none-any.whl (7.6 kB)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
python-dateutil>=2.8.1->pandas==1.5.3->-r /content/requirements.txt (line 6)) (1.16.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
ckages (from requests->transformers==4.26.0->-r /content/requirements.txt (line 7)) (1.26
.15)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packa
ges (from requests->transformers==4.26.0->-r /content/requirements.txt (line 7)) (2022.12
.7)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (f
rom requests->transformers==4.26.0->-r /content/requirements.txt (line 7)) (3.4)
Requirement already satisfied: websocket-client>=0.32.0 in /usr/local/lib/python3.10/dist
-packages (from docker->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (1.5.
1)
Collecting jedi>=0.16 (from ipython->otter-grader==1.0.0->-r /content/requirements.txt (l
ine 3))
  Downloading jedi-0.18.2-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 77.0 MB/s eta 0:00:00
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from
ipython->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (fr
om ipython->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.10/dist-packages
(from ipython->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (5.7.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local
/lib/python3.10/dist-packages (from ipython->otter-grader==1.0.0->-r /content/requirement
s.txt (line 3)) (3.0.38)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from
ipython->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (2.14.0)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from
ipython->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packag
es (from ipython->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (0.1.6)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (fr
om ipython->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (4.8.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages
(from jinja2->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (2.1.2)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbco
nvert->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (4.9.2)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages
(from nbconvert->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (4.11.2)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nb
convert->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (6.0.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (fro
m nbconvert->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-packa
ges (from nbconvert->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (0.4)
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.10/dist-packag
es (from nbconvert->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (5.3.0)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-pack
ages (from nbconvert->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (0.2.2)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packag
es (from nbconvert->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages
(from nbconvert->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (0.7.4)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-pac
kages (from nbconvert->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (1.5.0
)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from
nbconvert->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (1.2.1)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.10/dist-packages
```

```
(from nbformat->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (2.16.3)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages
(from nbformat->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (4.3.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (f
rom sympy->torch>=1.9.0->-r /content/requirements.txt (line 1)) (1.3.0)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in /usr/local/lib/python3.10/dist-pack
ages (from jedi>=0.16->ipython->otter-grader==1.0.0->-r /content/requirements.txt (line 3
)) (0.8.3)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in /usr/loca
l/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->otter-grader==1.0.0->-r /
content/requirements.txt (line 3)) (0.19.3)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packag
es (from jupyter-core>=4.7->nbconvert->otter-grader==1.0.0->-r /content/requirements.txt
(line 3)) (3.3.0)
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.10/dist-p
ackages (from nbclient>=0.5.0->nbconvert->otter-grader==1.0.0->-r /content/requirements.t
xt (line 3)) (6.1.12)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packages
(from pexpect>4.3->ipython->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (
0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from p
rompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython->otter-grader==1.0.0->-r /content/re
quirements.txt (line 3)) (0.2.6)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (
from beautifulsoup4->nbconvert->otter-grader==1.0.0->-r /content/requirements.txt (line 3
)) (2.4.1)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (f
rom bleach->nbconvert->otter-grader==1.0.0->-r /content/requirements.txt (line 3)) (0.5.1
)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-packages (from
jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert->otter-grader==1.0.0->-r /content/requ
irements.txt (line 3)) (23.2.1)
Building wheels for collected packages: wget
  Building wheel for wget (setup.py) ... done
  Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=9657 sha256=60d3be236e8
3c5f6bdb59c71578c32ec1978069ffe37620f3ce3371fbbf29776
  Stored in directory: /root/.cache/pip/wheels/8b/f1/7f/5c94f0a7a505ca1c81cd1d9208ae20646
75d97582078e6c769
Successfully built wget
Installing collected packages: wget, tokenizers, pdfkit, xxhash, PyPDF2, multidict, jedi,
frozenlist, dill, async-timeout, yarl, responses, multiprocess, huggingface-hub, docker,
aiosignal, transformers, aiohttp, otter-grader, datasets
Successfully installed PyPDF2-3.0.1 aiohttp-3.8.4 aiosignal-1.3.1 async-timeout-4.0.2 dat
asets-2.9.0 dill-0.3.6 docker-6.1.3 frozenlist-1.3.3 huggingface-hub-0.15.1 jedi-0.18.2 m
ultidict-6.0.4 multiprocess-0.70.14 otter-grader-1.0.0 pdfkit-1.0.0 responses-0.18.0 toke
nizers-0.13.2 transformers-4.26.0 wget-3.2 xxhash-3.2.0 yarl-1.9.2
```

In [3]:

```
# 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/cs236299-2023-spring/project2.git .tmp
 mv .tmp/requirements.txt ./
 rm -rf .tmp
fi
pip install -q -r requirements.txt
""")
```

In [4]:

```
# Initialize Otter
import otter
grader = otter.Notebook()
```

%%latex \newcommand{\vect}[1]{\mathbf{#1}} \newcommand{\cnt}[1]{\sharp(#1)} \newcommand{\argmax}[1]{\underset{#1}{\operatorname{argmax}}} \newcommand{\softmax}{\operatorname{softmax}} \newcommand{\Prob}{\Pr} \newcommand{\given}{\,|\,}

236299 - Introduction to Natural Language Processing

Project 2: Sequence labeling – The slot filling task

Introduction

The second segment of the project involves a sequence labeling task, in which the goal is to label the tokens in a text. Many NLP tasks have this general form. Most famously is the task of *part-of-speech labeling* as you explored in lab 2-4, where the tokens in a text are to be labeled with their part of speech (noun, verb, preposition, etc.). In this project segment, however, you'll use sequence labeling to implement a system for filling the slots in a template that is intended to describe the meaning of an ATIS query. For instance, the sentence

```
What's the earliest arriving flight between Boston and Washington DC?
```

might be associated with the following slot-filled template:

```
flight_id
    fromloc.cityname: boston
    toloc.cityname: washington
    toloc.state: dc
    flight_mod: earliest arriving
```

You may wonder how this task is a sequence labeling task. We label each word in the source sentence with a tag taken from a set of tags that correspond to the slot-labels. For each slot-label, say `flight_mod`, there are two tags: `B-flight_mod` and `I-flight_mod`. These are used to mark the beginning (B) or interior (I) of a phrase that fills the given slot. In addition, there is a tag for other (O) words that are not used to fill any slot. (This technique is thus known as IOB encoding.) Thus the sample sentence would be labeled as follows:

Token	Label
`BOS`	`O`
`what's`	`O`
`the`	`O`
`earliest`	`B-flight_mod`
`arriving`	`I-flight_mod`
`flight`	`O`
`between`	`O`

Token	Label
boston	B-fromloc.city_name
and	O
washington	B-toloc.city_name
dc	B-toloc.state_code
EOS	O

> See below for information about the `BOS` and `EOS` tokens.

The template itself is associated with the question type for the sentence, perhaps as recovered from the sentence in the last project segment.

In this segment, you'll implement three methods for sequence labeling: a hidden Markov model (HMM) and two recurrent neural networks, a simple RNN and a long short-term memory network (LSTM). By the end of this homework, you should have grasped the pros and cons of the statistical and neural approaches.

Goals

1. Implement an HMM-based approach to sequence labeling.
2. Implement an RNN-based approach to sequence labeling.
3. Implement an LSTM-based approach to sequence labeling.
4. Compare the performances of HMM and RNN/LSTM with different amounts of training data. Discuss the pros and cons of the HMM approach and the neural approach.

Setup

In [5]:

```
import copy
import math
import matplotlib.pyplot as plt
import random
import csv

import wget
import torch
import torch.nn as nn
import datasets

from datasets import load_dataset
from tokenizers import Tokenizer
from tokenizers.pre_tokenizers import WhitespaceSplit
from tokenizers.processors import TemplateProcessing
from tokenizers import normalizers
from tokenizers.models import WordLevel
from tokenizers.trainers import WordLevelTrainer
from transformers import PreTrainedTokenizerFast

from tqdm.auto import tqdm
```

In [6]:

```
# Set random seeds
seed = 1234
random.seed(seed)
torch.manual_seed(seed)

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

```
cpu
```

Loading data

We download the ATIS dataset, already presplit into training, validation (dev), and test sets.

