In [27]:
```
# Monter Google Drive pour accéder à tes fichiers
from google.colab import drive
drive.mount('/content/drive')

# Copier les fichiers depuis Google Drive vers l'environnement Colab
!cp -r /content/drive/MyDrive/nlp_lab/project2* .

# Installer les dépendances depuis le fichier requirements.txt
!pip install -r /content/drive/MyDrive/nlp_lab/project2/requirements.txt

# Installer Otter Grader (si nécessaire)
!pip install otter-grader

# Fonction pour exécuter les commandes shell
import os

def shell(commands, warn=True):
    """Exécute des commandes shell et affiche les résultats."""
    file = os.popen(commands)
    print(file.read().rstrip('\n'))
    exit_status = file.close()
    if warn and exit_status is not None:
        print(f"Command failed with exit code {exit_status}")
    return exit_status

# Vérifier si requirements.txt existe et télécharger le dépôt si nécessaire
shell("""
ls requirements.txt >/dev/null 2>&1
if [ ! $? = 0 ]; then
    rm -rf .tmp
    git clone https://github.com/cs236299-2024-winter/lab1-4.git .tmp
    mv .tmp/tests ./tests
    mv .tmp/requirements.txt ./requirements.txt
    rm -rf .tmp
fi
pip install -q -r requirements.txt
""")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dri
ve.mount("/content/drive", force_remount=True).
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages
(from -r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line 1)) (2.5.
1+cu121)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packa
ges (from -r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line 2))
(3.10.0)
Requirement already satisfied: otter-grader==1.0.0 in /usr/local/lib/python3.10/d
ist-packages (from -r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (l
ine 3)) (1.0.0)
Requirement already satisfied: wget in /usr/local/lib/python3.10/dist-packages (f
rom -r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line 4)) (3.2)
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-pac
kages (from -r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line 5))
(4.47.1)
Requirement already satisfied: datasets in /usr/local/lib/python3.10/dist-package
s (from -r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line 6)) (3.
2.0)
Requirement already satisfied: tokenizers in /usr/local/lib/python3.10/dist-packa
ges (from -r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line 7))
(0.21.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages
(from otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/requirement
s.txt (line 3)) (6.0.2)
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-package
s (from otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/requireme
nts.txt (line 3)) (5.10.4)
Requirement already satisfied: ipython in /usr/local/lib/python3.10/dist-packages
(from otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/requirement
s.txt (line 3)) (7.34.0)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-packag
es (from otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/requirem
ents.txt (line 3)) (7.16.5)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (f
rom otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/requirements.
txt (line 3)) (4.67.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packa
ges (from otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/require
ments.txt (line 3)) (75.1.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/requirement
s.txt (line 3)) (2.2.2)
Requirement already satisfied: tornado in /usr/local/lib/python3.10/dist-packages
(from otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/requirement
s.txt (line 3)) (6.3.3)
Requirement already satisfied: docker in /usr/local/lib/python3.10/dist-packages
(from otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/requirement
s.txt (line 3)) (7.1.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/requirement
s.txt (line 3)) (3.1.5)
Requirement already satisfied: dill in /usr/local/lib/python3.10/dist-packages (f
rom otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/requirements.
txt (line 3)) (0.3.8)
Requirement already satisfied: pdfkit in /usr/local/lib/python3.10/dist-packages
(from otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/requirement
s.txt (line 3)) (1.0.0)
Requirement already satisfied: PyPDF2 in /usr/local/lib/python3.10/dist-packages
(from otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/requirement
```

```
s.txt (line 3)) (3.0.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-package
s (from torch->-r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line
1)) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python
3.10/dist-packages (from torch->-r /content/drive/MyDrive/nlp_lab/project2/requir
ements.txt (line 1)) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-package
s (from torch->-r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line
1)) (3.4.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch->-r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line
1)) (2024.9.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-pa
ckages (from torch->-r /content/drive/MyDrive/nlp_lab/project2/requirements.txt
(line 1)) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/di
st-packages (from sympy==1.13.1->torch->-r /content/drive/MyDrive/nlp_lab/project
2/requirements.txt (line 1)) (1.3.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist
-packages (from matplotlib->-r /content/drive/MyDrive/nlp_lab/project2/requiremen
ts.txt (line 2)) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-pac
kages (from matplotlib->-r /content/drive/MyDrive/nlp_lab/project2/requirements.t
xt (line 2)) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dis
t-packages (from matplotlib->-r /content/drive/MyDrive/nlp_lab/project2/requireme
nts.txt (line 2)) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.10/dis
t-packages (from matplotlib->-r /content/drive/MyDrive/nlp_lab/project2/requireme
nts.txt (line 2)) (1.4.8)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.10/dist-pack
ages (from matplotlib->-r /content/drive/MyDrive/nlp_lab/project2/requirements.tx
t (line 2)) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->-r /content/drive/MyDrive/nlp_lab/project2/requirement
s.txt (line 2)) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib->-r /content/drive/MyDrive/nlp_lab/project2/requirements.txt
(line 2)) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist
-packages (from matplotlib->-r /content/drive/MyDrive/nlp_lab/project2/requiremen
ts.txt (line 2)) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/
dist-packages (from matplotlib->-r /content/drive/MyDrive/nlp_lab/project2/requir
ements.txt (line 2)) (2.8.2)
Requirement already satisfied: huggingface-hub<1.0,>=0.24.0 in /usr/local/lib/pyt
hon3.10/dist-packages (from transformers->-r /content/drive/MyDrive/nlp_lab/proje
ct2/requirements.txt (line 5)) (0.27.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dis
t-packages (from transformers->-r /content/drive/MyDrive/nlp_lab/project2/require
ments.txt (line 5)) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-package
s (from transformers->-r /content/drive/MyDrive/nlp_lab/project2/requirements.txt
(line 5)) (2.32.3)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/di
st-packages (from transformers->-r /content/drive/MyDrive/nlp_lab/project2/requir
ements.txt (line 5)) (0.5.0)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.10/dist-
packages (from datasets->-r /content/drive/MyDrive/nlp_lab/project2/requirements.
```

```
txt (line 6)) (17.0.0)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages
(from datasets->-r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line
6)) (3.5.0)
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.10/
