Lab6 Report:

German-to-English Translation with Attention-Mechanism Transformer Model

In [182…

```
!pip uninstall torch torchtext -y
!pip install numpy matplotlib torch==2.2.2 torchtext==0.17.2 seaborn panda
```

```
Found existing installation: torch 2.2.2
Uninstalling torch-2.2.2:
  Successfully uninstalled torch-2.2.2
Found existing installation: torchtext 0.17.2
Uninstalling torchtext-0.17.2:
  Successfully uninstalled torchtext-0.17.2
Requirement already satisfied: numpy in /Users/travishand/.pyenv/versions/
3.10.13/lib/python3.10/site-packages (2.2.5)
Requirement already satisfied: matplotlib in /Users/travishand/.pyenv/versi
ons/3.10.13/lib/python3.10/site-packages (3.10.1)
Collecting torch==2.2.2
  Using cached torch-2.2.2-cp310-none-macosx_11_0_arm64.whl (59.7 MB)
Collecting torchtext==0.17.2
  Using cached torchtext-0.17.2-cp310-cp310-macosx_11_0_arm64.whl (2.1 MB)
Requirement already satisfied: seaborn in /Users/travishand/.pyenv/version
s/3.10.13/lib/python3.10/site-packages (0.13.2)
Requirement already satisfied: pandas in /Users/travishand/.pyenv/versions/
3.10.13/lib/python3.10/site-packages (2.2.3)
Requirement already satisfied: SpaCy in /Users/travishand/.pyenv/versions/
3.10.13/lib/python3.10/site-packages (3.8.5)
Requirement already satisfied: typing-extensions>=4.8.0 in /Users/travishan
d/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from torch==2.2.2)
(4.13.2)
Requirement already satisfied: fsspec in /Users/travishand/.pyenv/versions/
3.10.13/lib/python3.10/site-packages (from torch==2.2.2) (2025.3.2)
Requirement already satisfied: jinja2 in /Users/travishand/.pyenv/versions/
3.10.13/lib/python3.10/site-packages (from torch==2.2.2) (3.1.6)
Requirement already satisfied: networkx in /Users/travishand/.pyenv/version
s/3.10.13/lib/python3.10/site-packages (from torch==2.2.2) (3.4.2)
Requirement already satisfied: sympy in /Users/travishand/.pyenv/versions/
3.10.13/lib/python3.10/site-packages (from torch==2.2.2) (1.14.0)
Requirement already satisfied: filelock in /Users/travishand/.pyenv/version
s/3.10.13/lib/python3.10/site-packages (from torch==2.2.2) (3.18.0)
Requirement already satisfied: requests in /Users/travishand/.pyenv/version
s/3.10.13/lib/python3.10/site-packages (from torchtext==0.17.2) (2.32.3)
Requirement already satisfied: tqdm in /Users/travishand/.pyenv/versions/3.
10.13/lib/python3.10/site-packages (from torchtext==0.17.2) (4.67.1)
Requirement already satisfied: cycler>=0.10 in /Users/travishand/.pyenv/ver
sions/3.10.13/lib/python3.10/site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: python-dateutil>=2.7 in /Users/travishand/.p
yenv/versions/3.10.13/lib/python3.10/site-packages (from matplotlib) (2.9.
0.post0)
Requirement already satisfied: kiwisolver>=1.3.1 in /Users/travishand/.pyen
v/versions/3.10.13/lib/python3.10/site-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /Users/travishand/.pyenv/
versions/3.10.13/lib/python3.10/site-packages (from matplotlib) (25.0)
```

```
Requirement already satisfied: pyparsing>=2.3.1 in /Users/travishand/.pyen
v/versions/3.10.13/lib/python3.10/site-packages (from matplotlib) (3.2.3)
Requirement already satisfied: contourpy>=1.0.1 in /Users/travishand/.pyen
v/versions/3.10.13/lib/python3.10/site-packages (from matplotlib) (1.3.2)
Requirement already satisfied: pillow>=8 in /Users/travishand/.pyenv/versio
ns/3.10.13/lib/python3.10/site-packages (from matplotlib) (11.2.1)
Requirement already satisfied: fonttools>=4.22.0 in /Users/travishand/.pyen
v/versions/3.10.13/lib/python3.10/site-packages (from matplotlib) (4.57.0)
Requirement already satisfied: tzdata>=2022.7 in /Users/travishand/.pyenv/v
ersions/3.10.13/lib/python3.10/site-packages (from pandas) (2025.2)
Requirement already satisfied: pytz>=2020.1 in /Users/travishand/.pyenv/ver
sions/3.10.13/lib/python3.10/site-packages (from pandas) (2025.2)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /Users/travis
hand/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from SpaCy) (1.
0.5)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /Use
rs/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from Sp
aCy) (2.11.4)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /Users/travis
hand/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from SpaCy) (3.
0.12)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in /Users/travishand/.py
env/versions/3.10.13/lib/python3.10/site-packages (from SpaCy) (0.15.3)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /Users/travishand/.p
yenv/versions/3.10.13/lib/python3.10/site-packages (from SpaCy) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /Users/travishand/.py
env/versions/3.10.13/lib/python3.10/site-packages (from SpaCy) (2.5.1)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /Users/travishand/.p
yenv/versions/3.10.13/lib/python3.10/site-packages (from SpaCy) (0.4.1)
Requirement already satisfied: thinc<8.4.0,>=8.3.4 in /Users/travishand/.py
env/versions/3.10.13/lib/python3.10/site-packages (from SpaCy) (8.3.6)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /Users/travishand/.
pyenv/versions/3.10.13/lib/python3.10/site-packages (from SpaCy) (3.0.9)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /Users/travishan
d/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from SpaCy) (2.0.1
0)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /Users/travisha
nd/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from SpaCy) (1.0.1
2)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /Users/travishan
d/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from SpaCy) (3.5.0)
Requirement already satisfied: setuptools in /Users/travishand/.pyenv/versi
ons/3.10.13/lib/python3.10/site-packages (from SpaCy) (65.5.0)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /Users/travishand/.py
env/versions/3.10.13/lib/python3.10/site-packages (from SpaCy) (2.0.11)
Requirement already satisfied: language-data>=1.2 in /Users/travishand/.pye
nv/versions/3.10.13/lib/python3.10/site-packages (from langcodes<4.0.0,>=3.
2.0->SpaCy) (1.3.0)
Requirement already satisfied: annotated-types>=0.6.0 in /Users/travishan
d/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from pydantic!=1.
8,!=1.8.1,<3.0.0,>=1.7.4->SpaCy) (0.7.0)
Requirement already satisfied: typing-inspection>=0.4.0 in /Users/travishan
d/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from pydantic!=1.
8,!=1.8.1,<3.0.0,>=1.7.4->SpaCy) (0.4.0)
Requirement already satisfied: pydantic-core==2.33.2 in /Users/travishand/.
pyenv/versions/3.10.13/lib/python3.10/site-packages (from pydantic!=1.8,!=
1.8.1,<3.0.0,>=1.7.4->SpaCy) (2.33.2)
Requirement already satisfied: six>=1.5 in /Users/travishand/.pyenv/version
s/3.10.13/lib/python3.10/site-packages (from python-dateutil>=2.7->matplotl
ib) (1.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/travishan
```

```
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/travishan
d/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from requests->torc
htext==0.17.2) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /Users/travishand/.pyenv/ver
sions/3.10.13/lib/python3.10/site-packages (from requests->torchtext==0.17.
2) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/travishand/.pye
nv/versions/3.10.13/lib/python3.10/site-packages (from requests->torchtext=
=0.17.2) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /Users/travishand/.pye
nv/versions/3.10.13/lib/python3.10/site-packages (from requests->torchtext=
=0.17.2) (2025.4.26)
Requirement already satisfied: blis<1.4.0,>=1.3.0 in /Users/travishand/.pye
nv/versions/3.10.13/lib/python3.10/site-packages (from thinc<8.4.0,>=8.3.4-
>SpaCy) (1.3.0)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /Users/travishan
d/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from thinc<8.4.0,>=
8.3.4->SpaCy) (0.1.5)
Requirement already satisfied: rich>=10.11.0 in /Users/travishand/.pyenv/ve
rsions/3.10.13/lib/python3.10/site-packages (from typer<1.0.0,>=0.3.0->SpaC
y) (14.0.0)
Requirement already satisfied: shellingham>=1.3.0 in /Users/travishand/.pye
nv/versions/3.10.13/lib/python3.10/site-packages (from typer<1.0.0,>=0.3.0-
>SpaCy) (1.5.4)
Requirement already satisfied: click>=8.0.0 in /Users/travishand/.pyenv/ver
sions/3.10.13/lib/python3.10/site-packages (from typer<1.0.0,>=0.3.0->SpaC
y) (8.1.8)
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /Users/travishan
d/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from weasel<0.5.0,>
=0.1.0->SpaCy) (7.1.0)
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /Users/travish
and/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from weasel<0.5.
0,>=0.1.0->SpaCy) (0.21.0)
Requirement already satisfied: MarkupSafe>=2.0 in /Users/travishand/.pyenv/
versions/3.10.13/lib/python3.10/site-packages (from jinja2->torch==2.2.2)
(3.0.2)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /Users/travishand/.pye
nv/versions/3.10.13/lib/python3.10/site-packages (from sympy->torch==2.2.2)
(1.3.0)
Requirement already satisfied: marisa-trie>=1.1.0 in /Users/travishand/.pye
nv/versions/3.10.13/lib/python3.10/site-packages (from language-data>=1.2->
langcodes<4.0.0,>=3.2.0->SpaCy) (1.2.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in /Users/travishand/.
pyenv/versions/3.10.13/lib/python3.10/site-packages (from rich>=10.11.0->ty
per<1.0.0,>=0.3.0->SpaCy) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /Users/travishan
d/.pyenv/versions/3.10.13/lib/python3.10/site-packages (from rich>=10.11.0-
>typer<1.0.0,>=0.3.0->SpaCy) (2.19.1)
Requirement already satisfied: wrapt in /Users/travishand/.pyenv/versions/
3.10.13/lib/python3.10/site-packages (from smart-open<8.0.0,>=5.2.1->weasel
<0.5.0,>=0.1.0->SpaCy) (1.17.2)
Requirement already satisfied: mdurl~=0.1 in /Users/travishand/.pyenv/versi
ons/3.10.13/lib/python3.10/site-packages (from markdown-it-py>=2.2.0->rich>
=10.11.0->typer<1.0.0,>=0.3.0->SpaCy) (0.1.2)
Installing collected packages: torch, torchtext
^C
```