In [7]:

```
# Prepare to download needed data
def download_if_needed(filename, source='./', dest='./'):
    os.makedirs(data_path, exist_ok=True) # ensure destination
    os.path.exists(f"./{dest}{filename}") or wget.download(source + filename, out=dest)

source_path = "https://raw.githubusercontent.com/nlp-course/data/master/ATIS/"
data_path = "data/"

# Download files
for filename in ["atis.train.txt",
                 "atis.dev.txt",
                 "atis.test.txt"
                ]:
    download_if_needed(filename, source_path, data_path)
```

Data preprocessing

We again use `datasets` and `tokenizers` to load data and convert words to indices in the vocabulary.

We treat words occurring fewer than three times in the training data as *unknown words*. They'll be replaced by the unknown word type `[UNK]`.

In [8]:

```
for split in ['train', 'dev', 'test']:
    in_file = f'data/atis.{split}.txt'
    out_file = f'data/atis.{split}.csv'

    with open(in_file, 'r') as f_in:
        with open(out_file, 'w') as f_out:
            text, tag = [], []
            writer = csv.writer(f_out)
            writer.writerow(('text','tag'))
            for line in f_in:
                if line.strip() == '':
                    writer.writerow((' '.join(text), ' '.join(tag)))
                    text, tag = [], []
                else:
                    token, label = line.split('\t')
                    text.append(token)
                    tag.append(label.strip())
```

Let's take a look at what each data file looks like.

In [9]:

```
shell('head "data/atis.train.csv"')
```

```
text,tag
BOS what is the cost of a round trip flight from pittsburgh to atlanta beginning on april
twenty fifth and returning on may sixth EOS,O O O O O O O B-round_trip I-round_trip O O B
-fromloc.city_name O B-toloc.city_name O O B-depart_date.month_name B-depart_date.day_num
ber I-depart_date.day_number O O O B-return_date.month_name B-return_date.day_number O
BOS now i need a flight leaving fort worth and arriving in denver no later than 2 pm next
monday EOS,O O O O O O O B-fromloc.city_name I-fromloc.city_name O O O B-toloc.city_name
B-arrive_time.time_relative I-arrive_time.time_relative I-arrive_time.time_relative B-arr
ive_time.time I-arrive_time.time B-arrive_date.date_relative B-arrive_date.day_name O
BOS i need to fly from kansas city to chicago leaving next wednesday and returning the fo
llowing day EOS,O O O O O O B-fromloc.city_name I-fromloc.city_name O B-toloc.city_name O
B-depart_date.date_relative B-depart_date.day_name O O B-return_date.date_relative I-retu
rn_date.date_relative I-return_date.date_relative O
BOS what is the meaning of meal code s EOS,O O O O O O B-meal_code I-meal_code I-meal_cod
```

```
e O
BOS show me all flights from denver to pittsburgh which serve a meal for the day after to
morrow EOS,O O O O O O B-fromloc.city_name O B-toloc.city_name O O O B-meal O B-depart_da
te.today_relative I-depart_date.today_relative I-depart_date.today_relative I-depart_date
.today_relative O
BOS show me all us air flights from atlanta to denver for the day after tomorrow EOS,O O
O O B-airline_name I-airline_name O O B-fromloc.city_name O B-toloc.city_name O B-depart_
date.today_relative I-depart_date.today_relative I-depart_date.today_relative I-depart_da
te.today_relative O
BOS list the nonstop flights early tuesday morning from dallas to atlanta EOS,O O O B-fli
ght_stop O B-arrive_time.period_mod B-arrive_date.day_name B-arrive_time.period_of_day O
B-fromloc.city_name O B-toloc.city_name O
BOS show me the flights from st. petersburg to toronto that arrive early in the morning E
OS,O O O O O O B-fromloc.city_name I-fromloc.city_name O B-toloc.city_name O O B-arrive_t
ime.period_mod O O B-arrive_time.period_of_day O
BOS i need a listing of flights from new york city to montreal canada departing thursday
in the morning EOS,O O O O O O O O B-fromloc.city_name I-fromloc.city_name I-fromloc.city
_name O B-toloc.city_name B-toloc.country_name O B-depart_date.day_name O O B-depart_time
.period_of_day O
```

We use `datasets` to prepare the data.

In [10]:

```
atis = load_dataset('csv', data_files={'train':'data/atis.train.csv', \
                                       'val': 'data/atis.dev.csv', \
                                       'test': 'data/atis.test.csv'})
atis
```

```
WARNING:datasets.builder:Using custom data configuration default-a31f1de2f7fa41be
```

```
Downloading and preparing dataset csv/default to /root/.cache/huggingface/datasets/csv/de
fault-a31f1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bfbe43546f190a60fcf74bb5e8afdcc
2317...
```

```
/usr/local/lib/python3.10/dist-packages/datasets/download/streaming_download_manager.py:7
76: FutureWarning: the 'mangle_dupe_cols' keyword is deprecated and will be removed in a
future version. Please take steps to stop the use of 'mangle_dupe_cols'
  return pd.read_csv(xopen(filepath_or_buffer, "rb", use_auth_token=use_auth_token), **kw
args)
```

```
/usr/local/lib/python3.10/dist-packages/datasets/download/streaming_download_manager.py:7
76: FutureWarning: the 'mangle_dupe_cols' keyword is deprecated and will be removed in a
future version. Please take steps to stop the use of 'mangle_dupe_cols'
  return pd.read_csv(xopen(filepath_or_buffer, "rb", use_auth_token=use_auth_token), **kw
args)
```

```
Dataset csv downloaded and prepared to /root/.cache/huggingface/datasets/csv/default-a31f
1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bfbe43546f190a60fcf74bb5e8afdcc2317. Subs
equent calls will reuse this data.
```

```
/usr/local/lib/python3.10/dist-packages/datasets/download/streaming_download_manager.py:7
76: FutureWarning: the 'mangle_dupe_cols' keyword is deprecated and will be removed in a
future version. Please take steps to stop the use of 'mangle_dupe_cols'
  return pd.read_csv(xopen(filepath_or_buffer, "rb", use_auth_token=use_auth_token), **kw
args)
```

Out[10]:

```
DatasetDict({
    train: Dataset({
        features: ['text', 'tag'],
        num_rows: 4274
    })
    val: Dataset({
        features: ['text', 'tag'],
        num_rows: 572
    })
    test: Dataset({
```

```
        features: ['text', 'tag'],
        num_rows: 586
    })
})
```

In [11]:

```
train_data = atis['train']
val_data = atis['val']
test_data = atis['test']

train_data.shuffle(seed=seed)
```

Out[11]:

```
Dataset({
    features: ['text', 'tag'],
    num_rows: 4274
})
```

We build tokenizers from the training data to tokenize both text and tag and convert them into word ids.

In [12]:

```
MIN_FREQ = 3
unk_token = '[UNK]'
pad_token = '[PAD]'
bos_token = '<bos>'

def train_tokenizers(dataset, min_freq):
    text_tokenizer = Tokenizer(WordLevel(unk_token=unk_token))
    text_tokenizer.pre_tokenizer = WhitespaceSplit()
    text_tokenizer.normalizer = normalizers.Lowercase()

    text_trainer = WordLevelTrainer(min_frequency=min_freq, special_tokens=[pad_token, u
nk_token,bos_token])
    text_tokenizer.train_from_iterator(dataset['text'], trainer=text_trainer)
    text_tokenizer.post_processor = TemplateProcessing(single=f"{bos_token} $A", special_
tokens=[(bos_token, text_tokenizer.token_to_id(bos_token))])

    tag_tokenizer = Tokenizer(WordLevel(unk_token=unk_token))
    tag_tokenizer.pre_tokenizer = WhitespaceSplit()

    tag_trainer = WordLevelTrainer(special_tokens=[pad_token, unk_token, bos_token])

    tag_tokenizer.train_from_iterator(dataset['tag'], trainer=tag_trainer)

    tag_tokenizer.post_processor = TemplateProcessing(single=f"{bos_token} $A", special_t
okens=[(bos_token, tag_tokenizer.token_to_id(bos_token))])
    return text_tokenizer, tag_tokenizer

text_tokenizer, tag_tokenizer = train_tokenizers(train_data, MIN_FREQ)
```

We use `datasets.Dataset.map` to convert text into word ids. As shown in lab 1-5, first we need to wrap `tokenizer` with the `transformers.PreTrainedTokenizerFast` class to be compatible with the `datasets` library.