dist-packages (from datasets->-r /content/drive/MyDrive/nlp_lab/project2/requirem
ents.txt (line 6)) (0.70.16)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages
(from datasets->-r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line
6)) (3.11.11)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.
10/dist-packages (from aiohttp->datasets->-r /content/drive/MyDrive/nlp_lab/proje
ct2/requirements.txt (line 6)) (2.4.4)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist
-packages (from aiohttp->datasets->-r /content/drive/MyDrive/nlp_lab/project2/req
uirements.txt (line 6)) (1.3.2)
Requirement already satisfied: async-timeout<6.0,>=4.0 in /usr/local/lib/python3.
10/dist-packages (from aiohttp->datasets->-r /content/drive/MyDrive/nlp_lab/proje
ct2/requirements.txt (line 6)) (4.0.3)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-pa
ckages (from aiohttp->datasets->-r /content/drive/MyDrive/nlp_lab/project2/requir
ements.txt (line 6)) (24.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dis
t-packages (from aiohttp->datasets->-r /content/drive/MyDrive/nlp_lab/project2/re
quirements.txt (line 6)) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/d
ist-packages (from aiohttp->datasets->-r /content/drive/MyDrive/nlp_lab/project2/
requirements.txt (line 6)) (6.1.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.10/dist
-packages (from aiohttp->datasets->-r /content/drive/MyDrive/nlp_lab/project2/req
uirements.txt (line 6)) (0.2.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.10/dis
t-packages (from aiohttp->datasets->-r /content/drive/MyDrive/nlp_lab/project2/re
quirements.txt (line 6)) (1.18.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-package
s (from python-dateutil>=2.7->matplotlib->-r /content/drive/MyDrive/nlp_lab/proje
ct2/requirements.txt (line 2)) (1.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python
3.10/dist-packages (from requests->transformers->-r /content/drive/MyDrive/nlp_la
b/project2/requirements.txt (line 5)) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pac
kages (from requests->transformers->-r /content/drive/MyDrive/nlp_lab/project2/re
quirements.txt (line 5)) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/di
st-packages (from requests->transformers->-r /content/drive/MyDrive/nlp_lab/proje
ct2/requirements.txt (line 5)) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/di
st-packages (from requests->transformers->-r /content/drive/MyDrive/nlp_lab/proje
ct2/requirements.txt (line 5)) (2024.12.14)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.10/dist-packa
ges (from ipython->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project
2/requirements.txt (line 3)) (0.19.2)
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packag
es (from ipython->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project
2/requirements.txt (line 3)) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-pack
ages (from ipython->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/projec
t2/requirements.txt (line 3)) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.10/dist-p
ackages (from ipython->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/pro
```

```
ject2/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 /u
sr/local/lib/python3.10/dist-packages (from ipython->otter-grader==1.0.0->-r /con
tent/drive/MyDrive/nlp_lab/project2/requirements.txt (line 3)) (3.0.48)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-package
s (from ipython->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/
requirements.txt (line 3)) (2.18.0)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-package
s (from ipython->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/project2/
requirements.txt (line 3)) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dis
t-packages (from ipython->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/
project2/requirements.txt (line 3)) (0.1.7)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-pack
ages (from ipython->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/projec
t2/requirements.txt (line 3)) (4.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-
packages (from jinja2->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/pro
ject2/requirements.txt (line 3)) (3.0.2)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-p
ackages (from nbconvert->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/p
roject2/requirements.txt (line 3)) (4.12.3)
Requirement already satisfied: bleach!=5.0.0 in /usr/local/lib/python3.10/dist-pa
ckages (from bleach[css]!=5.0.0->nbconvert->otter-grader==1.0.0->-r /content/driv
e/MyDrive/nlp_lab/project2/requirements.txt (line 3)) (6.2.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packa
ges (from nbconvert->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/proje
ct2/requirements.txt (line 3)) (0.7.1)
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.10/dis
t-packages (from nbconvert->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_la
b/project2/requirements.txt (line 3)) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/d
ist-packages (from nbconvert->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_
lab/project2/requirements.txt (line 3)) (0.3.0)
Requirement already satisfied: mistune<4,>=2.0.3 in /usr/local/lib/python3.10/dis
t-packages (from nbconvert->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_la
b/project2/requirements.txt (line 3)) (3.1.0)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-
packages (from nbconvert->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/
project2/requirements.txt (line 3)) (0.10.2)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/
dist-packages (from nbconvert->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp
_lab/project2/requirements.txt (line 3)) (1.5.1)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.10/
dist-packages (from nbformat->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_
lab/project2/requirements.txt (line 3)) (2.21.1)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-
packages (from nbformat->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/p
roject2/requirements.txt (line 3)) (4.23.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-pac
kages (from pandas->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/projec
t2/requirements.txt (line 3)) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-p
ackages (from pandas->otter-grader==1.0.0->-r /content/drive/MyDrive/nlp_lab/proj
ect2/requirements.txt (line 3)) (2024.2)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-pac
kages (from bleach!=5.0.0->bleach[css]!=5.0.0->nbconvert->otter-grader==1.0.0->-r
/content/drive/MyDrive/nlp_lab/project2/requirements.txt (line 3)) (0.5.1)
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in /usr/local/lib/python3.10/
dist-packages (from bleach[css]!=5.0.0->nbconvert->otter-grader==1.0.0->-r /conte
```

```
nt/drive/MyDrive/nlp_lab/project2/requirements.txt (line 3)) (1.4.0)
Requirement already satisfied: parso<0.9.0,>=0.8.4 in /usr/local/lib/python3.10/d
ist-packages (from jedi>=0.16->ipython->otter-grader==1.0.0->-r /content/drive/My
Drive/nlp_lab/project2/requirements.txt (line 3)) (0.8.4)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/loca
l/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->otter-grader==1.
0.0->-r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line 3)) (2024.
10.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/d
ist-packages (from jsonschema>=2.6->nbformat->otter-grader==1.0.0->-r /content/dr
ive/MyDrive/nlp_lab/project2/requirements.txt (line 3)) (0.35.1)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-p
ackages (from jsonschema>=2.6->nbformat->otter-grader==1.0.0->-r /content/drive/M