In [ ]:

```
import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd
```

```
import pandas as pd
import seaborn as sns
from typing import Iterable, List, Callable
from timeit import default_timer as timer
%matplotlib inline
import torch
import torch.nn as nn
from torch.nn import Transformer
from torch import Tensor

from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator
from torch.utils.data import DataLoader
from torch.nn.utils.rnn import pad_sequence

# Initialize your numpy and pytorch random seeds for reproducibility
np.random.seed(88)
torch.manual_seed(88)

# Create a torch.device object to tell pytorch where to store your tensors
# YOUR CODE HERE
```

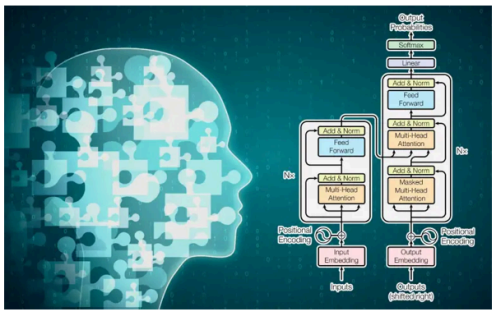
Out[ ]:
```
<torch._C.Generator at 0x113946f70>
```

In [ ]:
```
from IPython.display import Image # For displaying images in colab jupyter
```

In [ ]:
```
Image('lab6_exercise.png', width = 1000)
```

Out[ ]:

Language Translation with Attention-Mechanism Transformer



In this exercise, you will use a Sequence-to-Sequence Transformer model to translate German sentences into English. Your goal is to receive an average validation (Cross-Entropy) loss of less than 2.4.

Before training, it is important to properly tokenize the data. We have worked with language data in the past, but this time the data is more complex and higher-dimensional, and so is the tokenization process. See the example lab for an in-depth look at the tokenization.

During training, we employ the same sliding-window method used in Labs 5 and 6.

After training, demonstrate your model's translation ability by comparing a few of its outputs with the ground-truth labels, along with the inputs.

15

In [ ]:
```
# Seaborn plot styling
sns.set(style = 'white', font_scale = 2)
```

Download data

In [ ]:
```python
# Load the data and store it as a list of tuples: each element in the lis
# Assuming the text file has German and English sentences separated by a 
def load_translation_dataset(filepath: str, delimiter: str = '\t') -> lis
    dataset = []
    with open(filepath, 'r', encoding='utf-8') as f:
        for line in f:
            line = line.strip()
            if not line:
                continue
            parts = line.split(delimiter)
            if len(parts) == 2:
                german, english = parts
                dataset.append((german.strip(), english.strip()))
    return dataset

de_to_en_dataset = load_translation_dataset('de_to_en.txt')
en_to_fr_dataset = load_translation_dataset('en_to_fr.txt')
```

Let's see what the data looks like

In [ ]:
```python
# Print the first ten translated lines of each dataset
print(de_to_en_dataset[:10])
print(en_to_fr_dataset[:10])
```

```
[('Zwei junge weiße Männer sind im Freien in der Nähe vieler Büsche.', 'Two
young, White males are outside near many bushes.'), ('Mehrere Männer mit Sc
hutzhelmen bedienen ein Antriebsradsystem.', 'Several men in hard hats are
operating a giant pulley system.'), ('Ein kleines Mädchen klettert in ein S
pielhaus aus Holz.', 'A little girl climbing into a wooden playhouse.'),
('Ein Mann in einem blauen Hemd steht auf einer Leiter und putzt ein Fenste
r.', 'A man in a blue shirt is standing on a ladder cleaning a window.'),
('Zwei Männer stehen am Herd und bereiten Essen zu.', 'Two men are at the s
tove preparing food.'), ('Ein Mann in grün hält eine Gitarre, während der a
ndere Mann sein Hemd ansieht.', 'A man in green holds a guitar while the ot
her man observes his shirt.'), ('Ein Mann lächelt einen ausgestopften Löwen
an.', 'A man is smiling at a stuffed lion'), ('Ein schickes Mädchen spricht
mit dem Handy während sie langsam die Straße entlangschwebt.', 'A trendy gi
rl talking on her cellphone while gliding slowly down the street.'), ('Eine
Frau mit einer großen Geldbörse geht an einem Tor vorbei.', 'A woman with a
large purse is walking by a gate.'), ('Jungen tanzen mitten in der Nacht au
f Pfosten.', 'Boys dancing on poles in the middle of the night.')]
[('Hi.', 'Salut!'), ('Stop!', 'Arrête-toi !'), ('I won!', "J'ai gagné !"),
('Get up.', 'Lève-toi.'), ('Hop in.', 'Montez.'), ('I paid.', 'J’ai pay
é.'), ('No way!', "Il n'en est pas question !"), ('We won.', 'Nous gagnâme
s.'), ('Be fair.', 'Soyez juste !'), ('Be nice.', 'Soyez gentils !')]
```

Create source and target language tokenizers

But first, what exactly is a *tokenizer*?

A short but incomplete summary is that a tokenizer converts your text/string into a list of numerical values (a list of *tokens*). We performed tokenization in Lab 5 when we converted each alphanumeric character in our text into a number (an index in a

dictionary).