In [13]:

```
hf_text_tokenizer = PreTrainedTokenizerFast(tokenizer_object=text_tokenizer, pad_token=pa
d_token, unk_token=unk_token, bos_token=bos_token)

hf_tag_tokenizer = PreTrainedTokenizerFast(tokenizer_object=tag_tokenizer, pad_token=pad_
token, unk_token=unk_token, bos_token=bos_token)
```

In [14]:

```
def encode(example):
    example['input_ids'] = hf_text_tokenizer(example['text']).input_ids
```

```python
    example['tag_ids'] = hf_tag_tokenizer(example['tag']).input_ids
    return example

train_data = train_data.map(encode)
val_data = val_data.map(encode)
test_data = test_data.map(encode)
```

We can get some sense of the datasets by looking at the size of the text and tag vocabularies.

In [15]:

```python
# Compute size of vocabulary
text_vocab = text_tokenizer.get_vocab()
tag_vocab = tag_tokenizer.get_vocab()
vocab_size = len(text_vocab)
num_tags = len(tag_vocab)

print(f"Size of English vocabulary: {vocab_size}")
print(f"Number of tags: {num_tags}")
```

```
Size of English vocabulary: 518
Number of tags: 104
```

Special tokens and tags

You'll have already noticed the `BOS` and `EOS`, special tokens that the dataset developers used to indicate the beginning and end of the sentence; we'll leave them in the data.

We've also prepended `<bos>` for both text and tag. `Tokenizers` will prepend these to the sequence of words and tags. This relieves us from estimating the initial distribution of tags and tokens in HMMs, since we always start with a token `<bos>` whose tag is also `<bos>`. We'll be able to refer to these tags as exemplified here:

In [16]:

```python
print(f"""
Initial tag string: {bos_token}
Initial tag id:     {tag_vocab[bos_token]}
""")
```

```
Initial tag string: <bos>
Initial tag id:     2
```

Finally, since we will be providing the sentences in the training corpus in "batches", we willl force the sentences within a batch to be the same length by padding them with a special `[PAD]` token. Again, we can access that token as shown here:

In [17]:

```python
print(f"""
Pad tag string: {pad_token}
Pad tag id:     {tag_vocab[pad_token]}
""")
```

```
Pad tag string: [PAD]
Pad tag id:     0
```

To load data in batched tensors, we use `torch.utils.data.DataLoader` for data splits, which enables us to iterate over the dataset under a given `BATCH_SIZE`. For the test set, we use a batch size of 1, to make the decoding implementation easier.

In [18]:

```
BATCH_SIZE = 32      # batch size for training and validation

# Defines how to batch a list of examples together
def collate_fn(examples):
    batch = {}
    bsz = len(examples)
    input_ids, tag_ids = [], []
    for example in examples:
        input_ids.append(example['input_ids'])
        tag_ids.append(example['tag_ids'])

    max_length = max([len(word_ids) for word_ids in input_ids])

    tag_batch = torch.zeros(bsz, max_length).long().fill_(tag_vocab[pad_token]).to(devic
e)
    text_batch = torch.zeros(bsz, max_length).long().fill_(text_vocab[pad_token]).to(dev
ice)
    for b in range(bsz):
        text_batch[b][:len(input_ids[b])] = torch.LongTensor(input_ids[b]).to(device)
        tag_batch[b][:len(tag_ids[b])] = torch.LongTensor(tag_ids[b]).to(device)

    batch['tag_ids'] = tag_batch
    batch['input_ids'] = text_batch
    return batch

def get_iterators(train_data, val_data, test_data):
    train_iter = torch.utils.data.DataLoader(train_data,
                                             batch_size=BATCH_SIZE,
                                             shuffle=True,
                                             collate_fn=collate_fn)
    val_iter = torch.utils.data.DataLoader(val_data,
                                           batch_size=BATCH_SIZE,
                                           shuffle=False,
                                           collate_fn=collate_fn)
    test_iter = torch.utils.data.DataLoader(test_data,
                                            batch_size=BATCH_SIZE,
                                            shuffle=False,
                                            collate_fn=collate_fn)
    return train_iter, val_iter, test_iter

train_iter, val_iter, test_iter = get_iterators(train_data, val_data, test_data)
```

Now, we can iterate over the dataset. We used a non-trivial batch size to gain the benefit of training on multiple sentences at a shot. You'll need to be careful about the shapes of the various tensors that are being manipulated.

Each batch will be a tensor of size `batch_size x max_length`**. Let's examine a batch.**

In [19]:

```
# Get the first batch
batch = next(iter(train_iter))

# What's its shape? Should be batch_size x max_length.
print(f'Shape of batch text tensor: {batch["input_ids"].shape}\n')

# Extract the first sentence in the batch, both text and tags
first_sentence = batch['input_ids'][0]
first_tags = batch['tag_ids'][0]

# Print out the first sentence, as token ids and as text
print("First sentence in batch")
print(f"{first_sentence}")
print(f"{hf_text_tokenizer.decode(first_sentence)}\n")

print("First tags in batch")
print(f"{first_tags}")
print(f"{hf_tag_tokenizer.decode(first_tags)}")
```

```
Shape of batch text tensor: torch.Size([32, 21])
```

```
First sentence in batch
tensor([  2,   3,  82, 154,   7,  31,  50,  36,  14,  19,  20,  29,   9, 121,
        431,   4,   0,   0,   0,   0,   0])
<bos> bos how many flights are there between san francisco and philadelphia on august eig
hteenth eos [PAD] [PAD] [PAD] [PAD] [PAD]

First tags in batch
tensor([ 2,  3,  3,  3,  3,  3,  3,  3,  5,  8,  3,  4,  3, 13, 12,  3,  0,  0,
         0,  0,  0])
<bos> O O O O O O O B-fromloc.city_name I-fromloc.city_name O B-toloc.city_name O B-depar
t_date.month_name B-depart_date.day_number O [PAD] [PAD] [PAD] [PAD] [PAD]
```

The goal of this project is to predict the sequence of tags `batch['tag_ids']` **given a sequence of words** `batch['input_ids']`**.**

Majority class labeling

As usual, we can get a sense of the difficulty of the task by looking at a simple baseline, tagging every token with the majority tag. Here's a table of tag frequencies for the most frequent tags:

In [20]:

```
def count_tags(iterator):
  tag_counts = torch.zeros(len(tag_vocab), device=device)

  for batch in iterator:
    tags = batch['tag_ids'].view(-1)
    tag_counts.scatter_add_(0, tags, torch.ones(tags.shape).to(device))

  ## Alternative untensorized implementation for reference
  # for batch in iterator:                       # for each batch
  #   for sent_id in range(len(batch)):          # ... each sentence in the batch
  #     for tag in batch.tag[:, sent_id]:        # ... each tag in the sentence
  #       tag_counts[tag] += 1                   # bump the tag count

  # Ignore paddings
  tag_counts[tag_vocab[pad_token]] = 0
  return tag_counts

tag_counts = count_tags(train_iter)

for tag_id in range(len(tag_vocab)):
  print(f'{tag_id:3}  {hf_tag_tokenizer.decode(tag_id):30}{tag_counts[tag_id].item():3.0f
}')
```

```
  0  [PAD]                           0
  1  [UNK]                           0
  2  <bos>                        4274
  3  O                           38967
  4  B-toloc.city_name            3751
  5  B-fromloc.city_name          3726
  6  I-toloc.city_name            1039
  7  B-depart_date.day_name       835
  8  I-fromloc.city_name          636
  9  B-airline_name               610
 10  B-depart_time.period_of_day  555
 11  I-airline_name               374
 12  B-depart_date.day_number     351
 13  B-depart_date.month_name     340
 14  B-depart_time.time           321
 15  B-round_trip                 311
 16  I-round_trip                 303
 17  B-depart_time.time_relative  290
 18  B-cost_relative              281
 19  B-flight_mod                 264
 20  I-depart_time.time           258
 21  B-stoploc.city_name          202
 22  B-city_name                  191
```