yDrive/nlp_lab/project2/requirements.txt (line 3)) (0.22.3)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dis
t-packages (from jupyter-core>=4.7->nbconvert->otter-grader==1.0.0->-r /content/d
rive/MyDrive/nlp_lab/project2/requirements.txt (line 3)) (4.3.6)
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.1
0/dist-packages (from nbclient>=0.5.0->nbconvert->otter-grader==1.0.0->-r /conten
t/drive/MyDrive/nlp_lab/project2/requirements.txt (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/drive/MyDri
ve/nlp_lab/project2/requirements.txt (line 3)) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages
(from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython->otter-grader==1.0.0-
>-r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line 3)) (0.2.13)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-pa
ckages (from beautifulsoup4->nbconvert->otter-grader==1.0.0->-r /content/drive/My
Drive/nlp_lab/project2/requirements.txt (line 3)) (2.6)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-packag
es (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert->otter-grader==1.0.0-
>-r /content/drive/MyDrive/nlp_lab/project2/requirements.txt (line 3)) (24.0.1)
Requirement already satisfied: otter-grader in /usr/local/lib/python3.10/dist-pac
kages (1.0.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages
(from otter-grader) (6.0.2)
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-package
s (from otter-grader) (5.10.4)
Requirement already satisfied: ipython in /usr/local/lib/python3.10/dist-packages
(from otter-grader) (7.34.0)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-packag
es (from otter-grader) (7.16.5)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (f
rom otter-grader) (4.67.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packa
ges (from otter-grader) (75.1.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from otter-grader) (2.2.2)
Requirement already satisfied: tornado in /usr/local/lib/python3.10/dist-packages
(from otter-grader) (6.3.3)
Requirement already satisfied: docker in /usr/local/lib/python3.10/dist-packages
(from otter-grader) (7.1.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from otter-grader) (3.1.5)
Requirement already satisfied: dill in /usr/local/lib/python3.10/dist-packages (f
rom otter-grader) (0.3.8)
Requirement already satisfied: pdfkit in /usr/local/lib/python3.10/dist-packages
(from otter-grader) (1.0.0)
Requirement already satisfied: PyPDF2 in /usr/local/lib/python3.10/dist-packages
(from otter-grader) (3.0.1)
```

```
Requirement already satisfied: requests>=2.26.0 in /usr/local/lib/python3.10/dist
-packages (from docker->otter-grader) (2.32.3)
Requirement already satisfied: urllib3>=1.26.0 in /usr/local/lib/python3.10/dist-
packages (from docker->otter-grader) (2.3.0)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.10/dist-packa
ges (from ipython->otter-grader) (0.19.2)
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packag
es (from ipython->otter-grader) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-pack
ages (from ipython->otter-grader) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.10/dist-p
ackages (from ipython->otter-grader) (5.7.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /u
sr/local/lib/python3.10/dist-packages (from ipython->otter-grader) (3.0.48)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-package
s (from ipython->otter-grader) (2.18.0)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-package
s (from ipython->otter-grader) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dis
t-packages (from ipython->otter-grader) (0.1.7)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-pack
ages (from ipython->otter-grader) (4.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-
packages (from jinja2->otter-grader) (3.0.2)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-p
ackages (from nbconvert->otter-grader) (4.12.3)
Requirement already satisfied: bleach!=5.0.0 in /usr/local/lib/python3.10/dist-pa
ckages (from bleach[css]!=5.0.0->nbconvert->otter-grader) (6.2.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packa
ges (from nbconvert->otter-grader) (0.7.1)
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.10/dis
t-packages (from nbconvert->otter-grader) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/d
ist-packages (from nbconvert->otter-grader) (0.3.0)
Requirement already satisfied: mistune<4,>=2.0.3 in /usr/local/lib/python3.10/dis
t-packages (from nbconvert->otter-grader) (3.1.0)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-
packages (from nbconvert->otter-grader) (0.10.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packag
es (from nbconvert->otter-grader) (24.2)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/
dist-packages (from nbconvert->otter-grader) (1.5.1)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.10/
dist-packages (from nbformat->otter-grader) (2.21.1)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-
packages (from nbformat->otter-grader) (4.23.0)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-pa
ckages (from pandas->otter-grader) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.1
0/dist-packages (from pandas->otter-grader) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-pac
kages (from pandas->otter-grader) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-p
ackages (from pandas->otter-grader) (2024.2)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-pac
kages (from bleach!=5.0.0->bleach[css]!=5.0.0->nbconvert->otter-grader) (0.5.1)
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in /usr/local/lib/python3.10/
dist-packages (from bleach[css]!=5.0.0->nbconvert->otter-grader) (1.4.0)
Requirement already satisfied: parso<0.9.0,>=0.8.4 in /usr/local/lib/python3.10/d
ist-packages (from jedi>=0.16->ipython->otter-grader) (0.8.4)
```

```
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-pa
ckages (from jsonschema>=2.6->nbformat->otter-grader) (24.3.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/loca
l/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->otter-grader) (20
24.10.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/d
ist-packages (from jsonschema>=2.6->nbformat->otter-grader) (0.35.1)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-p
ackages (from jsonschema>=2.6->nbformat->otter-grader) (0.22.3)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dis
t-packages (from jupyter-core>=4.7->nbconvert->otter-grader) (4.3.6)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dis
t-packages (from mistune<4,>=2.0.3->nbconvert->otter-grader) (4.12.2)
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.1
0/dist-packages (from nbclient>=0.5.0->nbconvert->otter-grader) (6.1.12)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-
packages (from pexpect>4.3->ipython->otter-grader) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages
(from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython->otter-grader) (0.2.1
3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-package
s (from python-dateutil>=2.8.2->pandas->otter-grader) (1.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python
3.10/dist-packages (from requests>=2.26.0->docker->otter-grader) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pac
kages (from requests>=2.26.0->docker->otter-grader) (3.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/di
st-packages (from requests>=2.26.0->docker->otter-grader) (2024.12.14)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-pa
ckages (from beautifulsoup4->nbconvert->otter-grader) (2.6)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-packag
es (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert->otter-grader) (24.0.
1)

Command failed with exit code 256
```