Here, the tokenization is a bit different. Instead of converting each *character* into a number, we want to convert each *word* into a number. As you can imagine, this means the vocabulary of our dataset - the set of unique tokens it contains - will be much larger. There are many more words in English than there are letters! This also means the value of each token will be more unique and meaningful.

Part of tokenizing at the word level is the process of standardizing the text by converting it to lowercase, removing punctuation or special characters, and dealing with contractions or other language-specific features. This is sometimes called *stemming*, reflecting the fact that we want to only extract the *essential meaning* of each word - the "stem" - not necessarily the punctuation, prefixes, suffixes, etc. surrounding it.

Luckily for us, there are some existing Python packages that do this automatically. The below cell downloads two different tokenizers (one each for the source and larget languages), and assigns them to appropriate keys within the "token_transform" dictionary. In Lab 5, you performed tokenization when you converted each character of the text into a specific number.

In [ ]:

```
# Define MACRO - a high-level variable that won't change throughout the du
!python -m spacy download de_core_news_sm
!python -m spacy download en_core_web_sm
!python -m spacy download fr_core_news_sm

# Download the German and English tokenizers, and assign them to appropria
en_tokenizer = get_tokenizer('spacy', language='en_core_web_sm')
de_tokenizer = get_tokenizer('spacy', language='de_core_news_sm')
fr_tokenizer = get_tokenizer('spacy', language='fr_core_news_sm')

global SRC_LANGUAGE, TGT_LANGUAGE, token_transform
# Define MACRO for source and target languages
SRC_LANGUAGE = 'de'
TGT_LANGUAGE = 'en'

# Create a dictionary to hold tokenizers for each language
token_transform = {
    SRC_LANGUAGE: de_tokenizer,
    TGT_LANGUAGE: en_tokenizer
}

# convinience function to switch between the two datasets and languages
def set_languages_and_tokenizers(which):
    if which == 'dte':
        SRC_LANGUAGE = 'de'
        TGT_LANGUAGE = 'en'
        token_transform = {
            SRC_LANGUAGE: de_tokenizer,
            TGT_LANGUAGE: en_tokenizer
        }
    elif which == 'etf':
        SRC_LANGUAGE = 'en'
        TGT_LANGUAGE = 'fr'
```

```
        token_transform = {
            SRC_LANGUAGE: en_tokenizer,
            TGT_LANGUAGE: fr_tokenizer
        }
```

```
A module that was compiled using NumPy 1.x cannot be run in
NumPy 2.2.5 as it may crash. To support both 1.x and 2.x
versions of NumPy, modules must be compiled with NumPy 2.0.
Some module may need to rebuild instead e.g. with 'pybind11>=2.12'.

If you are a user of the module, the easiest solution will be to
downgrade to 'numpy<2' or try to upgrade the affected module.
We expect that some modules will need time to support NumPy 2.

Traceback (most recent call last):  File "/Users/travishand/.pyenv/version
s/3.10.13/lib/python3.10/runpy.py", line 187, in _run_module_as_main
    mod_name, mod_spec, code = _get_module_details(mod_name, _Error)
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/runpy.py",
line 146, in _get_module_details
    return _get_module_details(pkg_main_name, error)
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/runpy.py",
line 110, in _get_module_details
    __import__(pkg_name)
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/spacy/__init__.py", line 6, in <module>
    from .errors import setup_default_warnings
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/spacy/errors.py", line 3, in <module>
    from .compat import Literal
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/spacy/compat.py", line 4, in <module>
    from thinc.util import copy_array
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/thinc/__init__.py", line 5, in <module>
    from .config import registry
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/thinc/config.py", line 5, in <module>
    from .types import Decorator
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/thinc/types.py", line 27, in <module>
    from .compat import cupy, has_cupy
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/thinc/compat.py", line 35, in <module>
    import torch
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/__init__.py", line 1477, in <module>
    from .functional import *  # noqa: F403
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/functional.py", line 9, in <module>
    import torch.nn.functional as F
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/nn/__init__.py", line 1, in <module>
    from .modules import *  # noqa: F403
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/nn/modules/__init__.py", line 35, in <module>
    from .transformer import TransformerEncoder, TransformerDecoder, \
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/nn/modules/transformer.py", line 20, in <module>
    device: torch.device = torch.device(torch._C._get_default_device()),  #
torch.device('cpu'),
```

```
/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/torch/nn/modules/transformer.py:20: UserWarning: Failed to initialize NumPy: _ARRAY_API not found (Triggered internally at /Users/runner/work/pytorch/pytorch/pytorch/torch/csrc/utils/tensor_numpy.cpp:84.)
  device: torch.device = torch.device(torch._C._get_default_device()),  # torch.device('cpu'),
Collecting de-core-news-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/de_core_news_sm-3.8.0/de_core_news_sm-3.8.0-py3-none-any.whl (14.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.6/14.6 MB 97.9 MB/s eta 0:00:0000:0100:01

[notice] A new release of pip is available: 23.0.1 -> 25.1.1
[notice] To update, run: pip install --upgrade pip
✔ Download and installation successful
You can now load the package via spacy.load('de_core_news_sm')

A module that was compiled using NumPy 1.x cannot be run in
NumPy 2.2.5 as it may crash. To support both 1.x and 2.x
versions of NumPy, modules must be compiled with NumPy 2.0.
Some module may need to rebuild instead e.g. with 'pybind11>=2.12'.

If you are a user of the module, the easiest solution will be to
downgrade to 'numpy<2' or try to upgrade the affected module.
We expect that some modules will need time to support NumPy 2.

Traceback (most recent call last):  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/runpy.py", line 187, in _run_module_as_main
    mod_name, mod_spec, code = _get_module_details(mod_name, _Error)
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/runpy.py", line 146, in _get_module_details
    return _get_module_details(pkg_main_name, error)
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/runpy.py", line 110, in _get_module_details
    __import__(pkg_name)
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/spacy/__init__.py", line 6, in <module>
    from .errors import setup_default_warnings
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/spacy/errors.py", line 3, in <module>
    from .compat import Literal
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/spacy/compat.py", line 4, in <module>
    from thinc.util import copy_array
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/thinc/__init__.py", line 5, in <module>
    from .config import registry
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/thinc/config.py", line 5, in <module>
    from .types import Decorator
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/thinc/types.py", line 27, in <module>
    from .compat import cupy, has_cupy
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/thinc/compat.py", line 35, in <module>
    import torch
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/torch/__init__.py", line 1477, in <module>
    from .functional import *  # noqa: F403
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/torch/functional.py", line 9, in <module>
```

```
    import torch.nn.functional as F
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/nn/__init__.py", line 1, in <module>
    from .modules import *  # noqa: F403
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/nn/modules/__init__.py", line 35, in <module>
    from .transformer import TransformerEncoder, TransformerDecoder, \
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/nn/modules/transformer.py", line 20, in <module>
    device: torch.device = torch.device(torch._C._get_default_device()),  #
torch.device('cpu'),
/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/torc
h/nn/modules/transformer.py:20: UserWarning: Failed to initialize NumPy: _A
RRAY_API not found (Triggered internally at /Users/runner/work/pytorch/pyto
rch/pytorch/torch/csrc/utils/tensor_numpy.cpp:84.)
  device: torch.device = torch.device(torch._C._get_default_device()),  # t
orch.device('cpu'),
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/e
n_core_web_sm-3.8.0/en_core_web_sm-3.8.0-py3-none-any.whl (12.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 74.2 MB/s eta 0:
00:00a 0:00:01

[notice] A new release of pip is available: 23.0.1 -> 25.1.1
[notice] To update, run: pip install --upgrade pip
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')

A module that was compiled using NumPy 1.x cannot be run in
NumPy 2.2.5 as it may crash. To support both 1.x and 2.x
versions of NumPy, modules must be compiled with NumPy 2.0.
Some module may need to rebuild instead e.g. with 'pybind11>=2.12'.

If you are a user of the module, the easiest solution will be to
downgrade to 'numpy<2' or try to upgrade the affected module.
We expect that some modules will need time to support NumPy 2.

Traceback (most recent call last):  File "/Users/travishand/.pyenv/version
s/3.10.13/lib/python3.10/runpy.py", line 187, in _run_module_as_main
    mod_name, mod_spec, code = _get_module_details(mod_name, _Error)
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/runpy.py",
line 146, in _get_module_details
    return _get_module_details(pkg_main_name, error)
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/runpy.py",
line 110, in _get_module_details
    __import__(pkg_name)
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/spacy/__init__.py", line 6, in <module>
    from .errors import setup_default_warnings
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/spacy/errors.py", line 3, in <module>
    from .compat import Literal
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/spacy/compat.py", line 4, in <module>
    from thinc.util import copy_array
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/thinc/__init__.py", line 5, in <module>
    from .config import registry
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/thinc/config.py", line 5, in <module>
```

```
    from .types import Decorator
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/thinc/types.py", line 27, in <module>
    from .compat import cupy, has_cupy
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/thinc/compat.py", line 35, in <module>
    import torch
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/__init__.py", line 1477, in <module>
    from .functional import *  # noqa: F403
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/functional.py", line 9, in <module>
    import torch.nn.functional as F
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/nn/__init__.py", line 1, in <module>
    from .modules import *  # noqa: F403
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/nn/modules/__init__.py", line 35, in <module>
    from .transformer import TransformerEncoder, TransformerDecoder, \
  File "/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packa
ges/torch/nn/modules/transformer.py", line 20, in <module>
    device: torch.device = torch.device(torch._C._get_default_device()),  #
torch.device('cpu'),
/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/torc
h/nn/modules/transformer.py:20: UserWarning: Failed to initialize NumPy: _A
RRAY_API not found (Triggered internally at /Users/runner/work/pytorch/pyto
rch/pytorch/torch/csrc/utils/tensor_numpy.cpp:84.)
  device: torch.device = torch.device(torch._C._get_default_device()),  # t
orch.device('cpu'),
Collecting fr-core-news-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/f
r_core_news_sm-3.8.0/fr_core_news_sm-3.8.0-py3-none-any.whl (16.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.3/16.3 MB 96.5 MB/s eta 0:
00:0000:0100:01

[notice] A new release of pip is available: 23.0.1 -> 25.1.1
[notice] To update, run: pip install --upgrade pip
✔ Download and installation successful
You can now load the package via spacy.load('fr_core_news_sm')
```