23	B-arrive_time.time	182
24	B-class_type	181
25	B-arrive_time.time_relative	162
26	I-class_type	148
27	I-arrive_time.time	142
28	B-flight_stop	141
29	B-airline_code	109
30	I-depart_date.day_number	105
31	I-fromloc.airport_name	103
32	B-toloc.state_name	84
33	B-toloc.state_code	81
34	B-arrive_date.day_name	78
35	B-fromloc.airport_name	75
36	B-depart_date.date_relative	72
37	B-flight_number	72
38	B-depart_date.today_relative	70
39	I-airport_name	61
40	I-city_name	53
41	B-arrive_time.period_of_day	51
42	B-fare_basis_code	51
43	B-flight_time	51
44	B-fromloc.state_code	51
45	B-or	49
46	B-aircraft_code	48
47	B-meal_description	48
48	B-meal	47
49	I-cost_relative	45
50	I-stoploc.city_name	45
51	B-airport_name	44
52	B-transport_type	43
53	B-fromloc.state_name	42
54	B-arrive_date.day_number	40
55	B-arrive_date.month_name	40
56	B-depart_time.period_mod	39
57	B-flight_days	37
58	B-connect	36
59	I-toloc.airport_name	35
60	B-fare_amount	34
61	I-fare_amount	33
62	B-economy	32
63	B-toloc.airport_name	28
64	B-mod	24
65	I-flight_time	24
66	B-airport_code	22
67	B-depart_date.year	20
68	B-toloc.airport_code	19
69	B-arrive_time.start_time	18
70	B-depart_time.end_time	18
71	B-depart_time.start_time	18
72	I-transport_type	18
73	B-arrive_time.end_time	17
74	I-arrive_time.end_time	16
75	B-fromloc.airport_code	14
76	B-restriction_code	14
77	I-depart_time.end_time	13
78	I-flight_mod	12
79	I-flight_stop	12
80	B-arrive_date.date_relative	10
81	I-toloc.state_name	10
82	I-restriction_code	9
83	B-return_date.date_relative	8
84	I-depart_time.start_time	8
85	I-economy	8
86	B-state_code	7
87	I-arrive_time.start_time	7
88	I-fromloc.state_name	7
89	B-state_name	6
90	I-depart_date.today_relative	6
91	I-depart_time.period_of_day	5
92	B-period_of_day	4
93	I-arrive_date.day_number	4
94	B-day_name	3

```
 95   B-meal_code                       3
 96   B-stoploc.state_code              3
 97   B-arrive_time.period_mod          2
 98   B-toloc.country_name              2
 99   I-arrive_time.time_relative       2
100   I-meal_code                       2
101   I-return_date.date_relative       2
102   B-return_date.day_number          1
103   B-return_date.month_name          1
```

It looks like the `'O'` (other) tag is, unsurprisingly, the most frequent tag (except for the padding tag). The proportion of tokens labeled with that tag (ignoring the padding tag) gives us a good baseline accuracy for this sequence labeling task. To verify that intuition, we can calculate the accuracy of the majority tag on the test set:

In [21]:

```
tag_counts_test = count_tags(test_iter)
majority_baseline_accuracy = (
  tag_counts_test[tag_vocab['O']]
  / tag_counts_test.sum()
)
print(f'Baseline accuracy: {majority_baseline_accuracy:.3f}')
```

```
Baseline accuracy: 0.634
```

HMM for sequence labeling

Having established the baseline to beat, we turn to implementing an HMM model.

Notation

First, let's start with some notation. We use $\mathcal{V} = \langle \mathcal{V}_1, \mathcal{V}_2, \ldots \mathcal{V}_V \rangle$ to denote the vocabulary of word types and $Q = \langle Q_1, Q_2, \ldots, Q_N \rangle$ to denote the possible tags, which is the state space of the HMM. Thus V is the number of word types in the vocabulary and N is the number of states (tags).

We use $\mathbf{w} = w_1 \cdots w_T \in \mathcal{V}^T$
to denote the string of words at "time steps" t
(where t
varies from 1
to T
). Similarly, $\mathbf{q} = q_1 \cdots q_T \in Q^T$
denotes the corresponding sequence of states (tags).

Training an HMM by counting

Recall that an HMM is defined via a transition matrix A
, which stores the probability of moving from one state Q_i
to another Q_j
, that is,

$$A_{ij} = \Pr(q_{t+1} = Q_j \mid q_t = Q_i)$$

and an emission matrix B
, which stores the probability of generating word $\mathcal{V}_j$
given state Q_i
, that is,

$$B_{ij} = \Pr(w_t = \mathcal{V}_j \mid q_t = Q_i)$$

> **As is typical in notating probabilities, we'll use abbreviations**
>
> $$\Pr(q_{t+1} \mid q_t) \equiv \Pr(q_{t+1} = Q_j \mid q_t = Q_i)$$

$$\Pr(q_{t+1} \mid q_t) \equiv \Pr(q_{t+1} = Q_j \mid q_t = Q_i)$$
$$\Pr(w_t \mid q_t) \equiv \Pr(w_t = \mathcal{V}_j \mid q_t = Q_i)$$

> where the i and j are clear from context.

In our case, since the labels are observed in the training data, we can directly use counting to determine (maximum likelihood) estimates of A and B.

Goal 1(a): Find the transition matrix

The matrix A contains the transition probabilities: A_{ij} is the probability of moving from state Q_i to state Q_j in the training data, so that $\sum_{j=1}^{N} A_{ij} = 1$ for all i.

We find these probabilities by counting the number of times state Q_j appears right after state Q_i, as a proportion of all of the transitions from Q_i.

$$A_{ij} = \frac{\sharp(Q_i, Q_j) + \delta}{\sum_k \left(\sharp(Q_i, Q_k) + \delta\right)}$$

(In the above formula, we also used add-δ smoothing.)

Using the above definition, implement the method `train_A` in the `HMM` class below, which calculates and returns the A matrix as a tensor of size $N \times N$.

> You'll want to go ahead and implement this part now, and test it below, before moving on to the next goal.
>
> Remember that the training data is being delivered to you batched.

Goal 1(b): Find the emission matrix B

Similar to the transition matrix, the emission matrix contains the emission probabilities such that B_{ij} is probability of word $w_t = \mathcal{V}_j$ conditioned on state $q_t = Q_i$.

We can find this by counting as well.

$$B_{ij} = \frac{\sharp(Q_i, \mathcal{V}_j) + \delta}{\sum_k \left(\sharp(Q_i, \mathcal{V}_k) + \delta\right)} = \frac{\sharp(Q_i, \mathcal{V}_j) + \delta}{\sharp(Q_i) + \delta V}$$

Using the above definitions, implement the `train_B` method in the `HMM` class below, which calculates and returns the B
matrix as a tensor of size $N \times V$
.

> You'll want to go ahead and implement this part now, and test it below, before moving on to the next goal.

Sequence labeling with a trained HMM

Now that you're able to train an HMM by estimating the transition matrix A
and the emission matrix B
, you can apply it to the task of labeling a sequence of words $\mathbf{w} = w_1 \cdots w_T$
. Our goal is to find the most probable sequence of tags $\hat{\mathbf{q}} \in Q^T$
given a sequence of words $\mathbf{w} \in \mathcal{V}^T$
.

$$\begin{aligned}
\hat{\mathbf{q}} &= \underset{\mathbf{q} \in Q^T}{\operatorname{argmax}} \left(\Pr(\mathbf{q} \mid \mathbf{w}) \right) \\
&= \underset{\mathbf{q} \in Q^T}{\operatorname{argmax}} \left(\Pr(\mathbf{q}, \mathbf{w}) \right) \\
&= \underset{\mathbf{q} \in Q^T}{\operatorname{argmax}} \left(\Pi_{t=1}^{T} \Pr(w_t \mid q_t) \Pr(q_t \mid q_{t-1}) \right)
\end{aligned}$$

where $\Pr(w_t = \mathcal{V}_j \mid q_t = Q_i) = B_{ij}$
, $\Pr(q_t = Q_j \mid q_{t-1} = Q_i) = A_{ij}$
, and q_0
is the predefined initial tag `TAG.vocab.stoi[TAG.init_token]`.