Out[27]: 256

In [28]:
```python
# 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}{\,|\,}

Course 236299

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, for example, *part-of-speech tagging*, which you explored in (optional) lab 2-6. 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 often referred to 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`
`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 as in the last project segment.

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

Goals

1. (Optional, Bonus points) 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 the different models with different amounts of training data. Discuss the pros and cons of the each approach.

Setup

In [29]:
```
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 [30]:
```
# Set random seeds
seed = 1234

def reseed(seed=seed):
    random.seed(seed)
    torch.manual_seed(seed)

reseed()

# 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 [31]:
```
# Prepare to download needed data
def download_if_needed(source, dest, filename):
    os.makedirs(data_path, exist_ok=True)  # ensure destination
    if os.path.exists(f"./{dest}{filename}"):
        print(f"Skipping {filename}")
    else:
        print(f"Downloading {filename} from {source}")
        wget.download(source + filename, out=dest)
        print("", flush=True)


source_path = "https://raw.githubusercontent.com/" \
              "nlp-236299/data/master/ATIS/"
data_path = "data/"

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

```
Skipping atis.train.txt
Skipping atis.dev.txt
Skipping atis.test.txt
```

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 [32]:
```
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 the data files look like.

In [33]:
```
shell('head -n 3 "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_number 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-arrive_time.time I-arrive_time.time B-arrive_date.date_relative B-arrive_date.day_name O
```

Each is a CSV file where

- The first column contains a string of whitespace-separated tokens, demarcated with beginning of sentence (`BOS`) and end of sentence (`EOS`) tokens; and
- The second column contains the corresponding tags for each of the tokens.

We use Huggingface's `datasets` to prepare the data.

In [34]:
```
atis = load_dataset(
    "csv",
    data_files={
        "train": "data/atis.train.csv",
        "val": "data/atis.dev.csv",
        "test": "data/atis.test.csv",
    },
)
atis
```

```
Generating train split: 0 examples [00:00, ? examples/s]
Generating val split: 0 examples [00:00, ? examples/s]
Generating test split: 0 examples [00:00, ? examples/s]
```

Out[34]:
```
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 [35]:
```
train_data = atis['train']
val_data = atis['val']
test_data = atis['test']

train_data.shuffle(seed=seed)
```

Out[35]:
```
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 [36]:
```
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, unk_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_tok

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

    tag_tokenizer.post_processor = TemplateProcessing(
        single=f"{bos_token} $A",
        special_tokens=[(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 [37]:
```
hf_text_tokenizer = PreTrainedTokenizerFast(
    tokenizer_object=text_tokenizer,
    pad_token=pad_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,
```

```
)


def encode(example):
    """Encodes an example by tokenizing the text and converting to ids,
    and similarly for the tags, adding them under appropriate keys
    """
    example["input_ids"] = hf_text_tokenizer(example["text"]).input_ids
    example["tag_ids"] = hf_tag_tokenizer(example["tag"]).input_ids
    return example


# Encode the three datasets into ids
train_data = train_data.map(encode)
val_data = val_data.map(encode)
test_data = test_data.map(encode)
```

```
Map:   0%|          | 0/4274 [00:00<?, ? examples/s]
Map:   0%|          | 0/572 [00:00<?, ? examples/s]
Map:   0%|          | 0/586 [00:00<?, ? examples/s]
```