Let's see what these specific tokenizers do.

In [ ]:
```
# Tokenize the first line of each dataset, and print the tokenized versio
print(token_transform[SRC_LANGUAGE](de_to_en_dataset[0][0]), token_transf
print(en_tokenizer(en_to_fr_dataset[0][0]), fr_tokenizer(en_to_fr_dataset
```

```
['Zwei', 'junge', 'weiße', 'Männer', 'sind', 'im', 'Freien', 'in', 'der',
'Nähe', 'vieler', 'Büsche', '.'] ['Two', 'young', ',', 'White', 'males', 'a
re', 'outside', 'near', 'many', 'bushes', '.']
['Hi', '.'] ['Salut', '!']
```

Create a vocabulary for each language's dataset

In Lab 5, we did this with a simple dictionary that mapped each character to an integer (and vice versa). However, PyTorch has a built-in dictionary object that provides some extra functionality.

We will create this object using torchtext.data.build_vocab_from_iterator(). This function takes an iterator as input and returns a torchtext.vocab.Vocab object. This is a dictionary-like object that maps tokens to indices, but where it differs from a normal dictionary, is that its indices are assigned based on the frequency of the tokens in the dataset. For example, the most frequent token gets the index 0, the second most frequent gets the index 1, and so on. This frequency-index mapping saves a bunch of compute time and resources.

Moreover, this time we will also have the four "special" tokens, that will always be assigned to the first four indices.

In [ ]:

```
# Define a helper function that converts a list of strings into a list of
def tokenize_text(text: str, tokenizer) -> List[str]:
    return tokenizer(text)

# Define your special tokens and their indeces in your vocabulary
SPECIAL_TOKENS = ['<unk>', '<pad>', '<bos>', '<eos>']
UNK_IDX, PAD_IDX, BOS_IDX, EOS_IDX = range(4)

# Step 3: Helper to yield tokens from iterator
def yield_tokens(data_iter, language):
    for src, tgt in data_iter:
        yield token_transform[language](src if language == SRC_LANGUAGE e

# Define your vocabulary for each language using the build_vocab_from_ite
de_to_en_vocab_transform = {}
for lang in [SRC_LANGUAGE, TGT_LANGUAGE]:
    de_to_en_vocab = build_vocab_from_iterator(
        yield_tokens(de_to_en_dataset, lang),
        min_freq=1,
        specials=SPECIAL_TOKENS,
        special_first=True
    )
    # Set ``UNK_IDX`` as the default index.
    de_to_en_vocab.set_default_index(UNK_IDX)
    de_to_en_vocab_transform[lang] = de_to_en_vocab

# Let's see the first 20 words in each vocabulary
print(f"First 20 tokens in German vocab: {de_to_en_vocab_transform[SRC_LAN
print(f"First 20 tokens in English vocab: {de_to_en_vocab_transform[TGT_LA

# <!--------------------------------------------------------------------

# # Repeat for en_to_fr dataset

# set_languages_and_tokenizers('etf')
SRC_LANGUAGE = 'en'
TGT_LANGUAGE = 'fr'
token_transform = {
    SRC_LANGUAGE: en_tokenizer,
    TGT_LANGUAGE: fr_tokenizer
}
en_to_fr_vocab_transform = {}
# Define your vocabulary for each language using the build_vocab_from_ite
for lang in [SRC_LANGUAGE, TGT_LANGUAGE]:
    en_to_fr_vocab = build_vocab_from_iterator(
```

```
    en_to_fr_vocab = build_vocab_from_iterator(
        yield_tokens(en_to_fr_dataset, lang),
        min_freq=1,
        specials=SPECIAL_TOKENS,
        special_first=True
    )
    # Set ``UNK_IDX`` as the default index.
    en_to_fr_vocab.set_default_index(UNK_IDX)
    en_to_fr_vocab_transform[lang] = en_to_fr_vocab

# Let's see the first 20 words in each vocabulary

print(f"First 20 tokens in English vocab: {en_to_fr_vocab_transform[SRC_LA
print(f"First 20 tokens in French vocab: {en_to_fr_vocab_transform[TGT_LAN

CURRENT_DATASET = 'en_to_fr'
```

```
First 20 tokens in German vocab: ['<unk>', '<pad>', '<bos>', '<eos>', '.',
'Ein', 'einem', 'in', 'und', ',', 'mit', 'auf', 'Mann', 'einer', 'Eine', 'e
in', 'der', 'Frau', 'eine', 'die']
First 20 tokens in English vocab: ['<unk>', '<pad>', '<bos>', '<eos>', 'a',
'.', 'A', 'in', 'the', 'on', 'is', 'and', 'man', 'of', 'with', ',', 'woma
n', 'are', 'to', 'Two']
First 20 tokens in English vocab: ['<unk>', '<pad>', '<bos>', '<eos>', '.',
'I', 'you', 'to', '?', 'the', "n't", 'a', 'is', 'do', 'Tom', "'s", 'that',
'of', 'You', 'in']
First 20 tokens in French vocab: ['<unk>', '<pad>', '<bos>', '<eos>', '.',
'de', 'Je', '?', 'pas', 'est', 'que', 'à', 'ne', 'la', 'le', 'a', 'Tom',
"n'", 'un', 'Il']
```