Goal 1(c): Viterbi algorithm

Implement the `predict` method, which should use the Viterbi algorithm to find the most likely sequence of tags for a sequence of `words`.

> Warning: It may take up to 30 minutes to tag the entire test set depending on your implementation. (A fully tensorized implementation can be much faster though.) We highly recommend that you begin by experimenting with your code using a *very small subset* of the dataset, say two or three sentences, ramping up from there.
>
> Hint: Consider how to use vectorized computations where possible for speed.

Evaluation

We've provided you with the `evaluate` function, which takes a dataset iterator and uses `predict` on each sentence in each batch, comparing against the gold tags, to determine the accuracy of the model on the test set.

In [76]:

```
import pandas as pd
class HMMTagger():
  def __init__ (self, hf_text_tokenizer, hf_tag_tokenizer):
    self.hf_text_tokenizer = hf_text_tokenizer
    self.hf_tag_tokenizer = hf_tag_tokenizer

    self.V = len(self.hf_text_tokenizer)     # vocabulary size
    self.N = len(self.hf_tag_tokenizer)      # state space size
```

```
    self.initial_state_id = self.hf_tag_tokenizer.bos_token_id
    self.initial_tag_id = self.hf_text_tokenizer.bos_token_id
    self.pad_state_id = self.hf_tag_tokenizer.pad_token_id
    self.pad_word_id = self.hf_text_tokenizer.pad_token_id
  def train_A(self, iterator, delta):
    # Create transition_counts tensor
    transition_counts = torch.zeros(self.N, self.N, device=device)

    # Count transitions
    for batch in iterator:

        tags = batch['tag_ids']# Remove initial and final tags

        for tag_sequence in tags:

           for i in range(len(tag_sequence) - 1):
            transition = tag_sequence[i]
            next_transition = tag_sequence[i + 1]

            if transition== self.pad_state_id and next_transition == self.pad_state_id:



               transition_counts[transition, next_transition] = 0
            else:

                transition_counts[transition, next_transition] +=1

    # Apply add-delta smoothing



    # Normalize probabilities
    A =(transition_counts+delta) / (transition_counts.sum(dim=1, keepdim=True) +( delta
* self.N))


    return A





  def train_B(self, iterator, delta):
     """Returns B for training dataset `iterator` using add-`delta` smoothing."""
     # Create B
     B = torch.zeros(self.N, self.V, device=device)
     tag_counts = count_tags(iterator)

     for batch in iterator:
            words = batch['input_ids']
            tags = batch['tag_ids']

            for word_sequence, tag_sequence in zip(words, tags):
              word_sequence = word_sequence[word_sequence.ne(self.pad_word_id)]

              for word, tag in zip(word_sequence, tag_sequence):


                      B[tag, word] +=1
        # Apply add-delta smoothing

        # Normalize probabilities
     row_sums = B.sum(dim=1, keepdim=True)
     B =(B+delta)/(row_sums +( delta * self.V) )

     return B


  def train_all(self, iterator, delta=0.01):
    """Stores A and B (actually, their logs) for training dataset `iterator`."""
    self.log_A = self.train_A(iterator, delta).log()
```

```python
    self.log_B = self.train_B(iterator, delta).log()





  def predict(self, words):
    """Returns the most likely sequence of tags for a sequence of `words`.
    Arguments:
      words: a tensor of size (seq_len,)
    Returns:
      a list of tag ids
    """
    total_states = self.N
    time = len(words)

    viterbi = torch.zeros(total_states, time).to(device)
    backTrack = torch.zeros(total_states, time, dtype=torch.long).to(device)

    log_A = self.log_A.to(device)  # Assuming log_A is the transition matrix in logarith
mic form
    log_B = self.log_B.to(device)  # Assuming log_B is the emission matrix in logarithmi
c form

    words = words.to(device)

    # Initialization

    viterbi[:, 0] = log_A[ 0, :] + log_B[:, words[0]]
    backTrack[:, 0] = 0

    # Recursion
    for t in range(1, time):
        for i in range(total_states):
            prob = viterbi[:, t - 1] + log_A[:, i] + log_B[i, words[t]]

            viterbi[i, t] = torch.max(prob)
            backTrack[i, t] = torch.argmax(prob)

    # Trace back the best path
    best_path = []
    best_state = torch.argmax(viterbi[:, -1])
    best_path.append(best_state.item())

    for t in range(time - 1, 0, -1):
        best_state = backTrack[best_state, t]
        best_path.insert(0, best_state.item())

    return best_path





  def evaluate(self, iterator):
    """Returns the model's token accuracy on a given dataset `iterator`."""
    correct = 0
    total = 0
    for batch in tqdm(iterator, leave=False):
      for sent_id in range(len(batch['input_ids'])):
        words = batch['input_ids'][sent_id]
        words = words[words.ne(self.pad_word_id)] # remove paddings
        tags_gold = batch['tag_ids'][sent_id]
        tags_pred = self.predict(words)
```

```
        for tag_gold, tag_pred in zip(tags_gold, tags_pred):

          if tag_gold == self.pad_state_id:  # stop once we hit padding
            break
          else:

            total += 1
            if tag_pred == tag_gold:
                correct += 1
    return correct/total
```

Putting everything together, you should now be able to train and evaluate the HMM. A correct implementation can be expected to reach above 90% test set accuracy after running the following cell.

In [36]:

```
# Instantiate and train classifier
hmm_tagger = HMMTagger(hf_text_tokenizer, hf_tag_tokenizer)
hmm_tagger.train_all(train_iter)

# Evaluate model performance
print(f'Training accuracy: {hmm_tagger.evaluate(train_iter):.3f}\n'
      f'Test accuracy:     {hmm_tagger.evaluate(test_iter):.3f}')
```

```
Training accuracy: 0.906
Test accuracy:     0.896
```

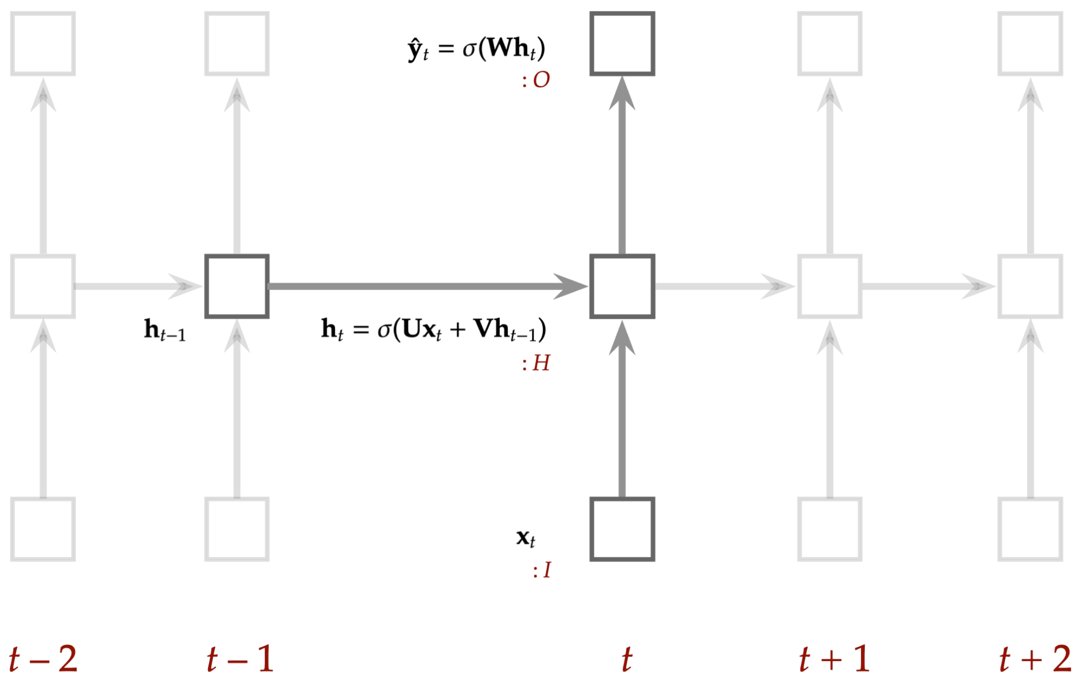
RNN for Sequence Labeling

HMMs work quite well for this sequence labeling task. Now let's take an alternative (and more trendy) approach: RNN/LSTM-based sequence labeling. Similar to the HMM part of this project, you will also need to train a model on the training data, and then use the trained model to decode and evaluate some testing data.