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

In [38]:
```
# Extract the text and tag vocabularies
text_vocab = text_tokenizer.get_vocab()
tag_vocab = tag_tokenizer.get_vocab()

# Compute size of vocabularies
vocab_size = len(text_vocab)
num_tags = len(tag_vocab)

print(f"Size of text vocabulary: {vocab_size}")
print(f"Size of tag vocabulary:  {num_tags}")
```

```
Size of text vocabulary: 518
Size of tag vocabulary:  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 [39]:
```
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 will 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 [40]:
```
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`. We use a non-trivial batch size to gain the benefit of training on multiple examples at a shot. You'll need to be careful about the shapes of the various tensors that are being manipulated.

In [41]:
```
BATCH_SIZE = 32  # batch size

# Defines how to prepare a list of examples to form a batch
def collate_fn(examples):
    batch = {}
    bsz = len(examples)  # no. of examples in the batch, which may be less
    # than BATCH_SIZE in the final batch
    input_ids, tag_ids = [], []
    for example in examples:
        input_ids.append(example["input_ids"])
        tag_ids.append(example["tag_ids"])

    # pad all the examples to be the size of the longest
    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
    )
    text_batch = (
        torch.zeros(bsz, max_length).long().fill_(text_vocab[pad_token]).to(devi
    )
    # tensorize the text and tag sequences
    for b in range(bsz):
        text_batch[b][: len(input_ids[b])] = torch.LongTensor(input_ids[b]).to(d
        tag_batch[b][: len(tag_ids[b])] = torch.LongTensor(tag_ids[b]).to(device
    # create the batch
    batch["input_ids"] = text_batch
    batch["tag_ids"] = tag_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. Each batch will be a tensor of size `batch_size x max_length`. Let's examine a batch.

In [42]:
```
# 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 au
gust eighteenth 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-depart_date.month_name B-depart_date.day_number O [PAD] [PAD] [PAD] [PAD] [PAD]
```

The goal of this project is thus 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 [43]:
```
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}  "
          f"{hf_tag_tokenizer.decode(tag_id):30}"
          f"{tag_counts[tag_id].item():>7.0f}"
    )
```

```
 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 [44]:
```
tag_counts_test = count_tags(test_iter)
majority_baseline_accuracy = tag_counts_test[tag_vocab["O"]] / tag_counts_test.s
print(f"Baseline accuracy: {majority_baseline_accuracy:.3f}")
```

```
Baseline accuracy: 0.634
```

We could try a more sophisticated version of majority class labeling, where we used the majority class tag of each token type, but the gains would not be great. We'll just move on to some more sophisticated methods.

HMM for sequence labeling (Optional, Bonus points)

Having established the baseline to beat, we turn to implementing an HMM model. **The implementation will be in the HMM class below.** Before getting there, however, we summarize all of the aspects that you'll be implementing in that class.

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,

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

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

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

> As is typical in notating probabilities, we'll use abbreviations
>
> $$\Pr(q_{t+1} \,|\, q_t) \equiv \Pr(q_{t+1} = Q_j \,|\, q_t = Q_i) \quad (1)$$
> $$\Pr(w_t \,|\, q_t) \equiv \Pr(w_t = \mathcal{V}_j \,|\, q_t = Q_i) \quad (2)$$
>
> 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.

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

(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.

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

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 $\mathbf{w} \in \mathcal{V}^T$.

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

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 `hf_tag_tokenizer.bos_token_id`.

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 [45]:

```
class HMMTagger:
    def __init__(self, hf_text_tokenizer, hf_tag_tokenizer):
        self.hf_text_tokenizer = hf_text_tokenizer  # text tokenizer
        self.hf_tag_tokenizer = hf_tag_tokenizer  # tag tokenizer

        self.V = len(self.hf_text_tokenizer)  # vocabulary size (how many word t
        self.N = len(self.hf_tag_tokenizer)  # state space size (how many tag ty

        self.initial_state_id = self.hf_tag_tokenizer.bos_token_id  # start with
        self.pad_state_id = (
            self.hf_tag_tokenizer.pad_token_id
        )  # pad tokens for text and tags
        self.pad_word_id = self.hf_text_tokenizer.pad_token_id

    def train_A(self, iterator, delta):
        """Returns A for training dataset `iterator` using add-`delta` smoothing
        # Create A table to fill in
        A = torch.zeros(self.N, self.N, device=device)

        # TODO: Add your solution from Goal 1(a) here.
        #      The returned value should be a tensor for the A matrix
        #      of size N x N.

        ...

        return A

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

        # TODO: Add your solution from Goal 1 (b) here.
```

```
        #      The returned value should be a tensor for the $B$ matrix
        #      of size N x V.

        ...

        return B

    def train_all(self, train_iter, val_iter, delta=0.01):
        """Stores A and B (actually, their logs) for training dataset `train_ite
        Ignores `val_iter`, which is provided for for consistency with other
        models."""
        self.log_A = self.train_A(train_iter, delta).log()
        self.log_B = self.train_B(train_iter, 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
        """
        # TODO: Add your solution from Goal 1 (c) here.
        #      The returned value should be a list of tag ids.