Train-Validate-Test split

In [ ]:

```
import random
random.seed(88)

random.shuffle(de_to_en_dataset)

dte_data_size = len(de_to_en_dataset)
dte_train_size, dte_validation_size, dte_test_size = int(dte_data_size * (
dte_train_dataset, dte_validation_dataset, dte_test_dataset = de_to_en_da

# Check the size of each data set
print("Checking de to en dataset sizes...")
print(f"Total size: {dte_data_size}",)
print(f"Train size: {len(dte_train_dataset)}",
      f"Validation size: {len(dte_validation_dataset)}",
      f"Test size: {len(dte_test_dataset)}",
      f"Sum of all matches original dataset: {len(dte_train_dataset) + le


# <!-----------------------------------------------------------------------
print()

# # Repeat for en_to_fr dataset

# Shuffle the text pairs
random.shuffle(en_to_fr_dataset)

# Let's go for a 70-20-10 train-val-test split
```

```
etf_data_size = len(en_to_fr_dataset)
etf_train_size, etf_validation_size, etf_test_size = int(etf_data_size * (
etf_train_dataset, etf_validation_dataset, etf_test_dataset = en_to_fr_da

# Check the size of each data set
print("Checking en to fr dataset sizes...")
print(f"Total size: {etf_data_size}",)
print(f"Train size: {len(etf_train_dataset)}",
      f"Validation size: {len(etf_validation_dataset)}",
      f"Test size: {len(etf_test_dataset)}",
      f"Sum of all matches original dataset: {len(etf_train_dataset) + le
```

```
Checking de to en dataset sizes...
Total size: 31013
Train size: 21709 Validation size: 6202 Test size: 3102 Sum of all matches
original dataset: True

Checking en to fr dataset sizes...
Total size: 17563
Train size: 12294 Validation size: 3512 Test size: 1757 Sum of all matches
original dataset: True
```

Mask functions

The mask function plays an essential role in the training of a transformer model, specifically during the pre-training phase when the model learns to understand and generate language. The two main purposes of the mask function are:

1. To facilitate self-attention mechanism: Transformers use self-attention mechanisms to identify relationships between words in a sequence. Masking is used to prevent the model from "cheating" by looking at future tokens when trying to predict the current token. In other words, the mask function ensures that the model only attends to the current token and the previous tokens, not the future tokens, during the training process.

2. To enable masked language modeling (MLM): Masked language modeling is a popular pre-training objective used in transformer-based models like BERT. In MLM, a certain percentage of input tokens are randomly masked (usually around 15%), and the model is tasked with predicting the original tokens at these masked positions. The mask function serves as a way of hiding the original token from the model, forcing it to learn contextual representations that can help it predict the masked tokens accurately.

The use of the mask function in both self-attention and MLM helps the transformer model learn meaningful context-dependent representations, making it more effective at understanding and generating natural language.

In [ ]:

```
# Define your masking function
def create_mask(src: Tensor, tgt: Tensor) -> tuple[Tensor, Tensor]:
    src_mask = (src != PAD_IDX).unsqueeze(-2)
    tgt_mask = (tgt != PAD_IDX).unsqueeze(-2)
    tgt_len = tgt.size(1)
```

```
    tgt_len = tgt.size(1)
    nopeak_mask = torch.triu(torch.ones((1, tgt_len, tgt_len)), diagonal=1
    tgt_mask = tgt_mask & nopeak_mask
    return src_mask, tgt_mask
```

Collation

The collation function is what converts our strings into batches of tensors that can be processed by our model, based on the vocabularies and tokenization functions we have built up thus far.

Again, this is something we can do manually, but at some point the data transformations get so complicated that we might as well put them all into a function. Moreover, defining our transformation as a *function* allows us to use some more built-in PyTorch functionality that makes our jobs a whole lot easier. See: torch.utils.data.DataLoader.

In [183…

```
SRC_LANGUAGE = 'de'
TGT_LANGUAGE = 'en'
token_transform = {
    SRC_LANGUAGE: de_tokenizer,
    TGT_LANGUAGE: en_tokenizer
}

# Define helper function to club together sequential operations
def sequential_transforms(*transforms: Callable) -> Callable:
    def func(txt_input):
        for transform in transforms:
            txt_input = transform(txt_input)
        return txt_input
    return func

# Define function to add BOS/EOS and create a tensor for input sequence in
BOS_IDX = de_to_en_vocab_transform[SRC_LANGUAGE]['<bos>']
EOS_IDX = de_to_en_vocab_transform[SRC_LANGUAGE]['<eos>']

def tensor_transform(token_ids: List[int]) -> torch.Tensor:
    return torch.tensor([BOS_IDX] + token_ids + [EOS_IDX], dtype=torch.lon

# Define your ``src`` and ``tgt`` language text transforms to convert raw
# Compose transforms for both languages
dte_text_transform = {}
for ln in [SRC_LANGUAGE, TGT_LANGUAGE]:
    dte_text_transform[ln] = sequential_transforms(
        token_transform[ln],              # Tokenizer
        de_to_en_vocab_transform[ln],             # Token -> Index
        tensor_transform                  # Add BOS/EOS + tensor
    )

# Define your "collation" function to collate data samples into batch tens
def dte_collate_fn(batch):
    src_batch, tgt_batch = [], []
    for src_sample, tgt_sample in batch:
        src_batch.append(dte_text_transform[SRC_LANGUAGE](src_sample))
        tgt_batch.append(dte_text_transform[TGT_LANGUAGE](tgt_sample))
```

```
    src_batch = pad_sequence(src_batch, padding_value=PAD_IDX, batch_firs
    tgt_batch = pad_sequence(tgt_batch, padding_value=PAD_IDX, batch_firs

    return src_batch, tgt_batch

# <!--------------------------------------------------------------------

# Repeat for en_to_fr dataset
SRC_LANGUAGE = 'en'
TGT_LANGUAGE = 'fr'
token_transform = {
    SRC_LANGUAGE: en_tokenizer,
    TGT_LANGUAGE: fr_tokenizer
}

# Define your ``src`` and ``tgt`` language text transforms to convert raw
# Compose transforms for both languages
etf_text_transform = {}
for ln in [SRC_LANGUAGE, TGT_LANGUAGE]:
    etf_text_transform[ln] = sequential_transforms(
        token_transform[ln],              # Tokenizer
        en_to_fr_vocab_transform[ln],             # Token -> Index
        tensor_transform                  # Add BOS/EOS + tensor
    )

# Define your "collation" function to collate data samples into batch tens
def etf_collate_fn(batch):
    src_batch, tgt_batch = [], []
    for src_sample, tgt_sample in batch:
        src_batch.append(etf_text_transform[SRC_LANGUAGE](src_sample))
        tgt_batch.append(etf_text_transform[TGT_LANGUAGE](tgt_sample))

    src_batch = pad_sequence(src_batch, padding_value=PAD_IDX, batch_firs
    tgt_batch = pad_sequence(tgt_batch, padding_value=PAD_IDX, batch_firs

    return src_batch, tgt_batch

SRC_LANGUAGE = 'de'
TGT_LANGUAGE = 'en'
token_transform = {
    SRC_LANGUAGE: de_tokenizer,
    TGT_LANGUAGE: en_tokenizer
}
```

Define training, evaluation functions

Modularization is the name of the game.

Not only does this help us here, but if you ever need to train a similar model in the future, you can simply import the ones defined here!