After unfolding an RNN, the cell at time t **generates the observed output** $\mathbf{y}_t$ **based on the input** $\mathbf{x}_t$ **and the hidden state of the previous cell** $\mathbf{h}_{t-1}$ **, according to the following equations.**

$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{x}_t + \mathbf{V}\mathbf{h}_{t-1})$$
$$\hat{\mathbf{y}}_t = \mathrm{softmax}(\mathbf{W}\mathbf{h}_t)$$

The parameters here are the elements of the matrices $\mathbf{U}$ **,** $\mathbf{V}$ **, and** $\mathbf{W}$ **. Similar to the last project segment, we will perform the forward computation, calculate the loss, and then perform the backward computation to compute the gradients with respect to these model parameters. Finally, we will adjust the parameters opposite the direction of the gradients to minimize the loss, repeating until convergence.**



You've seen these kinds of neural network models before, for language modeling in lab 2-3 and sequence labeling in lab 2-5. The code there should be very helpful in implementing an `RNNTagger` **class below. Consequently, we've provided very little guidance on the implementation. We do recommend you follow the steps below however.**

Goal 2(a): RNN training

Implement the forward pass of the RNN tagger and the loss function. A reasonable way to proceed is to implement the following methods:

1. `forward(self, text_batch)`: Performs the RNN forward computation over a whole `text_batch` (`batch.text` in the above data loading example). The `text_batch` will be of shape `max_length x batch_size`. You might run it through the following layers: an embedding layer, which maps each token index to an embedding of size `embedding_size` (so that the size of the mapped batch becomes `max_length x batch_size x embedding_size`); then an RNN, which maps each token embedding to a vector of `hidden_size` (the size of all outputs is `max_length x batch_size x hidden_size`); then a linear layer, which maps each RNN output element to a vector of size N (which is commonly referred to as "logits", recall that $N = |Q|$, the size of the tag set).

This function is expected to return `logits`, which provides a logit for each tag of each word of each sentence in the batch (structured as a tensor of size `max_length x batch_size x N`).

> You might find the following functions useful:
>
> - `nn.Embedding`
> - `nn.Linear`
> - `nn.RNN`

1. `compute_loss(self, logits, tags)`: Computes the loss for a batch by comparing `logits` of a batch returned by `forward` to `tags`, which stores the true tag ids for the batch. Thus `logits` is a tensor of size `max_length x batch_size x N`, and `tags` is a tensor of size `max_length x batch_size`. Note that the criterion functions in `torch` expect outputs of a certain shape, so you might need to perform some shape conversions.

> You might find `nn.CrossEntropyLoss` from the last project segment useful. Note that if you use `nn.CrossEntropyLoss` then you should not use a softmax layer at the end since that's already absorbed into the loss function. Alternatively, you can use `nn.LogSoftmax` as the final sublayer in the forward pass, but then you need to use `nn.NLLLoss`, which does not contain its own softmax. We recommend the former, since working in log space is usually more numerically stable.
>
> Be careful about the shapes/dimensions of tensors. You might find `torch.Tensor.view` useful for reshaping tensors.

1. `train_all(self, train_iter, val_iter, epochs=10, learning_rate=0.001)`: Trains the model on training data generated by the iterator `train_iter` and validation data `val_iter`.The `epochs` and `learning_rate` variables are the number of epochs (number of times to run through the training data) to run for and the learning rate for the optimizer, respectively. You can use the validation data to determine which model was the best one as the epocks go by. Notice that our code below assumes that during training the best model is stored so that `rnn_tagger.load_state_dict(rnn_tagger.best_model)` restores the parameters of the best model.

Goal 2(b) RNN decoding

Implement a method to predict the tag sequence associated with a sequence of words:

1. `predict(self, text_batch)`: Returns the batched predicted tag sequences associated with a batch of sentences.
2. `def evaluate(self, iterator)`: Returns the accuracy of the trained tagger on a dataset provided by `iterator`.

In [37]:

```
import torch
import torch.nn as nn
import torch.optim as optim

class RNNTagger(nn.Module):
    def __init__(self, hf_text_tokenizer, hf_tag_tokenizer, embedding_size, hidden_size)
:
        super(RNNTagger, self).__init__()
        self.hf_text_tokenizer = hf_text_tokenizer
        self.hf_tag_tokenizer = hf_tag_tokenizer
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size

        self.V = len(self.hf_text_tokenizer)    # vocabulary size
        self.N = len(self.hf_tag_tokenizer)     # state space size

        self.embedding = nn.Embedding(self.V, self.embedding_size)
        self.rnn = nn.RNN(self.embedding_size, self.hidden_size, batch_first=True)
        self.linear = nn.Linear(self.hidden_size, self.N)

        self.best_model = None

    def forward(self, text_batch):
        embedded = self.embedding(text_batch)
        output, _ = self.rnn(embedded)
        logits = self.linear(output)
        return logits

    def compute_loss(self, logits, tags):
        loss_fn = nn.CrossEntropyLoss()
        logits_flat = logits.view(-1, self.N)
        tags_flat = tags.view(-1)
        loss = loss_fn(logits_flat, tags_flat)
        return loss

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

        best_val_loss = float('inf')

        for epoch in range(epochs):
            self.train()

            for batch in train_iter:
                optimizer.zero_grad()
                text_batch = batch['input_ids']  # Pass text_batch through the text toke
nizer
                tag_batch = batch['tag_ids']
                logits = self.forward(text_batch)
                loss = self.compute_loss(logits, tag_batch)
                loss.backward()
                optimizer.step()

            self.eval()

            with torch.no_grad():
                val_loss = 0
                for batch in val_iter:
                    text_batch = batch['input_ids']  # Pass text_batch through the text
tokenizer
                    tag_batch = batch['tag_ids']
                    logits = self.forward(text_batch)
                    loss = self.compute_loss(logits, tag_batch)
                    val_loss += loss.item()

                val_loss /= len(val_iter)

                if val_loss < best_val_loss:
```

```
                    best_val_loss = val_loss
                    self.best_model = self.state_dict()
    def evaluate(self, iterator):
        total_correct = 0
        total_count = 0

        self.eval()

        with torch.no_grad():
            for batch in iterator:
                text_batch = batch['input_ids']   # Pass text_batch through the text to
kenizer
                tag_batch = batch['tag_ids']
                logits = self.forward(text_batch)
                _, predictions = torch.max(logits, dim=2)

                # Flatten the predictions and tags tensors
                predictions = predictions.view(-1)
                tags = tag_batch.view(-1)

                # Count the number of correct predictions
                correct = (predictions == tags).sum().item()
                total_correct += correct
                total_count += len(tags)

        accuracy = total_correct / total_count
        return accuracy







    def load_best_model(self):
        if self.best_model is not None:
            self.load_state_dict(self.best_model)
```

Now train your tagger on the training and validation set. Run the cell below to train an RNN, and evaluate it. A proper implementation should reach about 95%+ accuracy.

In [38]:

```
# Instantiate and train classifier
rnn_tagger = RNNTagger(hf_text_tokenizer, hf_tag_tokenizer, embedding_size=36, hidden_si
ze=36).to(device)
rnn_tagger.train_all(train_iter, val_iter, epochs=10, learning_rate=0.001)
rnn_tagger.load_state_dict(rnn_tagger.best_model)

# Evaluate model performance
print(f'Training accuracy: {rnn_tagger.evaluate(train_iter):.3f}\n'
      f'Test accuracy:     {rnn_tagger.evaluate(test_iter):.3f}')
```

```
Training accuracy: 0.976
Test accuracy:     0.971
```

LSTM for slot filling

Did your RNN perform better than HMM? How much better was it? Was that expected?

RNNs tend to exhibit the [vanishing gradient problem](). To remedy this, the Long-Short Term Memory (LSTM) model was introduced. In PyTorch, we can simply use `nn.LSTM`.