        ...

        return bestpath

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

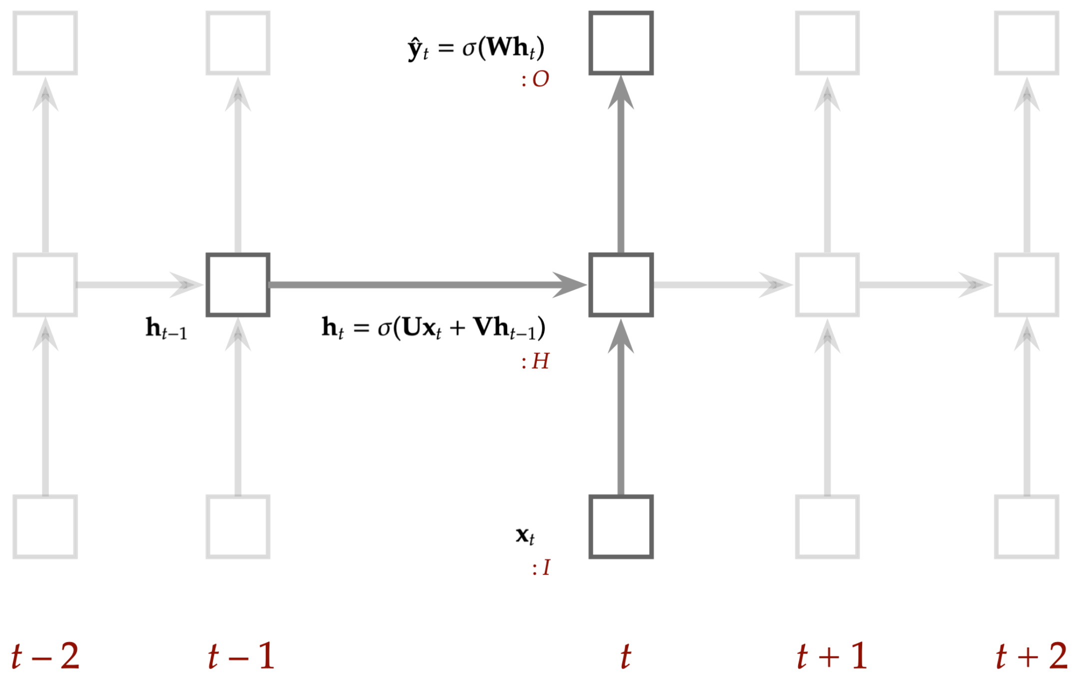
Putting everything together, you should now be able to train and evaluate the HMM. A correct implementation can be expected to reach above **90% test set accuracy** after running the following cell.

In [46]:
```
# Instantiate and train classifier
hmm_tagger = HMMTagger(hf_text_tokenizer, hf_tag_tokenizer)
hmm_tagger.train_all(train_iter, val_iter)

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

```
  0%|          | 0/134 [00:00<?, ?it/s]
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-46-7a490e0a253f> in <cell line: 6>()
      4 
      5 # Evaluate model performance
----> 6 print(f'Training accuracy: {hmm_tagger.evaluate(train_iter):.3f}\n'
      7       f'Test accuracy:     {hmm_tagger.evaluate(test_iter):.3f}')

<ipython-input-45-1c8e9ab5462d> in evaluate(self, iterator)
     69                 words = words[words.ne(self.pad_word_id)]  # remove paddi
ngs
     70                 tags_gold = batch["tag_ids"][sent_id]
---> 71                 tags_pred = self.predict(words)
     72                 for tag_gold, tag_pred in zip(tags_gold, tags_pred):
     73                     if tag_gold == self.pad_state_id:  # stop once we hit
padding

<ipython-input-45-1c8e9ab5462d> in predict(self, words)
     58         ...
     59 
---> 60         return bestpath
     61 
     62     def evaluate(self, iterator):

NameError: name 'bestpath' is not defined
```

RNN for Sequence Labeling

Now let's take an alternative (and more trendy) approach: RNN/LSTM-based sequence labeling. You will 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`

2. `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.

3. `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 methods to predict the tag sequence associated with a sequence of words and to evaluate a full test dataset:

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

In [47]:
```
import torch.optim as optim

class RNNTagger(nn.Module):
    def __init__(self, text_tokenizer, tag_tokenizer, embedding_size, hidden_siz
        super(RNNTagger, self).__init__()
        self.text_tokenizer = text_tokenizer
        self.tag_tokenizer = tag_tokenizer

        self.embedding = nn.Embedding(len(text_tokenizer.get_vocab()), embedding
        self.rnn = nn.RNN(embedding_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, len(tag_tokenizer.get_vocab()))

    def forward(self, text_batch):
        # Embedding layer
        embedded = self.embedding(text_batch)  # (batch_size, max_length, embedd

        # RNN layer
        rnn_out, _ = self.rnn(embedded)  # (batch_size, max_length, hidden_size)
```

```
        # Linear layer
        logits = self.fc(rnn_out)  # (batch_size, max_length, num_tags)

        return logits

    def compute_loss(self, logits, tags):
        # Reshape logits and tags for loss computation
        logits = logits.view(-1, logits.shape[-1])  # (batch_size * max_length,
        tags = tags.view(-1)  # (batch_size * max_length)

        # Loss function
        criterion = nn.CrossEntropyLoss(ignore_index=self.tag_tokenizer.pad_toke
        loss = criterion(logits, tags)

        return loss

    def train_all(self, train_iter, val_iter, epochs=10, learning_rate=0.001):
        optimizer = optim.Adam(self.parameters(), lr=learning_rate)
        best_accuracy = 0

        for epoch in range(epochs):
            self.train()
            total_loss = 0

            for batch in tqdm(train_iter, desc=f"Epoch {epoch+1}/{epochs}"):
                text_batch = batch['input_ids']
                tag_batch = batch['tag_ids']

                optimizer.zero_grad()

                logits = self.forward(text_batch)
                loss = self.compute_loss(logits, tag_batch)
                loss.backward()
                optimizer.step()

                total_loss += loss.item()

            print(f"Epoch {epoch+1}: Training loss = {total_loss:.4f}")