For example, imagine this was a Python script and not a notebook, and the filename was "german_to_english_transformer.py" Then, in whichever future script or notebook you wish to use these functions, you could simply call: "from german_to_english_transformer import train_epoch, evaluate"

In [184…

```
from torch.optim.lr_scheduler import StepLR

# Define a function to train the model for a single epoch
def train_epoch(model, optimizer, loss_fn, dataloader, device):
    model.train()
    total_loss = 0
    scheduler = StepLR(optimizer, step_size=1, gamma=0.95)

    for src, tgt in dataloader:
        src, tgt = src.to(device), tgt.to(device)

        tgt_input = tgt[:, :-1]
        tgt_output = tgt[:, 1:]

        src_mask, tgt_mask = create_mask(src, tgt_input)

        optimizer.zero_grad()

        # Forward pass
        logits = model(src, tgt_input, src_mask, tgt_mask)

        # Compute loss
        loss = loss_fn(logits.view(-1, logits.size(-1)), tgt_output.view(-
        total_loss += loss.item()

        # Backward pass and optimization
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()
        scheduler.step()

    return total_loss / len(dataloader)

# Define a function to evaluate the model
def evaluate(model, loss_fn, dataloader, device):
    model.eval()
    total_loss = 0

    with torch.no_grad():
        for src, tgt in dataloader:
            src, tgt = src.to(device), tgt.to(device)

            tgt_input = tgt[:, :-1]
            tgt_output = tgt[:, 1:]

            src_mask, tgt_mask = create_mask(src, tgt_input)

            # Forward pass
            logits = model(src, tgt_input, src_mask, tgt_mask)

            # Compute loss
            loss = loss_fn(logits.view(-1, logits.size(-1)), tgt_output.vi
            total_loss += loss.item()

    return total_loss / len(dataloader)
```

Define model

Define model

In [ ]:

```
# Define the PositionalEncoding module that quantifies the relative positi
# Notice that this is not actually an MLP or neural network, i.e. it has n
# it is just a function that you could represent analytically, if you want
class PositionalEncoding(nn.Module):
    def __init__(self, emb_size: int, dropout: float = 0.1, maxlen: int =
        super(PositionalEncoding, self).__init__()
        den = torch.exp(- torch.arange(0, emb_size, 2) * math.log(10000) /
        pos = torch.arange(0, maxlen).reshape(maxlen, 1)
        pos_embedding = torch.zeros((maxlen, emb_size))
        pos_embedding[:, 0::2] = torch.sin(pos * den)
        pos_embedding[:, 1::2] = torch.cos(pos * den)
        pos_embedding = pos_embedding.unsqueeze(0)  # shape: [1, maxlen, e

        self.dropout = nn.Dropout(dropout)
        self.register_buffer('pos_embedding', pos_embedding)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        # x: [batch_size, seq_len, emb_size]
        x = x + self.pos_embedding[:, :x.size(1), :]
        return self.dropout(x)

# Define the TokenEmbedding module converts a tensor of vocabulary-indices
# Also not a neural network, but a lookup table
class TokenEmbedding(nn.Module):
    def __init__(self, vocab_size: int, emb_size: int):
        super(TokenEmbedding, self).__init__()
        self.embedding = nn.Embedding(vocab_size, emb_size)
        self.emb_size = emb_size

    def forward(self, tokens: torch.Tensor) -> torch.Tensor:
        # tokens: [batch_size, seq_len]
        return self.embedding(tokens) * math.sqrt(self.emb_size)


# Define the actual seq2seq transformer model
# Question: What are we "transforming" between?
class TransformerModel(nn.Module):
    def __init__(self, num_encoder_layers: int, num_decoder_layers: int, e
                 nhead: int, src_vocab_size: int, tgt_vocab_size: int, dim
        super(TransformerModel, self).__init__()

        self.transformer = nn.Transformer(d_model=emb_size,
                                          nhead=nhead,
                                          num_encoder_layers=num_encoder_l
                                          num_decoder_layers=num_decoder_l
                                          dim_feedforward=dim_feedforward,
                                          dropout=dropout,
                                          batch_first=batch_first)

        self.generator = nn.Linear(emb_size, tgt_vocab_size)

        self.src_tok_emb = TokenEmbedding(src_vocab_size, emb_size)
        self.tgt_tok_emb = TokenEmbedding(tgt_vocab_size, emb_size)
        self.positional_encoding = PositionalEncoding(emb_size, dropout=dr

    def forward(self, src, tgt, src_mask, tgt_mask,
                src_padding_mask, tgt_padding_mask, memory_key_padding_mas
        src_emb = self.positional_encoding(self.src_tok_emb(src))
```

```
        tgt_emb = self.positional_encoding(self.tgt_tok_emb(tgt))
        outs = self.transformer(src_emb, tgt_emb, src_mask, tgt_mask, None
                                src_padding_mask, tgt_padding_mask, memory
        return self.generator(outs)

    def encode(self, src, src_mask=None, src_key_padding_mask=None):
        src_emb = self.positional_encoding(self.src_tok_emb(src))
        return self.transformer.encoder(src_emb, mask=src_mask, src_key_pa

    def decode(self, tgt, memory, tgt_mask=None, tgt_key_padding_mask=None
        tgt_emb = self.positional_encoding(self.tgt_tok_emb(tgt))
        return self.transformer.decoder(tgt_emb, memory, tgt_mask=tgt_mask
                                        tgt_key_padding_mask=tgt_key_paddi
                                        memory_key_padding_mask=memory_key
```

In [186…

```
# define loss function

from torch.nn.functional import log_softmax

class LabelSmoothingLoss(nn.Module):
    def __init__(self, label_smoothing, tgt_vocab_size, ignore_index=PAD_I
        super().__init__()
        self.criterion = nn.KLDivLoss(reduction="batchmean")
        self.padding_idx = ignore_index
        self.confidence = 1.0 - label_smoothing
        self.smoothing = label_smoothing
        self.tgt_vocab_size = tgt_vocab_size

    def forward(self, pred, target):
        pred = log_softmax(pred, dim=-1)

        true_dist = torch.zeros_like(pred)
        true_dist.fill_(self.smoothing / (self.tgt_vocab_size - 2))
        true_dist.scatter_(1, target.data.unsqueeze(1), self.confidence)
        true_dist[:, self.padding_idx] = 0
        mask = torch.nonzero(target.data == self.padding_idx)
        if mask.dim() > 0:
            true_dist.index_fill_(0, mask.squeeze(), 0.0)

        return self.criterion(pred, true_dist)
```

Question #2: What's the significance of the "num_heads" parameter in the **init** function of the Seq2SeqTransformer above?

Each head lets the model learn a different part of the input sequence, running 'num_heads' times in parallel for each layer with differents weights and then concatenates them together and passing them through another linear layer for the final result.

Question #3: In less detail, state the significance of these other two parameters:

1. embedding_size: the size of the embedding dimension determines how much meaning the model can assign to each token in the vocab. higher embedding dimension means a deeper contextual understanding, but longer training times.
2. src_vocab_size: the number of unique tokens in the vocabulary that the model will learn.

Define hyperparameters

In [197…

```
# Define your hyperparameters
lr = 0.001
epochs = 50
batch_size = 32
embedding_dim = 512
num_heads = 8
src_vocab_size = len(de_to_en_vocab_transform[SRC_LANGUAGE])
tgt_vocab_size = len(de_to_en_vocab_transform[TGT_LANGUAGE])
num_encoder_layers = 6
num_decoder_layers = 6
dropout = 0.3

# Define your model, loss function, and optimizer
model = TransformerModel(num_encoder_layers=num_encoder_layers,
                         num_decoder_layers=num_decoder_layers,
                         emb_size=embedding_dim,
                         nhead=num_heads,
                         src_vocab_size=src_vocab_size,
                         tgt_vocab_size=tgt_vocab_size,
                         batch_first=True,
                         dropout=dropout)
loss_fn = LabelSmoothingLoss(tgt_vocab_size=tgt_vocab_size, ignore_index=I
optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=1e-5

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
if torch.cuda.is_available():
    model = model.to(device)
    loss_fn = loss_fn.to(device)
```

Identify tracked values

In [188…

```
# Track training loss and validation accuracy
train_loss_vals = []
val_accuracy_vals = []
```