In this section, you'll implement an LSTM model for slot filling. If you've got the RNN model well implemented, this should be extremely straightforward. Just copy and paste your solution, change the call to `nn.RNN` to a call to `nn.LSTM`, and make any other minor adjustments that are necessary. In particular, LSTMs have *two* recurrent parts, `h` and `c`. You'll thus need to initialize both of these when performing forward computations.

In [39]:

```
import torch
import torch.nn as nn
import torch.optim as optim

class LSTMTagger(nn.Module):
    def __init__(self, hf_text_tokenizer, hf_tag_tokenizer, embedding_size, hidden_size)
:
        super(LSTMTagger, self).__init__()
        self.hf_text_tokenizer = hf_text_tokenizer
        self.hf_tag_tokenizer = hf_tag_tokenizer
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size

        self.V = len(self.hf_text_tokenizer)     # vocabulary size
        self.N = len(self.hf_tag_tokenizer)      # state space size

        self.embedding = nn.Embedding(self.V, self.embedding_size)
        self.lstm = nn.LSTM(self.embedding_size, self.hidden_size, batch_first=True)
        self.linear = nn.Linear(self.hidden_size, self.N)

        self.best_model = None

    def forward(self, text_batch):
        embedded = self.embedding(text_batch)
        lstm_out, (h, c) = self.lstm(embedded)
        logits = self.linear(lstm_out)
        return logits

    def compute_loss(self, logits, tags):
        loss_fn = nn.CrossEntropyLoss()
        logits_flat = logits.view(-1, self.N)
        tags_flat = tags.view(-1)
        loss = loss_fn(logits_flat, tags_flat)
        return loss

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

        best_val_loss = float('inf')

        for epoch in range(epochs):
            self.train()

            for batch in train_iter:
                optimizer.zero_grad()
                text_batch = batch['input_ids']  # Pass text_batch through the text toke
nizer
                tag_batch = batch['tag_ids']
                logits = self.forward(text_batch)
                loss = self.compute_loss(logits, tag_batch)
                loss.backward()
                optimizer.step()

            self.eval()

            with torch.no_grad():
                val_loss = 0
                for batch in val_iter:
                    text_batch = batch['input_ids']  # Pass text_batch through the text
tokenizer
                    tag_batch = batch['tag_ids']
                    logits = self.forward(text_batch)
                    loss = self.compute_loss(logits, tag_batch)
                    val_loss += loss.item()

                val_loss /= len(val_iter)

                if val_loss < best_val_loss:
                    best_val_loss = val_loss
```

```
                self.best_model = self.state_dict()
    def evaluate(self, iterator):
        total_correct = 0
        total_count = 0

        self.eval()

        with torch.no_grad():
            for batch in iterator:
                text_batch = batch['input_ids']  # Pass text_batch through the text to
kenizer

                tag_batch = batch['tag_ids']
                logits = self.forward(text_batch)
                _, predictions = torch.max(logits, dim=2)

                # Flatten the predictions and tags tensors
                predictions = predictions.view(-1)
                tags = tag_batch.view(-1)

                # Count the number of correct predictions
                correct = (predictions == tags).sum().item()
                total_correct += correct
                total_count += len(tags)

        accuracy = total_correct / total_count
        return accuracy






    def load_best_model(self):
        if self.best_model is not None:
            self.load_state_dict(self.best_model)
```

Run the cell below to train an LSTM, and evaluate it. A proper implementation should reach about 94%+ accuracy.

In [40]:

```
# Instantiate and train classifier
lstm_tagger = LSTMTagger(hf_text_tokenizer, hf_tag_tokenizer, embedding_size=36, hidden_
size=36).to(device)
lstm_tagger.train_all(train_iter, val_iter, epochs=10, learning_rate=0.001)
lstm_tagger.load_state_dict(lstm_tagger.best_model)

# Evaluate model performance
print(f'Training accuracy: {lstm_tagger.evaluate(train_iter):.3f}\n'
      f'Test accuracy:     {lstm_tagger.evaluate(test_iter):.3f}')
```

```
Training accuracy: 0.971
Test accuracy:     0.967
```

Goal 4: Compare HMM to RNN/LSTM with different amounts of training data

Vary the amount of training data and compare the performance of HMM to RNN or LSTM (Since RNN is similar to LSTM, picking one of them is enough.) Discuss the pros and cons of HMM and RNN/LSTM based on your experiments.

> **This part is more open-ended. We're looking for thoughtful experiments and analysis of the results, not any particular result or conclusion.**

The code below shows how to subsample the training set with downsample ratio `ratio`. To speedup evaluation we only use 50 test samples.

In [92]:

```
import torch


# Set random seeds to make sure subsampling is the same for HMM and RNN
random_seed = 42
torch.manual_seed(random_seed)

# Vary the ratio for subsampling
ratios = [0.5, 0.7, 0.9,0.95]

# Test size
test_size = 50

# Load and split the dataset
atis = load_dataset('csv', data_files={'train': 'data/atis.train.csv',
                                       'val': 'data/atis.dev.csv',
                                       'test': 'data/atis.test.csv'})
train_data = atis['train']
test_data = atis['test']

# Lists to store accuracies
hmm_train_accuracies = []
hmm_test_accuracies = []
rnn_train_accuracies = []
rnn_test_accuracies = []

# Iterate over different ratios
for ratio in ratios:
    # Subsample the training data
    train_data_subsampled = train_data.shuffle(seed=random_seed).select(range(int(len(tr
ain_data) * ratio)))

    # Shuffle and select a fixed number of test samples for evaluation
    test_data_subsampled = test_data.shuffle(seed=random_seed).select(range(test_size))

    # Rebuild vocabulary
    text_tokenizer, tag_tokenizer = train_tokenizers(train_data_subsampled, MIN_FREQ)

    # Encode data
    hf_text_tokenizer = PreTrainedTokenizerFast(tokenizer_object=text_tokenizer, pad_toke
n=pad_token)
    hf_tag_tokenizer = PreTrainedTokenizerFast(tokenizer_object=tag_tokenizer, pad_token=
pad_token)

    def encode(example):
        example['input_ids'] = hf_text_tokenizer(example['text']).input_ids
        example['tag_ids'] = hf_tag_tokenizer(example['tag']).input_ids
        return example

    train_data_subsampled = train_data_subsampled.map(encode)
    test_data_subsampled = test_data_subsampled.map(encode)

    # Create iterators
    train_iter, val_iter, test_iter = get_iterators(train_data_subsampled, val_data, test
_data_subsampled)

    # Train and evaluate HMM
    hmm_tagger = HMMTagger(hf_text_tokenizer, hf_tag_tokenizer)

    # Concatenate input data across the batch dimension


    hmm_tagger.train_all(train_iter)
    print("Ratio",ratio)
    print("HMM")
    # Evaluate model performance

    hmm_train_accuracy = hmm_tagger.evaluate(train_iter)
```

```
    hmm_test_accuracy = hmm_tagger.evaluate(test_iter)
    print(f'Training accuracy: {hmm_train_accuracy:.3f}\n'
            f'Test accuracy:     {hmm_test_accuracy:.3f}')
    hmm_train_accuracies.append(hmm_train_accuracy)
    hmm_test_accuracies.append(hmm_test_accuracy)

    # Train and evaluate RNN
    # Instantiate and train classifier
    rnn_tagger = RNNTagger(hf_text_tokenizer, hf_tag_tokenizer, embedding_size=36, hidde
n_size=36).to(device)
    rnn_tagger.train_all(train_iter, test_iter, epochs=10, learning_rate=0.001)
    rnn_tagger.load_state_dict(rnn_tagger.best_model)
    print("RNN")
    # Evaluate model performance

    rnn_train_accuracy = rnn_tagger.evaluate(train_iter)
    rnn_test_accuracy = rnn_tagger.evaluate(test_iter)
    print(f'Training accuracy: {rnn_train_accuracy:.3f}\n'
          f'Test accuracy:     {rnn_test_accuracy:.3f}')
    rnn_train_accuracies.append(rnn_train_accuracy)
    rnn_test_accuracies.append(rnn_test_accuracy)
    print("---------------------")


# Plotting the accuracies
plt.plot(ratios, hmm_train_accuracies, label='HMM Train Accuracy')
plt.plot(ratios, hmm_test_accuracies, label='HMM Test Accuracy')
plt.plot(ratios, rnn_train_accuracies, label='RNN Train Accuracy')
plt.plot(ratios, rnn_test_accuracies, label='RNN Test Accuracy')
plt.xlabel('Ratio')
plt.ylabel('Accuracy')
plt.title('Comparison of HMM and RNN Accuracies')
plt.legend()
plt.show()
```

```
WARNING:datasets.builder:Using custom data configuration default-a31f1de2f7fa41be
WARNING:datasets.builder:Found cached dataset csv (/root/.cache/huggingface/datasets/csv/
default-a31f1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bfbe43546f190a60fcf74bb5e8afd
cc2317)
```

```
WARNING:datasets.arrow_dataset:Loading cached shuffled indices for dataset at /root/.cach
e/huggingface/datasets/csv/default-a31f1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bf
be43546f190a60fcf74bb5e8afdcc2317/cache-47afa9512d73ac5f.arrow
WARNING:datasets.arrow_dataset:Loading cached shuffled indices for dataset at /root/.cach
e/huggingface/datasets/csv/default-a31f1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bf
be43546f190a60fcf74bb5e8afdcc2317/cache-171ea4b5b879c54f.arrow
WARNING:datasets.arrow_dataset:Loading cached processed dataset at /root/.cache/huggingfa
ce/datasets/csv/default-a31f1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bfbe43546f190
a60fcf74bb5e8afdcc2317/cache-0a6a3cc3c0f12c7b.arrow
WARNING:datasets.arrow_dataset:Loading cached processed dataset at /root/.cache/huggingfa
ce/datasets/csv/default-a31f1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bfbe43546f190
a60fcf74bb5e8afdcc2317/cache-3fc0e1fe44ac3d9d.arrow
```

```
Ratio 0.5
HMM


Training accuracy: 0.903
Test accuracy:     0.881
RNN
```

```
WARNING:datasets.arrow_dataset:Loading cached shuffled indices for dataset at /root/.cach
e/huggingface/datasets/csv/default-a31f1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bf
be43546f190a60fcf74bb5e8afdcc2317/cache-47afa9512d73ac5f.arrow
WARNING:datasets.arrow_dataset:Loading cached shuffled indices for dataset at /root/.cach
e/huggingface/datasets/csv/default-a31f1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bf
be43546f190a60fcf74bb5e8afdcc2317/cache-171ea4b5b879c54f.arrow
```

```
Training accuracy: 0.961
Test accuracy:     0.958
---------------------
```