            # Evaluate on validation set
            val_accuracy = self.evaluate(val_iter)
            print(f"Validation accuracy = {val_accuracy:.4f}")

            if val_accuracy > best_accuracy:
                best_accuracy = val_accuracy
                self.best_model = self.state_dict()

    def predict(self, text_batch):
        self.eval()
        with torch.no_grad():
            logits = self.forward(text_batch)  # (batch_size, max_length, num_ta
            predictions = torch.argmax(logits, dim=-1)  # (batch_size, max_lengt
        return predictions

    def evaluate(self, iterator):
        self.eval()
        total, correct = 0, 0

        with torch.no_grad():
            for batch in iterator:
```

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

                predictions = self.predict(text_batch)

                for pred, true in zip(predictions, tag_batch):
                    mask = true != self.tag_tokenizer.pad_token_id
                    correct += (pred[mask] == true[mask]).sum().item()
                    total += mask.sum().item()

        return correct / total if total > 0 else 0
```

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 [48]:
```
# Instantiate and train classifier
rnn_tagger = RNNTagger(hf_text_tokenizer,
                       hf_tag_tokenizer,
                       embedding_size=36,
                       hidden_size=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}')
```

```
Epoch 1/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 1: Training loss = 315.7861
Validation accuracy = 0.7204
Epoch 2/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 2: Training loss = 133.0108
Validation accuracy = 0.8430
Epoch 3/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 3: Training loss = 83.5297
Validation accuracy = 0.8976
Epoch 4/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 4: Training loss = 59.2953
Validation accuracy = 0.9170
Epoch 5/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 5: Training loss = 46.3659
Validation accuracy = 0.9291
Epoch 6/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 6: Training loss = 38.2178
Validation accuracy = 0.9382
Epoch 7/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 7: Training loss = 32.5986
Validation accuracy = 0.9453
Epoch 8/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 8: Training loss = 28.5363
Validation accuracy = 0.9490
Epoch 9/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 9: Training loss = 25.3465
Validation accuracy = 0.9532
Epoch 10/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 10: Training loss = 22.9299
Validation accuracy = 0.9554
Training accuracy: 0.962
Test accuracy:     0.955
```

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`.

Goal 3: Implement an LSTM model

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 [49]:
```
import torch.optim as optim

class LSTMTagger(nn.Module):
    def __init__(self, text_tokenizer, tag_tokenizer, embedding_size, hidden_siz
        super(LSTMTagger, self).__init__()
        self.text_tokenizer = text_tokenizer
        self.tag_tokenizer = tag_tokenizer
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(len(text_tokenizer.get_vocab()), embedding
        self.rnn = nn.LSTM(embedding_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, len(tag_tokenizer.get_vocab()))

    def forward(self, text_batch):
        # Embedding layer
        embedded = self.embedding(text_batch)  # (batch_size, max_length, embedd

        # Init Cell, and Hidden_state
        batch_size = text_batch.size(0)
        h_0 = torch.zeros(1, batch_size, self.hidden_size).to(text_batch.device)
        c_0 = torch.zeros(1, batch_size, self.hidden_size).to(text_batch.device)

        # RNN layer
        lstm_out, (h_n, c_n) = self.rnn(embedded, (h_0, c_0))  # (batch_size, ma

        # Fully connected layer
        logits = self.fc(lstm_out)  # (batch_size, max_length, num_tags)

        return logits

    def compute_loss(self, logits, tags):
        # Reshape logits and tags for loss computation
        logits = logits.view(-1, logits.shape[-1])  # (batch_size * max_length,
        tags = tags.view(-1)  # (batch_size * max_length)

        # Loss function
        criterion = nn.CrossEntropyLoss(ignore_index=self.tag_tokenizer.pad_toke
        loss = criterion(logits, tags)

        return loss

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

```
        best_accuracy = 0

        for epoch in range(epochs):
            self.train()
            total_loss = 0

            for batch in tqdm(train_iter, desc=f"Epoch {epoch+1}/{epochs}"):
                text_batch = batch['input_ids']
                tag_batch = batch['tag_ids']

                optimizer.zero_grad()

                logits = self.forward(text_batch)
                loss = self.compute_loss(logits, tag_batch)
                loss.backward()
                optimizer.step()

                total_loss += loss.item()

            print(f"Epoch {epoch+1}: Training loss = {total_loss:.4f}")

            # Evaluate on validation set
            val_accuracy = self.evaluate(val_iter)
            print(f"Validation accuracy = {val_accuracy:.4f}")

            if val_accuracy > best_accuracy:
                best_accuracy = val_accuracy
                self.best_model = self.state_dict()

    def predict(self, text_batch):
        self.eval()
        with torch.no_grad():
            logits = self.forward(text_batch)  # (batch_size, max_length, num_ta
            predictions = torch.argmax(logits, dim=-1)  # (batch_size, max_lengt
        return predictions

    def evaluate(self, iterator):
        self.eval()
        total, correct = 0, 0

        with torch.no_grad():
            for batch in iterator:
                text_batch = batch['input_ids']
                tag_batch = batch['tag_ids']

                predictions = self.predict(text_batch)

                for pred, true in zip(predictions, tag_batch):
                    mask = true != self.tag_tokenizer.pad_token_id
                    correct += (pred[mask] == true[mask]).sum().item()
                    total += mask.sum().item()

        return correct / total if total > 0 else 0
```

Run the cell below to train an LSTM, and evaluate it. A proper implementation should reach about **95%+ accuracy**.

In [50]:

```
# 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}')
```

```
Epoch 1/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 1: Training loss = 345.4330
Validation accuracy = 0.7080
Epoch 2/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 2: Training loss = 175.5504
Validation accuracy = 0.7471
Epoch 3/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 3: Training loss = 122.7307
Validation accuracy = 0.8303
Epoch 4/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 4: Training loss = 91.5590
Validation accuracy = 0.8608
Epoch 5/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 5: Training loss = 73.3664
Validation accuracy = 0.8912
Epoch 6/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 6: Training loss = 60.5023
Validation accuracy = 0.9145
Epoch 7/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 7: Training loss = 50.7915
Validation accuracy = 0.9238
Epoch 8/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 8: Training loss = 43.7424
Validation accuracy = 0.9319
Epoch 9/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 9: Training loss = 38.1531
Validation accuracy = 0.9357
Epoch 10/10:   0%|          | 0/134 [00:00<?, ?it/s]
Epoch 10: Training loss = 34.0235
Validation accuracy = 0.9421
Training accuracy: 0.948
Test accuracy:     0.941
```

Goal 4: Compare the various models with different amounts of training data

Vary the amount of training data and compare the performance of RNN to LSTM to HMM (if you implemented it). Discuss the pros and cons of the models 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. In addition to experimenting with different amounts of training data, you might want to vary other aspects of the models. We recommend you

> structure your code with useful abstractions to simplify the experimentation and reporting of results.

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

In [51]:
```
ratio = 0.1
test_size = 50

# Set random seeds to make sure subsampling is the same for all models
reseed()

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']

# Subsample
train_data = train_data.shuffle(seed=seed)
train_data = train_data.select(list(range(len(train_data)))[:int(math.floor(len(
test_data = test_data.shuffle(seed=seed)
test_data = test_data.select(list(range(len(test_data)))[:test_size])

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

# Encode data
hf_text_tokenizer = PreTrainedTokenizerFast(tokenizer_object=text_tokenizer, pad

hf_tag_tokenizer = PreTrainedTokenizerFast(tokenizer_object=tag_tokenizer, pad_t

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

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

```
Map:   0%|          | 0/427 [00:00<?, ? examples/s]
Map:   0%|          | 0/572 [00:00<?, ? examples/s]
Map:   0%|          | 0/50 [00:00<?, ? examples/s]
```

In [52]:
```
...
```

Out[52]: Ellipsis

In [53]:
```
...
```

Out[53]: Ellipsis

In [54]:
```
...
```

Out[54]: Ellipsis

In my experiments with RNN and LSTM models, I observed significant differences depending on the size and complexity of the datasets. For smaller datasets, RNNs performed well. Their simple architecture and fewer parameters made them faster to train, and they were less prone to overfitting. However, RNNs struggled with capturing long-term dependencies due to the vanishing gradient problem, limiting their effectiveness on tasks like sequence labeling or text tagging that require understanding relationships between distant tokens in a sequence.

With larger datasets, LSTMs showed clear advantages. Their use of hidden states ($h\ t$ h t) and cell states ($c\ t$ c t) allowed them to manage long-term dependencies and retain important information over time. In sequence labeling tasks, LSTMs consistently produced higher accuracy compared to RNNs, especially as the amount of training data increased. However, LSTMs required more computational resources and took longer to train due to their additional parameters and gate mechanisms.

In my personal project on music generation using large audio datasets, LSTMs were essential. Their ability to maintain context and continuity across long sequences was critical for generating coherent musical structures. This reinforced that while RNNs are efficient for smaller datasets, LSTMs are the superior choice for tasks involving large, complex datasets that require understanding and preserving long-term dependencies.

Prompting Modern Large Language Models (LLMs)

Question: Modern large-scale language models (such as Claude, ChatGPT, Gemini, Llama) have various capabilities, that can be shown by prompting them correctly (i.e. giving them a correct input prompt). For example, they can even be useful for solving text classification, discussed in this segment. Try to see if you can prompt an LLM (of your choosing) to solve the task of sequence labeling on some of the examples of data samples seen in this segment. Write a short paragraph about your experience - what worked better, what worked worse. Note that your not expected to devise a fool-proof prompting method, but only to qualitatively experiment with prompting.

I compared Mistral AI, a French-focused LLM, with ChatGPT for sequence labeling tasks. Mistral AI worked well when labeling French text, especially with clear prompts and examples in IOB format. It produced accurate tags like B-flight_mod or B-fromloc.city_name for sentences like "Quel est le premier vol de Paris à Lyon ?". However, it sometimes struggled with more complex sequences or when the prompts were not detailed enough.

ChatGPT, on the other hand, was more flexible and required fewer examples to perform the same tasks. It handled ambiguous cases better and was more consistent overall. However, its tagging in French was slightly less precise than Mistral AI, especially for tasks requiring domain-specific knowledge.

In conclusion, Mistral AI is better suited for tasks requiring precision in French, while ChatGPT is more robust and easier to use for general or multilingual tasks. This comparison showed me how each model has strengths depending on the language and complexity of the dataset.

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, but you are not restricted to these:

- 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?

The project segment was very well-organized and clear, allowing me to learn a lot through hands-on practice. The steps for implementing and understanding different models, like RNNs and LSTMs, were straightforward and helped me grasp their differences and use cases effectively. I particularly appreciated the opportunity to explore how these models handle sequence labeling tasks in different scenarios, which deepened my understanding of their strengths and weaknesses. Overall, the structure and content of this segment were excellent, and it provided a valuable learning experience.

Instructions for submission of the project segment

This project segment should be submitted to Gradescope, 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://www.gradescope.com/courses/903849?submit_assignment_id=5229511.

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://www.gradescope.com/courses/903849?submit_assignment_id=5229512.

End of project segment 2 {-}