Train the model

In [198…

```
import time
# Create dataloaders for training, validation, and testing
# German to English
```

```
dte_train_dataloader = DataLoader(dte_train_dataset, batch_size=batch_size
dte_val_dataloader   = DataLoader(dte_validation_dataset, batch_size=batch
dte_test_dataloader  = DataLoader(dte_test_dataset, batch_size=batch_size

# English to French
etf_train_dataloader = DataLoader(etf_train_dataset, batch_size=batch_size
etf_val_dataloader   = DataLoader(etf_validation_dataset, batch_size=batch
etf_test_dataloader  = DataLoader(etf_test_dataset, batch_size=batch_size

for src, tgt in dte_train_dataloader:
    print("src shape:", src.shape)  # should be [batch_size, src_seq_len]
    print("tgt shape:", tgt.shape)  # should be [batch_size, tgt_seq_len]
    break


# Train your model
NUM_EPOCHS = 10
LEARNING_RATE = 0.0005

# === Loss and Optimizer ===
loss_fn = nn.CrossEntropyLoss(ignore_index=PAD_IDX)
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)

# === Mask Function ===
def generate_square_subsequent_mask(sz):
    return torch.triu(torch.full((sz, sz), float('-inf')), diagonal=1)

# === Training and Evaluation Functions ===
def train_epoch(model, optimizer, dataloader):
    # Set the model to training mode
    model.train()
    total_loss = 0

    for src, tgt in dataloader:
        # Move data to the device
        src = src.to(device)
        tgt = tgt.to(device)

        # Prepare the target input and output
        tgt_input = tgt[:, :-1]
        tgt_output = tgt[:, 1:]

        # Create masks
        src_mask = None
        tgt_mask = generate_square_subsequent_mask(tgt_input.size(1)).to(

        # Create padding masks
        src_padding_mask = (src == PAD_IDX).to(device)
        tgt_padding_mask = (tgt_input == PAD_IDX).to(device)

        # Forward pass
        logits = model(src, tgt_input, src_mask, tgt_mask,
                       src_padding_mask, tgt_padding_mask, src_padding_ma

        # Compute loss
        optimizer.zero_grad()

        # Reshape logits and target output for loss calculation
        loss = loss_fn(logits.reshape(-1, logits.shape[-1]), tgt_output.re

        # Backward pass and optimization
```

```
        loss.backward()

        # Clip gradients to prevent exploding gradients
        optimizer.step()

        total_loss += loss.item()

    return total_loss / len(dataloader)

def evaluate(model, dataloader):
    # Set the model to evaluation mode
    model.eval()
    total_loss = 0

    with torch.no_grad():
        for src, tgt in dataloader:
            # Move data to the device
            src = src.to(device)
            tgt = tgt.to(device)

            # Prepare the target input and output
            tgt_input = tgt[:, :-1]
            tgt_output = tgt[:, 1:]

            # Create masks
            src_mask = None
            tgt_mask = generate_square_subsequent_mask(tgt_input.size(1))

            # Create padding masks
            src_padding_mask = (src == PAD_IDX).to(device)
            tgt_padding_mask = (tgt_input == PAD_IDX).to(device)

            # Forward pass
            logits = model(src, tgt_input, src_mask, tgt_mask,
                           src_padding_mask, tgt_padding_mask, src_padding

            # Compute loss
            loss = loss_fn(logits.reshape(-1, logits.shape[-1]), tgt_outpu
            total_loss += loss.item()

    return total_loss / len(dataloader)

# === Training Loop ===
print("Starting training for German to English translation...")
for epoch in range(1, NUM_EPOCHS + 1):
    start_time = time.time()
    train_loss = train_epoch(model, optimizer, dte_train_dataloader)
    train_loss_vals.append(train_loss)
    val_loss = evaluate(model, dte_val_dataloader)
    val_accuracy_vals.append(val_loss)
    end_time = time.time()

    print(f"Epoch: {epoch}, Train loss: {train_loss:.4f}, Val loss: {val_
dte_train_loss_vals = train_loss_vals
dte_val_accuracy_vals = val_accuracy_vals
print("Training complete!")

print("-----------------------------------------------------------")
```

```
src shape: torch.Size([32, 24])
```

```
tgt shape: torch.Size([32, 24])
Starting training for German to English translation...
Epoch: 1, Train loss: 5.6634, Val loss: 9.7061, Epoch time = 349.53s
Epoch: 2, Train loss: 5.3197, Val loss: 9.7938, Epoch time = 353.00s
Epoch: 3, Train loss: 5.2591, Val loss: 11.4878, Epoch time = 357.40s
Epoch: 4, Train loss: 5.2232, Val loss: 11.2084, Epoch time = 354.08s
Epoch: 5, Train loss: 5.1983, Val loss: 11.2284, Epoch time = 358.86s
Epoch: 6, Train loss: 5.1786, Val loss: 11.5082, Epoch time = 363.99s
Epoch: 7, Train loss: 5.1682, Val loss: 12.3310, Epoch time = 370.43s
Epoch: 8, Train loss: 5.1560, Val loss: 11.4982, Epoch time = 370.10s
Epoch: 9, Train loss: 5.1483, Val loss: 12.0916, Epoch time = 370.51s
Epoch: 10, Train loss: 5.1465, Val loss: 11.3036, Epoch time = 370.02s
Training complete!
------------------------------------------------------------
```

In [ ]:

```
SRC_LANGUAGE = 'en'
TGT_LANGUAGE = 'fr'
token_transform = {
    SRC_LANGUAGE: en_tokenizer,
    TGT_LANGUAGE: fr_tokenizer
}
print("Starting training for English to French translation...")
for epoch in range(1, NUM_EPOCHS + 1):
    start_time = time.time()
    train_loss = train_epoch(model, optimizer, etf_train_dataloader)
    train_loss_vals.append(train_loss)
    val_loss = evaluate(model, etf_val_dataloader)
    val_accuracy_vals.append(val_loss)
    end_time = time.time()

    print(f"Epoch: {epoch}, Train loss: {train_loss:.4f}, Val loss: {val_
etf_train_loss_vals = train_loss_vals
etf_val_accuracy_vals = val_accuracy_vals
print("Training complete!")
```

```
Starting training for English to French translation...
/Users/travishand/.pyenv/versions/3.10.13/lib/python3.10/site-packages/torc
h/nn/functional.py:5109: UserWarning: Support for mismatched key_padding_ma
sk and attn_mask is deprecated. Use same type for both instead.
  warnings.warn(
Epoch: 1, Train loss: 5.6158, Val loss: 9.5840, Epoch time = 155.47s
Epoch: 2, Train loss: 5.4628, Val loss: 9.1109, Epoch time = 150.97s
Epoch: 3, Train loss: 5.4220, Val loss: 9.3349, Epoch time = 294.68s
Epoch: 4, Train loss: 5.3972, Val loss: 10.2666, Epoch time = 628.04s
Epoch: 5, Train loss: 5.3775, Val loss: 8.6228, Epoch time = 551.28s
Epoch: 6, Train loss: 5.3640, Val loss: 10.3624, Epoch time = 1554.69s
Epoch: 7, Train loss: 5.3608, Val loss: 8.2745, Epoch time = 931.89s
Epoch: 8, Train loss: 5.3484, Val loss: 8.3942, Epoch time = 1418.54s
Epoch: 9, Train loss: 5.3403, Val loss: 9.0782, Epoch time = 542.22s
Epoch: 10, Train loss: 5.3360, Val loss: 8.8580, Epoch time = 347.99s
Training complete!
```