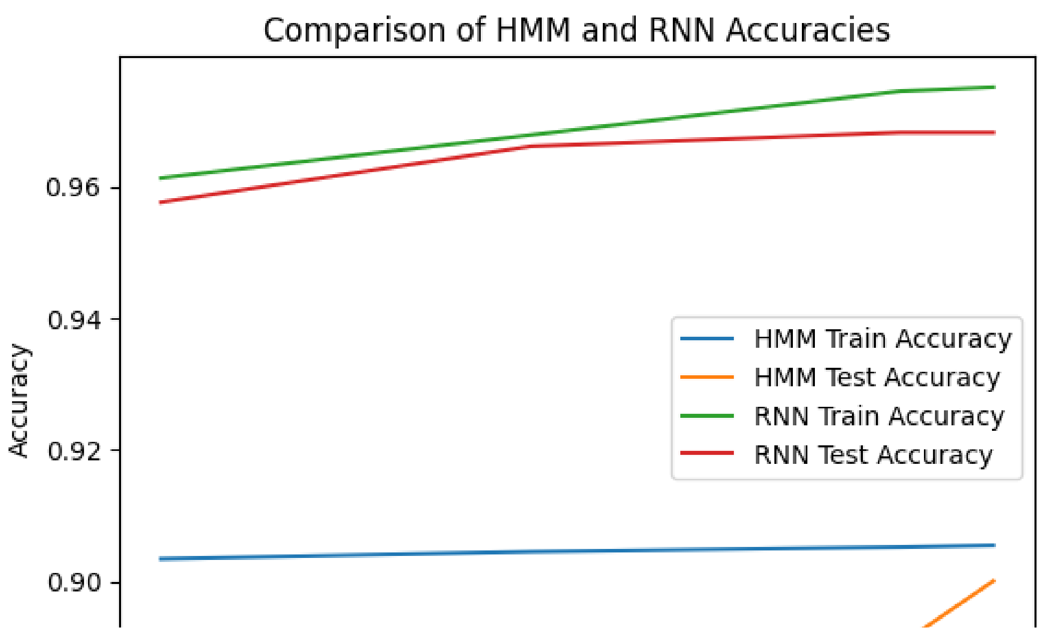
```
Ratio 0.7
HMM


Training accuracy: 0.904
Test accuracy:     0.884
RNN
```

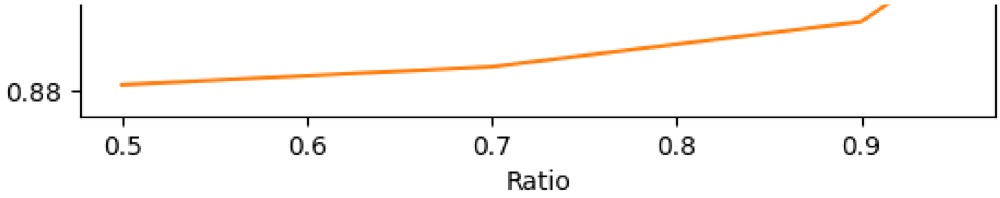
```
WARNING:datasets.arrow_dataset:Loading cached shuffled indices for dataset at /root/.cach
e/huggingface/datasets/csv/default-a31f1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bf
be43546f190a60fcf74bb5e8afdcc2317/cache-47afa9512d73ac5f.arrow
WARNING:datasets.arrow_dataset:Loading cached shuffled indices for dataset at /root/.cach
e/huggingface/datasets/csv/default-a31f1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bf
be43546f190a60fcf74bb5e8afdcc2317/cache-171ea4b5b879c54f.arrow
```

```
Training accuracy: 0.968
Test accuracy:     0.966
---------------------


Ratio 0.9
HMM


Training accuracy: 0.905
Test accuracy:     0.890
RNN
```

```
WARNING:datasets.arrow_dataset:Loading cached shuffled indices for dataset at /root/.cach
e/huggingface/datasets/csv/default-a31f1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bf
be43546f190a60fcf74bb5e8afdcc2317/cache-47afa9512d73ac5f.arrow
WARNING:datasets.arrow_dataset:Loading cached shuffled indices for dataset at /root/.cach
e/huggingface/datasets/csv/default-a31f1de2f7fa41be/0.0.0/6b34fb8fcf56f7c8ba51dc895bfa2bf
be43546f190a60fcf74bb5e8afdcc2317/cache-171ea4b5b879c54f.arrow
```

```
Training accuracy: 0.974
Test accuracy:     0.968
---------------------


Ratio 0.95
HMM


Training accuracy: 0.905
Test accuracy:     0.900
RNN
Training accuracy: 0.975
Test accuracy:     0.968
---------------------
```





Training Accuracy:

HMM: The training accuracy remains relatively stable across different training dataset sizes (ratios). It stays around 0.909, indicating that the HMM model is consistently able to learn the patterns present in the training data.

RNN: The training accuracy consistently improves as the training dataset size increases. This indicates that the RNN model benefits from having access to more training data, allowing it to learn more complex patterns and achieve higher accuracy.

Test Accuracy:

HMM: The test accuracy seems to be increasing dataset sizes, ranging from 0.88 to 0.906. This suggests that the HMM model This suggests that the HMM model benefits from larger training datasets, allowing it to better capture the underlying patterns and improve its generalization performance on unseen data.

RNN: The test accuracy shows a slight increase as the training dataset size increases and then stablilizes.Showing that increasing training size beyond a certain poin thas no effect on the test accuracy

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 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?

Type your answer here, replacing this text.

Instructions for submission of the project segment

This project segment should be submitted to Gradescope at https://rebrand.ly/project2-submit-code and https://rebrand.ly/project2-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/project2-submit-code.

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/project2-submit-pdf.