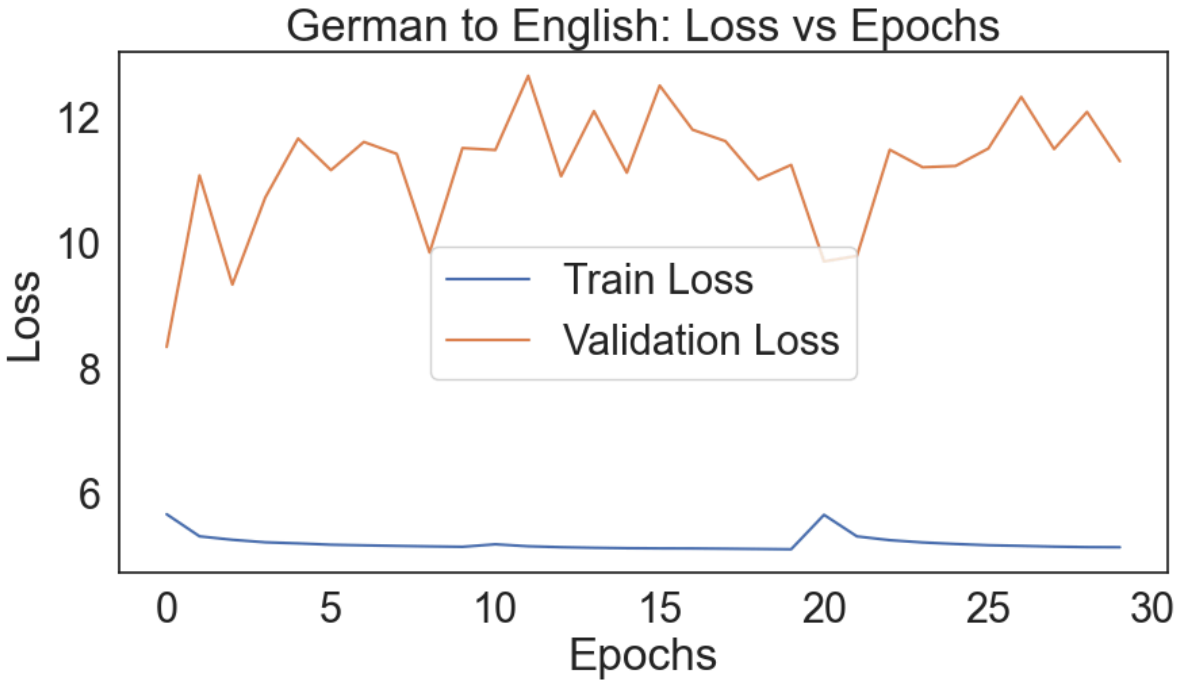
Visualize and Evaluate the model

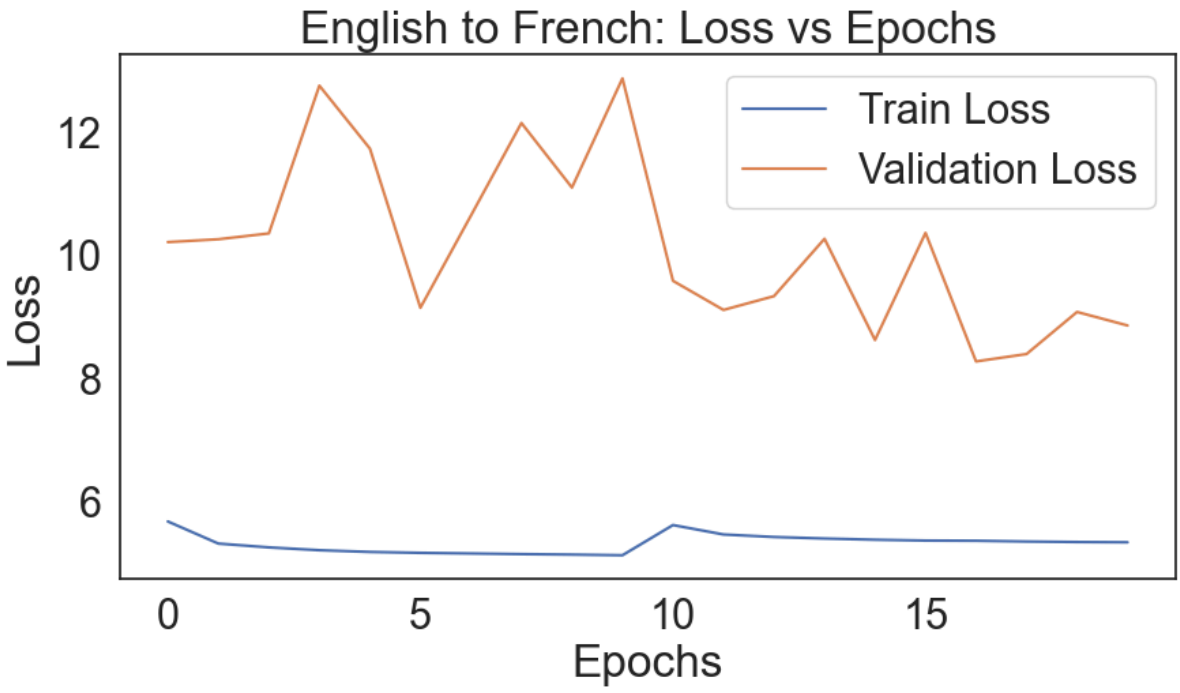
In [199…

```
# Plot the loss
# YOUR CODE HERE
plt.figure(figsize=(10, 5))
```

```
plt.plot(dte_train_loss_vals, label='Train Loss')
plt.plot(dte_val_accuracy_vals, label='Validation Loss')
plt.title('German to English: Loss vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(etf_train_loss_vals, label='Train Loss')
plt.plot(etf_val_accuracy_vals, label='Validation Loss')
plt.title('English to French: Loss vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





In [ ]:

```
# Define a decode function to generate output sequence using greedy algori
# This basically saves us some compute time by taking a bunch of shortcuts
# YOUR CODE HERE
def greedy_decode(model, src, src_key_padding_mask, max_len, start_symbol
    memory = model.encode(src, src_mask=None, src_key_padding_mask=src_key
    ys = torch.ones(1, 1).fill_(start_symbol).type_as(src.data)

    for _ in range(max_len - 1):
        tgt_mask = generate_square_subsequent_mask(ys.size(1)).type_as(src
        out = model.decode(ys, memory, tgt_mask=tgt_mask)
        prob = model.generator(out[:, -1])
        _, next_word = torch.max(prob, dim=1)
        next_word = next_word.item()
        ys = torch.cat([ys, torch.ones(1, 1).fill_(next_word).type_as(src

    return ys


# Define a translation function that actually uses the model to translate
def translate_sentence(model, sentence, src_vocab_transform, tgt_vocab_tra
    model.eval()
    with torch.no_grad():
        # Tokenize and numericalize input sentence
        tokens = token_transform[SRC_LANGUAGE](sentence)
        src_indices = [src_vocab_transform[SRC_LANGUAGE][token] for token
        src_tensor = torch.tensor(src_indices, dtype=torch.long).unsqueeze

        # Create key padding mask instead of src_mask
        src_key_padding_mask = (src_tensor == PAD_IDX)

        # Generate the output sequence
        output = greedy_decode(model, src_tensor, src_key_padding_mask, ma

        # Convert the output tensor to tokens
        output_tokens = [tgt_vocab_transform[TGT_LANGUAGE].lookup_token(to

        # Remove <bos> and <eos> tokens
        output_tokens = output_tokens[1:]  # optionally trim <eos> if pres

    return ' '.join(output_tokens)


# Test the translation function
test_sentence = "Das ist ein Test."
translated_sentence = translate_sentence(model, test_sentence, de_to_en_vo
print(f"Input: {test_sentence}")
print(f"Translated: {translated_sentence}")

print("--------------------------------------------------------------")

set_languages_and_tokenizers('etf')

# Test the translation function for English to French
test_sentence = "This is a test."
translated_sentence = translate_sentence(model, test_sentence, en_to_fr_vo
print(f"Input: {test_sentence}")
print(f"Translated: {translated_sentence}")
```

```
Das eis un Test.
```

```
This is a Test.
--------------------------------------------------------
This is a test.
C'est un test.
```

Let's try the model out on a few of our test sequences. Print the first 10 target/translated sequences from our test set